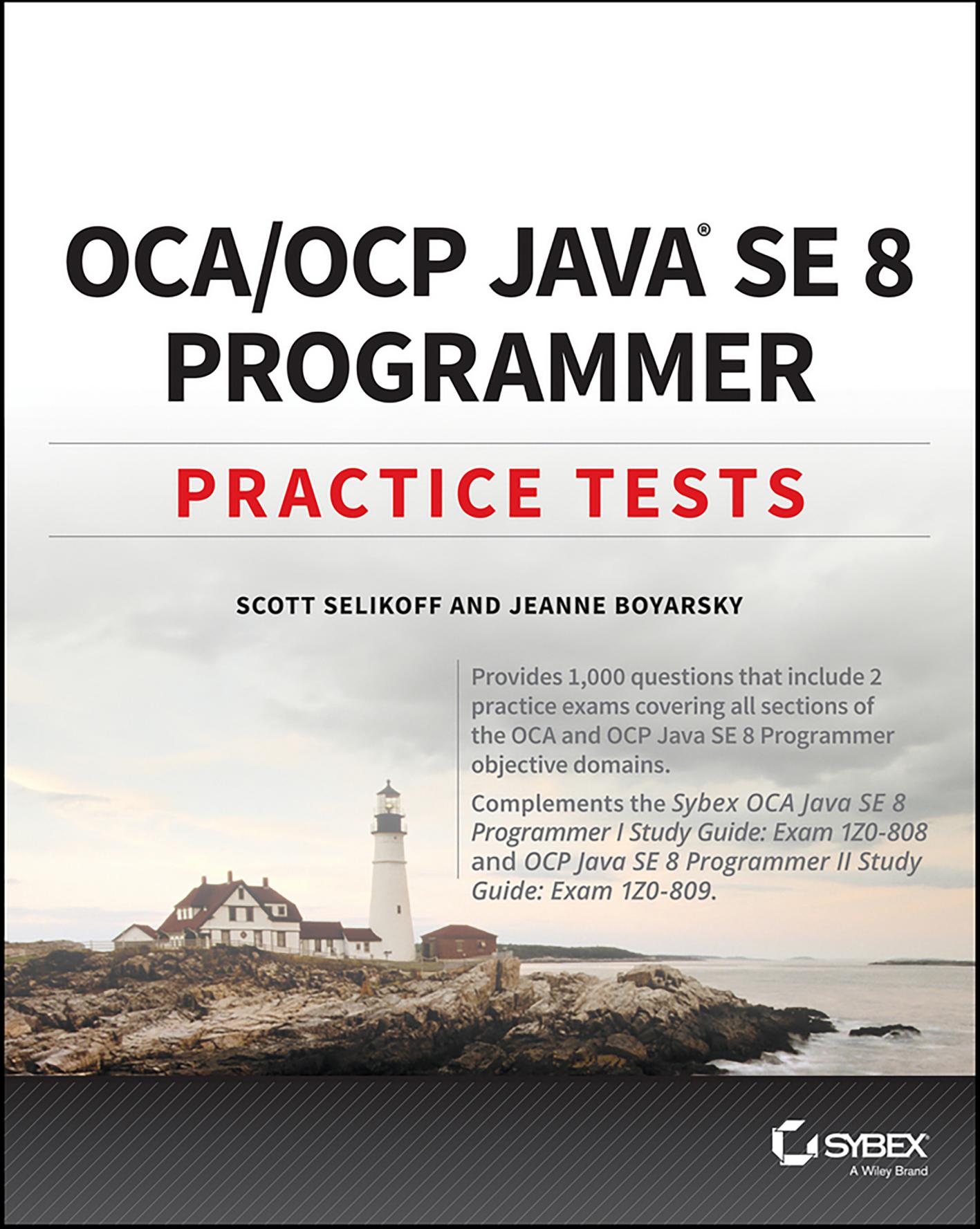


OCA/OCP JAVA® SE 8 PROGRAMMER

PRACTICE TESTS

SCOTT SELIKOFF AND JEANNE BOYARSKY



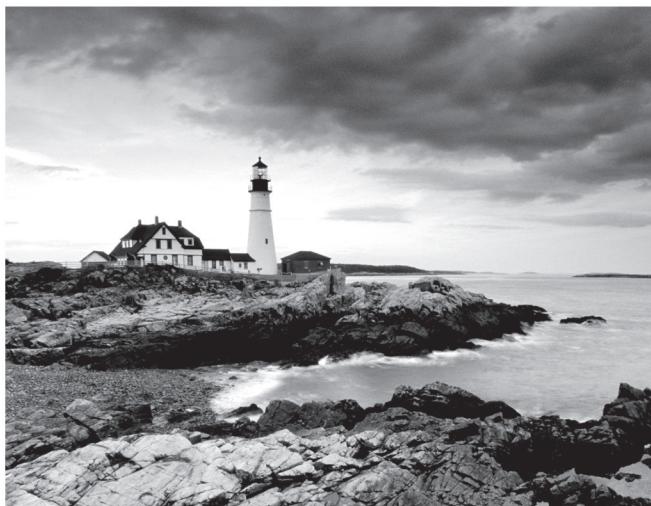
Provides 1,000 questions that include 2 practice exams covering all sections of the OCA and OCP Java SE 8 Programmer objective domains.

Complements the *Sybex OCA Java SE 8 Programmer I Study Guide: Exam 1Z0-808* and *OCP Java SE 8 Programmer II Study Guide: Exam 1Z0-809*.

OCA/OCP

Java® SE 8 Programmer

Practice Tests



Scott Selikoff
Jeanne Boyarsky



Senior Acquisitions Editor: Kenyon Brown
Development Editor: David Clark
Technical Editors: Janeice DelVecchio, Elena Felder
Production Editor: Rebecca Anderson
Copy Editor: Judy Flynn
Editorial Manager: Mary Beth Wakefield
Production Manager: Kathleen Wisor
Executive Editor: Jim Minatel
Book Designer: Judy Fung and Bill Gibson
Proofreader: Nancy Carrasco
Indexer: Ted Laux
Project Coordinator, Cover: Brent Savage
Cover Designer: Wiley
Cover Image: ©Getty Images Inc./Jeremy Woodhouse
Copyright © 2017 by John Wiley & Sons, Inc., Indianapolis, Indiana
Published simultaneously in Canada
ISBN: 978-1-119-36339-2
ISBN: 978-1-119-36336-1 (ebk.)
ISBN: 978-1-119-36340-8 (ebk.)

Manufactured in the United States of America

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommends it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (877) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2017931099

TRADEMARKS: Wiley, the Wiley logo, and the Sybex logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Java is a registered trademark of Oracle America, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

10 9 8 7 6 5 4 3 2 1

To the new little bundle of joy my wife is carrying.

— Scott

Remembering Einstein and CV in same year. Congrats 694!

— Jeanne

Acknowledgments

Scott and Jeanne would like to thank numerous individuals for their contribution to this book. Thank you to David Clark for guiding us through the process and making the book better in so many ways. Thank you to Janeice DelVecchio for being our technical editor as we wrote this book. Janeice pointed out many subtle errors in addition to the big ones. And thank you to Elena Felder for being our technical proofreader and finding the errors that we managed to sneak by Janeice. This book also wouldn't be possible without many people at Wiley, including Kenyon Brown, Rebecca Anderson, Judy Flynn, Nancy Carrasco, and so many others.

Scott could not have reached this point without the help of a small army of people, led by his perpetually understanding wife, Patti, the most wonderful mother their twin daughters, Olivia and Sophia, could ask for. Professors Johannes Gehrke and John H. Hubbard of Cornell University always believed in him and knew he would excel in his career. Scott would like to extend his gratitude to his wonderfully patient co-author Jeanne, on this their third book. "I don't know how she puts up with me, but I'm glad she does and thrilled at the quality of books we produce." A big thanks to Matt Dalen, who has been a great friend, sounding board, and caring father to the wonderfully sweet Olivia Dalen. Joel McNary introduced Scott to CodeRanch.com and encouraged him to post regularly, a step that changed his life. Finally, Scott would like to thank his mother and retired teacher, Barbara Selikoff, for teaching him the value of education and his father, Mark Selikoff, for instilling in him the benefits of working hard.

Jeanne would personally like to thank everyone at CodeRanch.com who asked and responded to OCA and OCP questions and comments about our first two books. Having dialog with the readers made this book even stronger. Jeanne would like to thank the members of FIRST robotics FRC team 694 in addition to FTC teams 310 and 479 for their support. Your questions make it so she can never forget what new learners are thinking. Go StuyPulse! Jeanne would also like to thank Onur Otlu and Thomas Campos for the feedback on some localization questions. Finally, Jeanne would like to thank Scott for being a great co-author and "not getting tired of me."

Last but not least, both Scott and Jeanne would like to give a big thank you to the readers of our OCA 8 and OCP 8 books. Hearing from all of you who enjoyed the book and passed the exam is a great feeling. We'd also like to thank those who pointed out errors and made suggestions for improvements in our OCP book. As of December 2016, the top three were Guillaume Bailly, Thalita Vergilio, and Sébastien Canonica. We also would like to thank Olivier Chalet, Mihaela Hetea, Peter Deak, Anton Shaikin, Ramya R, Tim Moroz, Aurelien Gamet, Javid Azimli, Salim Rahal, and Sander Wamelink.

About the Authors

Scott Selikoff is a professional software consultant, author, and owner of Selikoff Solutions, LLC, which provides software development solutions to businesses in the tri-state New York City area. Skilled in a plethora of software languages and platforms, Scott specializes in full-stack database-driven systems, cloud-based applications, and service-oriented architectures.

A native of Toms River, New Jersey, Scott achieved his Bachelor of Arts from Cornell University in Mathematics and Computer Science in 2002, after three years of study. In 2003, he received his Masters of Engineering in Computer Science, also from Cornell University.

As someone with a deep love of education, Scott has always enjoyed teaching others new concepts. He's given lectures at Cornell University and Rutgers University, as well as conferences including the Server Side Java Symposium. Scott lives in New Jersey with his loving wife, Patti; amazing twin toddler daughters, Olivia and Sophia; and two very playful dogs, Webby and Georgette. You can find out more about Scott at <http://www.linkedin.com/in/selikoff>.

Jeanne Boyarsky has worked as a Java developer for over 14 years at a bank in New York City where she develops, mentors, and conducts training. Besides being a senior moderator at CodeRanch.com in her free time, she works on the forum's code base. Jeanne also mentors the programming division of a FIRST robotics team, where she works with students just getting started with Java.

Jeanne got her Bachelor of Arts degree in 2002 in Computer Science and her Masters in Computer Information Technology in 2005. She enjoyed getting her Masters degree in an online program while working full time. This was before online education was cool! Jeanne is also a Distinguished Toastmaster and a Scrum Master. You can find out more about Jeanne at <https://coderanch.com/wiki/660334>.

Scott and Jeanne are both moderators on the CodeRanch.com forums, and they can be reached there for questions and comments. They also co-author a technical blog called Down Home Country Coding at <http://www.selikoff.net>.

In addition to this book, Scott and Jeanne are also the authors of *OCA Oracle Certified Associate Java SE 8 Programmer I Study Guide* (Sybex, 2015) and *OCP Oracle Certified Professional Java SE 8 Programmer II Study Guide* (Sybex, 2016). More recently, these two books have been combined into the single release *OCA / OCP Java SE 8 Programmer Certification Kit: Exam 1Z0-808 and Exam 1Z0-809* (Sybex 2016).

Contents

<i>Introduction</i>	<i>ix</i>
---------------------	-----------

Part I	OCA	1
Chapter 1	Java Basics	3
Chapter 2	Working with Java Data Types	17
Chapter 3	Using Operators and Decision Constructs	33
Chapter 4	Creating and Using Arrays	49
Chapter 5	Using Loop Constructs	63
Chapter 6	Working with Methods and Encapsulation	79
Chapter 7	Working with Inheritance	97
Chapter 8	Handling Exceptions	115
Chapter 9	Working with Selected Classes from the Java API	135
Chapter 10	OCA Practice Exam	151
Part II	OCP	185
Chapter 11	Java Class Design	187
Chapter 12	Advanced Java Class Design	205
Chapter 13	Generics and Collections	223
Chapter 14	Lambda Built-in Functional Interfaces	239
Chapter 15	Java Stream API	255
Chapter 16	Exceptions and Assertions	271
Chapter 17	Use Java SE 8 Date/Time API	289

Chapter 18	Java I/O Fundamentals	301
Chapter 19	Java File I/O (NIO.2)	319
Chapter 20	Java Concurrency	337
Chapter 21	Building Database Applications with JDBC	357
Chapter 22	Localization	371
Chapter 23	OCP Practice Exam	385
Appendix	Answers to Review Questions	427
<i>Index</i>		565

Introduction

This book is intended for those taking either the 1Z0-808 or 1Z0-809 Oracle Java Programmer exams as well as those who want to test their knowledge of Java 8. If you are new to Java 8, we strongly recommend you start with a study guide to learn all of the facets of the language and come back to this book once you are thinking of taking the exam.

We recommend the best-selling *OCA Oracle Certified Associate Java SE 8 Programmer I Study Guide* and *OCP Oracle Certified Professional Java SE 8 Programmer II Study Guide*, which we happen to be the authors of, to start in your studies. Regardless of which study guide you are using to prepare, you can use this book to hone your skills, since it is based on topics on the actual exams.

Unlike the questions in our study guides, which are designed to be harder than the real exam, the questions in this book mirror the exam format. All the questions in this book tell you how many answers are correct. They will say “Choose two” or “Choose three” if more than one answer is correct.

Throughout this book, we use the same set of assumptions that Oracle uses for its exams. In many cases, these assumptions are actually accounting for unintended omissions or type-setting errors that Oracle is directing you to ignore when solving a question focused on a particular exam objective. The assumptions are listed at <http://education.oracle.com> and listed here for your convenience:

- **Missing package and import statements:** If sample code does not include package or import statements, and the question does not explicitly refer to these missing statements, then assume that all sample code is in the same package, and import statements exist to support them.
- **No file or directory path names for classes:** If a question does not state the file names or directory locations of classes, then assume one of the following, whichever will enable the code to compile and run:
 - All classes are in one file.
 - Each class is contained in a separate file, and all files are in one directory.
- **Unintended line breaks:** Sample code might have unintended line breaks. If you see a line of code that looks like it has wrapped, and this creates a situation where the wrapping is significant (for example, a quoted `String` literal has wrapped), assume that the wrapping is an extension of the same line, and the line does not contain a hard carriage return that would cause a compilation failure.
- **Code fragments:** A code fragment is a small section of source code that is presented without its context. Assume that all necessary supporting code exists, and that the supporting environment fully supports the correct compilation and execution of the code shown and its omitted environment.
- **Descriptive comments:** Take descriptive comments, such as “setter and getters go here,” at face value. Assume that correct code exists, compiles, and runs successfully to create the described effect.

Choosing an Exam

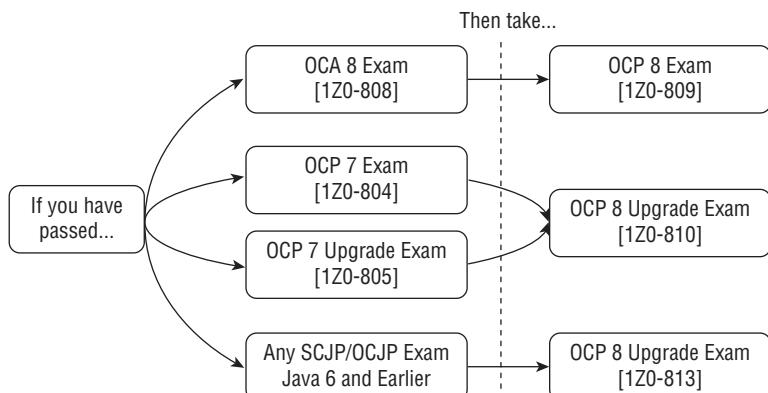
The first step on the road to a Java certification is the Oracle Certified Associate 1Z0-808 (OCA 8) exam. The material includes the basic day-to-day tools every Java developer should be familiar with. Once you have completed that exam, you can move on to the more difficult Oracle Certified Professional 1Z0-809 (OCP 8) exam. This includes every topic on the first exam, along with many additional libraries and APIs that you may not even be familiar with in your career, such as the Concurrency and NIO.2 APIs.

If you already hold a Java certification, you are eligible for an upgrade exam. Table I.1 and Figure I.1 show the various pathways to the OCP 8 certification title. This book will help you prepare for all four of the exams listed in the table.

TABLE I.1 Exams this book covers

Exam Code	Name	Who should take
1Z0-808	Java Programmer I	Everyone can take this exam.
1Z0-809	Java Programmer II	Those who pass the OCA 8 exam
1Z0-810	Upgrade Java SE 7 to Java SE 8 OCP Programmer	Holders of an OCP 7 certification
1Z0-813	Upgrade to Java SE 8 OCP (Java SE 6 and all prior versions)	Holders of any of the following certifications: <ul style="list-style-type: none"> ▪ SCJP/OCJP 6 ▪ SCJP/OCJP 5 ▪ SCJP 1.4 Any older SCJP certs

FIGURE I.1 Exam prerequisites



For the OCA exam, we recommend a study plan that uses our *OCA Oracle Certified Associated Java SE 8 Programmer I Study Guide* along with Chapters 1 through 10 of this book. For the OCP exam, we recommend a study plan that uses our *OCP Oracle Certified Professional Java SE 8 Programmer II Study Guide* along with Chapters 11 through 23 of this book. That's right, this book is actually two books in one! It assists you for both the OCA and OCP exams.

Taking an Upgrade Exam

There are some subtle differences in how you should prepare if you are taking one of the upgrade exams. The good news is that if you currently hold a Java 7 OCP certification and are taking the 1Z0-810 upgrade exam, this book is all you need. While the exam contains a stronger focus on changes between Java 7 and Java 8, there aren't any new topics not covered with the existing OCP 7 material. This means you have the same study plan as those taking the OCP for the first time.

On the other hand, if you are taking the 1Z0-813 upgrade exam and hold a Sun Certified Java Programmer (SCJP) or Oracle Certified Java Programmer (OCJP) title, then you will need additional material. This upgrade inexplicably contains material that was in the Java 7 OCP exam but removed from the Java 8 version, such as `ReentrantLock` and `WatchService`. For that, we have no explanation, although this does make the 1Z0-813 arguably the most difficult of the three OCP exams because it requires knowing everything from multiple versions of Java.

Have no fear! When we wrote our *OCP Oracle Certified Professional Java SE 8 Programmer II Study Guide*, we added a special 50-page appendix, Appendix C, "Upgrading from Java 6 or Earlier," which covers the topics solely on the 1Z0-813 exam in detail. If you are taking the 1Z0-813 exam, then after completing your studies with this book, you should consult our appendix in our OCP book or a Java 7 OCP study guide so that you are prepared for all of the objectives.

Considering Other Study Paths

If you already hold a Java certification, you are not required to take an upgrade exam. Some developers find it more straightforward to start with the easier 1Z0-808 OCA exam and then take the more challenging 1Z0-809 OCP exam. If you have no experience with Java 8 or just want more practice with it, taking the OCA exam followed by the OCP exam could be a better, albeit more expensive, learning experience than taking a single upgrade exam.

Also remember, if you do hold an old Java title and want to take an upgrade exam, you will need to prove it before you will be granted the OCP 8 title, even if you pass the upgrade exam. This requires registering your older certification with Oracle's CertView system. We've heard feedback from some readers who had to search through old binders and emails from over a decade ago to submit the proper documentation that allows them to be granted the OCP 8 title.

Finally, if you are new to Java certification, it is traditional to take the exams in order, with the OCA exam followed by the OCP exam. However, this is a not strict requirement. It is possible, although quite unorthodox, to take the harder OCP exam first and then circle back to take the OCA exam. In this case, you'll be granted both OCA and OCP titles upon completion of the OCA exam. While we don't recommend this strategy, Oracle does allow it.

Who Should Buy This Book

If you are looking to take the OCA 8 exam (1Z0-808), then Chapters 1 through 10 of this book are for you. And once you've passed with flying colors, you'll have the OCP material handy in case you decide to take the exam.

If you are looking to take the OCP 8 exam (1Z0-809) or OCP 8 upgrade exam (1Z0-810), then Chapters 1 through 23 of this book are for you. While Chapters 1 through 10 are primarily focused on the OCA exam, the OCP exam is cumulative. We recommend you start reviewing the OCA Chapters (1–10) to make sure you have a solid foundation and then move on to the OCP Chapters (11–23) when you are ready.

As mentioned earlier, if you are looking to take the OCP 8 upgrade exam (1Z0-813), this book will still help for the vast majority of topics. You will need to supplement this book with a Java 7 study guide or use Appendix C in our OCP book, which focuses on the precise topics on the 1Z0-813 exam that you need to know.

Regardless of which exam you plan to take, make sure to always keep your study guide handy. This book is about honing your knowledge of Java 8, while your study guide is about building it.

How This Book Is Organized

For this book, we decided to write two books in one, divided into Parts I and II. Part I includes Chapters 1 through 10, with nine objective-based chapters for each of the OCA exam objective sets followed by a simulated OCA practice exam. Part II encompasses Chapters 11 through 23, with 12 objective-based chapters for each of the OCP exam objective sets followed by a simulated OCP practice exam.

There are some subtle differences between the objective-based chapters and practice exam chapters that you should be aware of while reading this book.

Using the Objective-Based Chapters

We designed the structure and style of each question in the objective-based chapters to reflect a more positive learning experience, allowing you to spend less time on each question but covering a broader level of material. For example, you may see two questions that

look similar within a chapter but contain a subtle difference that has drastic implications on whether or not the code compiles, or what output it produces.

Each question in the objective-based chapters has exactly four options with only one correct answer. Just like the review questions in our study guide, these questions are designed so that you can answer them many times. While these questions may be easier than exam questions, they will reinforce concepts if you keep taking them on a topic you don't feel strongly on.

In our study guides, we often group related topics into chapters. For example, if/then statements and loops, which are in separate Oracle objective sets, were presented in a single chapter on Operators and Statements in our OCA book. In this book, we decided to design our chapters solely around Oracle's objectives so you can strengthen your skills. While you don't need to read an entire study guide before using an objective-based chapter in this book, you do need to study the relevant objectives. Tables I.2 and I.3 show what chapters you need to have read in our study guides as a minimum before practicing with the questions in this book.

TABLE I.2 Reference in OCA study guide

Chapter in This Book	Objectives	OCA Study Guide Chapters
1	Java Basics	1
2	Working with Java Data Types	1, 3
3	Using Operators and Decision Constructs	2, 3
4	Creating and Using Arrays	3
5	Using Loop Constructs	2
6	Working with Methods and Encapsulation	4
7	Working with Inheritance	5
8	Handling Exceptions	6
9	Working with Selected Classes from the Java API	1, 3, 4

TABLE I.3 Reference in OCP study guide

Chapter in This Book	Objectives	OCP Study Guide Chapters
11	Java Class Design	1, 2
12	Advanced Java Class Design	1, 2, 3
13	Generics and Collections	3, 4
14	Lambda Built-in Functional Interfaces	2, 4
15	Java Stream API	3, 4
16	Exceptions and Assertions	6
17	Use Java SE 8 Date/Time API	5
18	Java I/O Fundamentals	8
19	Java File I/O (NIO.2)	9
20	Java Concurrency	7
21	Building Database Applications with JDBC	10
22	Localization	5

Using the Practice Exam Chapters

This book contains two full-length practice exam chapters, with Chapter 10 being an 80-question OCA practice exam and Chapter 23 being an 85-question OCP practice exam. The questions in these two chapters are quite different from the objective-based chapters in a number of important ways. These practice exam questions tend to be harder because they are designed to test your cumulative knowledge rather than reinforcing your existing skill set.

While all of the objective-based chapters had four options with only one correct answer, these questions have up to six options, with up to three correct answer choices. Based on feedback from our first two books, we do indicate exactly how many answers are correct in the practice exam chapters, as is done on the real exam. Some readers thought the lack of knowing the correct number of answers made the questions too challenging for studying.

Both practice exam chapters are designed to be taken within 150 minutes and have a passing score of 65 percent. Remember not to take the practice exam until you feel ready. There are only so many practice exams available, so you don't want to waste a fresh attempt.

While an objective-based chapter can be completed over the course of a few days, the practice exam chapters were designed to be completed in one sitting. You should try simulating the exam experience as much as possible. This means setting aside two and a half hours, grabbing a whiteboard or scrap paper, and answering every question even if you aren't sure of the answer. Remember, there is no penalty for guessing, and the more incorrect answers you can eliminate the better.

Reviewing Exam Changes

Oracle does change the number of questions, passing score, and time limit from time to time. In fact, the exam writers changed the number of OCA 8 questions from 77 to 80 while this book was being written! Scott and Jeanne maintain a blog that tracks updates to the real exams, as quickly as Oracle updates them:

<https://www.selikoff.net/jpt>

We recommend you read this page before you take the real exam, in case any of the information since the time this book was published has changed. Although less common, Oracle does add, remove, or reword objectives. When this happens, we offer free supplemental material on our website as blog entries.

Ready to Take the Exam

If you can score above 80 percent consistently on all of the chapters related to the exam you want to take, including the simulated practice exam, then you are probably ready to take the real exam. Just remember there's a big difference between taking a practice test at home and spending hundreds of dollars to take a real exam at a test center.

We could write an entire chapter on test taking skills and study tips. Oh wait, we did! Both our OCA 8 and OCP 8 books each contain an appendix chock-full of helpful tips and suggestions that are designed to help you manage your time. They also include notes on how to eliminate obviously wrong answers so that when you have to guess, you're choosing between two choices and not five.

Finally, although a lot of people are inclined to cram as much material as they can in the hours leading up to the exam, most studies have shown that this is a poor test-taking strategy. The best thing we can recommend that you do before the exam is to get a good night's rest!

Need More Help Preparing?

Both of the authors are moderators at CodeRanch.com, a very large and active programming forum that is very friendly toward Java beginners. It has separate forums for each of the exams:

OCA Forum: <https://coderanch.com/f/117>

OCP Forum: <https://coderanch.com/f/24>

If you don't understand a question, even after reading the explanation, feel free to ask about it in one of those forums. You'll get an answer from a knowledgeable Java programmer. It might even be one of us.

Good luck on the exam and happy studies!

Bonus Contents

This book has a web page that provides all the questions in this book using Wiley's interactive online test engine.



You can link to this from www.wiley.com/go/sybextestprep.

OCA

PART

1



Chapter

1



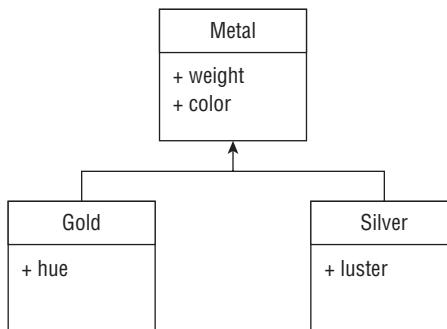
Java Basics

**THE OCA EXAM TOPICS COVERED
IN THIS PRACTICE TEST INCLUDE
THE FOLLOWING:**

✓ **Java Basics**

- Define the scope of variables
- Define the structure of a Java class
- Create executable Java applications with a main method; run a Java program from the command line; produce console output
- Import other Java packages to make them accessible in your code
- Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.

- Which of the following method signatures is a valid declaration of an entry point in a Java application?
 - public void main(String[] args)
 - public static void main()
 - private static void start(String[] mydata)
 - public static final void main(String[] mydata)
- The following class diagram demonstrates the relationship between Gold and Silver, which extend the Metal class. Assume the attributes are all declared public. Which statement about the diagram is **not true**?



- The diagram demonstrates platform independence in Java.
- The diagram demonstrates object-oriented design in Java.
- The Gold and Silver classes inherit weight and color attributes from the Metal class.
- Gold does not inherit the luster attribute.
- What is the proper filename extension for a Java bytecode compiled file?
 - .java
 - .bytecode
 - .class
 - .dll
- Given that a Date class exists in both the java.util and java.sql packages, what is the result of compiling the following class?

```

1: import java.util.*;
2: import java.sql.*;
3: public class BirthdayManager {
4:     private Date rob = new Date();
5:     private java.util.Date sharon = new java.util.Date();
6: }
  
```

- A. The code does not compile because of lines 1 and 2.
 - B. The code does not compile because of line 4.**
 - C. The code does not compile because of line 5.
 - D. The code compiles without issue.
5. Which of the following is not a facet of traditional object-oriented programming languages?
- A. Objects are grouped as procedures, separate from the data they act on.**
 - B. An object can take many forms via casting.
 - C. An object can hold data, referred to as attributes.
 - D. An object can perform actions, via methods.
6. Which variables have a scope limited to a method?
- A. Interface variables
 - B. Class variables
 - C. Instance variables
 - D. Local variables**
7. Which package is imported into every Java class by default?
- A. `java.util`
 - B. `java.lang`**
 - C. `system.lang`
 - D. `java.system`
8. Which of the following is not a valid code comment in Java?
- A. `// Add 5 to the result`
 - B. `/**/ TODO: Fix bug 12312 */`
 - C. `# Add configuration value`**
 - D. `/* Read file from system */`
9. Which statement about a valid `.java` file is true?
- A. It can only contain one class declaration.
 - B. It can contain one public class declaration and one public interface definition.
 - C. It must define at least one public class.
 - D. It may define at most one public class.**
10. Given the following application, fill in the missing values in the table starting from the top and going downward.

```
package competition;
public class Robot {
    static String weight = "A lot";
    /* default */ double ageMonths = 5, ageDays = 2;
```

```
private static boolean success = true;  
public void main(String[] args) {  
    final String retries = "1";  
    // P1  
}  
}
```

Variable Type	Number of Variables Accessible at P1
---------------	--------------------------------------

Class	_____
-------	-------

Instance	_____
----------	-------

Local	_____
-------	-------

- A. 2, 0, 1
- B. 2, 2, 1**
- C. 1, 0, 1
- D. 0, 2, 1

11. Which statement about `import` statements is true?

- A. The class will not compile if it contains unused `import` statements.
- B. Unused `import` statements can be removed from the class without causing a class to become unable to be compiled.**
- C. The class will not compile if a duplicate `import` statement is present.
- D. If a class contains an `import` statement for a class used in the program that cannot be found, it can still compile.

12. What is the result of compiling and executing the following class?

```
1: public class ParkRanger {  
2:     int birds = 10;  
3:     public static void main(String[] data) {  
4:         int trees = 5;  
5:         System.out.print(trees+birds);  
6:     }  
7: }
```

- A. It does not compile.**
- B. It compiles but throws an exception at runtime.
- C. It compiles and outputs 5.
- D. It compiles and outputs 15.

- 13.** Which statements about Java are true?
- I.** The `java` command can execute `.java` and `.class` files.
 - II.** Java is not object oriented.
 - III.** The `javac` command compiles directly into native machine code.
 - A.** I only
 - B.** III only
 - C.** II and III
 - D.** None are true.
- 14.** Which of the following lines of code is not allowed as the first line of a Java class file?
- A.** `import widget.*;`
 - B.** `// Widget Manager`
 - C.** `package sprockets;`
 - D.** `int facilityNumber;`
- 15.** Which one of the following statements is true about using packages to organize your code in Java?
- A.** Every class is required to include a package declaration.
 - B.** To create a new package, you need to add a `package.init` file to the directory.
 - C.** Packages allow you to limit access to classes, methods, or data from classes outside the package.
 - D.** It is not possible to restrict access to objects and methods within a package.
- 16.** Given that the current directory is `/user/home`, with an application Java file in `/user/home/Manager.java` that uses the default package, which are the correct commands to compile and run the application in Java?
- A.** `javac Manager`
`java Manager`
 - B.** `javac Manager.java`
`java Manager`
 - C.** `javac Manager`
`java Manager.class`
 - D.** `javac Manager.java`
`java Manager.class`
- 17.** Structuring a Java class such that only methods within the class can access its instance variables is referred to as _____.
- A.** platform independence
 - B.** object orientation
 - C.** inheritance
 - D.** encapsulation

18. What is the output of the following code snippet?

```
String tree = "pine";
int count = 0;
if (tree.equals("pine")) {
    int height = 55;
    count = count + 1;
}
System.out.print(height + count);
```

- A. 1
- B. 55
- C. 56
- D. It does not compile.**

19. Which of the following is true of a Java bytecode file?

- A. It can be run on any computer with a compatible JVM.**
- B. It can only be executed on the same type of computer that it was created on.
- C. It can be easily read and modified in a standard text editor.
- D. It requires the corresponding .java that created it to execute.

20. What is the correct character for terminating a statement in Java?

- A. A colon (:
- B. An end-of-line character
- C. A tab character
- D. A semicolon (;**

21. What is the result of compiling and executing the following class?

```
1: public class Tolls {
2:     private static int yesterday = 1;
3:     int tomorrow = 10;
4:     public static void main(String[] args) {
5:         Tolls tolls = new Tolls();
6:         int today=20, tomorrow = 40;
7:         System.out.print(today + tolls.tomorrow + tolls.yesterday);
8:     }
9: }
```

- A. The code does not compile due to line 6.
- B. The code does not compile due to line 7.
- C. 31**
- D. 61

22. Given the following class definition, which is the only line that does not contain a compilation error?

```
1: public ThisClassDoesNotCompile {  
2:     double int count;  
3:     void errors() {}  
4:     static void private limit; }
```

- A. Line 1
- B. Line 2
- C. Line 3**
- D. Line 4

23. Which of the following features allows a Java class to be run on a wide variety of computers and devices?

- A. Encapsulation
- B. Object oriented
- C. Inheritance
- D. Platform independence**

24. Which of the following is not a property of a JVM?

- A. It prevents Java bytecode from being easily decoded/decompiled.**
- B. It supports platform independence.
- C. It manages memory for the application.
- D. It translates Java instructions to machine instructions.

25. Which of the following variables are always in scope for the entire program?

- A. Package variables
- B. Class variables**
- C. Instance variables
- D. Local variables

26. Given the following wildcard import statements, which class would be included in the import?

```
import television.actor.*;  
import movie.director.*;  
  
A. television.actor.recurring.Marie  
B. movie.directors.John  
C. television.actor.Package  
D. movie.NewRelease
```

27. Which is the correct order of statements for a Java class file?
- A. import statements, package statement, class declaration
 - B. package statement, class declaration, import statement
 - C. class declaration, import statements, package declaration
 - D. package statement, import statements, class declaration**
28. Given the following class definition, what is the maximum number of `import` statements that can be discarded and still have the code compile? For this question, assume that the `Blackhole` class is defined only in the `stars` package.

```
package planetarium;  
import java.lang.*;  
import stars.*;  
import java.lang.Object;  
import stars.Blackhole;  
  
public class Observer {  
    public void find(Blackhole blackhole) {}  
}
```

- A. Zero**
 - B. One
 - C. Two
 - D. Three**
29. Given the following class definition, which command will cause the application to output the message `White-tailed`?

```
package forest;  
public class Deer {  
    public static void main(String... deerParams) {  
        System.out.print(theInput[2]);  
    }  
}
```

- A. java forest.Deer deer 5 "White-tailed deer"**
 - B. java forest.Deer "White-tailed deer" deer 3
 - C. java forest.Deer Red deer White-tailed deer**
 - D. java forest.Deer My "deer White-tailed"
30. Which of the following is a true statement?
- A. The `java` command compiles a `.java` file into a `.class` file.**
 - B. The `javac` command compiles a `.java` file into a `.class` file.**
 - C. The `java` command compiles a `.class` file into a `.java` file.
 - D. The `javac` command compiles a `.class` file into a `.java` file.

31. Which of the following statements about Java is true?

- A. Java is a procedural programming language.
- B. Java allows method overloading.**
- C. Java allows operator overloading.
- D. Java allows direct access to objects in memory.

32. Given the following code, what values inserted in order into the blank lines, allow the code to compile?

```
_____ agent;
public _____ Banker {
    private static _____ getMaxWithdrawal() {
        return 10;
    }
}
```

- A. import, class, null
- B. import, interface, void
- C. package, int, int
- D. package, class, long**

33. What is the output of the following application?

```
public class Airplane {
    static int start = 2;
    final int end;
    public Airplane(int x) {
        x = 4;
        end = x;
    }
    public void fly(int distance) {
        System.out.print(end-start+" ");
        System.out.print(distance);
    }
    public static void main(String... start) {
        new Airplane(10).fly(5);
    }
}
```

- A. 2 5**
- B. 8 5
- C. 6 5
- D. The code does not compile.

- 34.** What is one of the **most important** reasons that Java supports extending classes via inheritance?
- A.** Inheritance requires that a class that extends another class be in the same package.
 - B.** The program must spend extra time/resources at runtime jumping through multiple layers of inheritance to determine precise methods and variables.
 - C.** Method signature changes in parent classes may break subclasses that use overloaded methods.
 - D.** Developers minimize duplicate code in new classes by sharing code in a **common parent class**.
- 35.** Which of the following is a valid code comment in Java?
- A.** `///////// Walk my dog`
 - B.** `#! Go team!`
 - C.** `/ Process fails at runtime /`
 - D.** None of the above
- 36.** Which of the following method signatures is not a valid declaration of an entry point in a Java application?
- A.** `public static void main(String... arguments)`
 - B.** `public static void main(String arguments)`
 - C.** `public static final void main(String[] arguments)`
 - D.** `public static void main(String[] arguments)`
- 37.** Given the file `Magnet.java` below, which of the marked lines can you independently insert the line `public String color;` into and still have the code compile?
- ```
// line a1
public class Magnet {
 // line a2
 public void attach() {
 // line a3
 }
 // line a4
}
```
- A.** a1 and a3
  - B.** a2 and a4
  - C.** a2, a3, and a4
  - D.** a1, a2, a3, and a4
- 38.** What is required to define a valid Java class file?
- A.** A **class declaration**
  - B.** A **package statement**
  - C.** At least one **import statement**
  - D.** The **public modifier**

39. What is the proper filename extension for a Java source file?

- A. .jav
- B. .class
- C. .source
- D. .java**

40. Given that a Math class exists in both the `java.lang` and `pocket.complex` packages, what is the result of compiling the following class?

```
1: package pocket;
2: import pocket.complex.*;
3: import java.util.*;
4: public class Calculator {
5: public static void main(String[] args) {
6: System.out.print(Math.floor(5));
7: }
8: }
```

- A. The code does not compile because of line 2.**
  - B. The code does not compile because of line 3.**
  - C. The code does not compile because of line 6.**
  - D. The code compiles without issue.**
41. Given a class that uses the following `import` statements, which class would not be automatically accessible within the class without using its full package name?

```
import dog.*;
import dog.puppy.*;
```

- A. dog.puppy.female.KC**
  - B. dog.puppy.Georgette**
  - C. dog.Webby**
  - D. java.lang.Object**
42. \_\_\_\_\_ is the technique of structuring programming data as a unit consisting of attributes, with actions defined on the unit.
- A. Encapsulation**
  - B. Object orientation**
  - C. Platform independence**
  - D. Polymorphism**

43. Given the following class definition, what is the maximum number of `import` statements that can be discarded and still have the code compile? For this question, assume that the `Broccoli` class is in the `food.vegetables` package, and the `Apple` class is in the `food.fruit` package.

```
package food;
import food.vegetables.*;
import food.fruit.*;
import java.util.Date;

public class Grocery {
 Apple a; Broccoli b; Date c;
}
```

- A.** 0
- B.** 1
- C.** 2
- D.** 3

44. Given the following application, what is the expected output?

```
public class Keyboard {
 private boolean numLock = true;
 static boolean capLock = false;
 public static void main(String... shortcuts) {
 System.out.print(numLock+" "+capLock);
 }
}

- A. true false
- B. false false
- C. It does not compile.
- D. It compiles but throws an exception at runtime.

```

45. What is the result of compiling and executing the following class?

```
public class RollerSkates {
 static int wheels = 1;
 int tracks = 5;
 public static void main(String[] arguments) {
 RollerSkates s = new RollerSkates();
 int feet=4, tracks = 15;
 System.out.print(feet + tracks + s.wheels);
 }
}
```

- A. The code does not compile.
- B. 5
- C. 10
- D. 20**

46. What is the result of compiling and executing the following class?

```
package sports;
public class Bicycle {
 String color = "red";
 private void printColor(String color) {
 color = "purple";
 System.out.print(color);
 }
 public static void main(String[] rider) {
 new Bicycle().printColor("blue");
 }
}
```

- A. red
- B. purple**
- C. blue
- D. It does not compile.

47. Which statements about calling the compilation command `javac` and the execution command `java` are true?

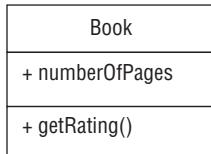
- I. java may use a period `.` to separate packages.
  - II. javac takes a `.java` file and returns a `.class` file.
  - III. java may use a slash `(/)` to separate packages.
- A. I only
  - B. II only
  - C. I and II**
  - D. I, II, and III

48. What is the result of compiling and executing the following application?

```
package forecast;
public class Weather {
 private static boolean heatWave = true;
 public static void main() {
 boolean heatWave = false;
 System.out.print(heatWave);
 }
}
```

- A. true
- B. false
- C. It does not compile.
- D. It compiles but throws an error at runtime.**

49. Given the following class diagram, which Java implementation most closely matches this structure?



- A. 

```
public class Book {
 public int numOfPages;
```
  - B. 

```
public class Book {
 public String getRating() {return null;} }
```
  - C. 

```
public class Book {
 public int numberOfPages;
 public String getRating() {return null;} }
```**
  - D. 

```
public class Book {
 void numberOfPages; }
```
50. Which statement about the JVM is true?
- A. The JVM schedules garbage collection on a predictable schedule.
  - B. The JVM ensures that the application will always terminate.
  - C. The JVM requires a properly defined entry point method to execute the application.**
  - D. A Java compiled code can be run on any computer.

# Chapter 2



# Working with Java Data Types

---

**THE OCA EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Working with Java Data Types**

- Declare and initialize variables (including casting of primitive data types)
- Differentiate between object reference variables and primitive variables
- Know how to read or write to object fields
- Explain an Object's Lifecycle (creation, "dereference by reassignment" and garbage collection)
- Develop code that uses wrapper classes such as Boolean, Double, and Integer

1. Which of the following declarations does not compile?

- A. double num1, int num2 = 0;
- B. int num1, num2;
- C. int num1, num2 = 0;
- D. int num1 = 0, num2 = 0;

2. What is the output of the following?

```
public static void main(String... args) {
 String chair, table = "metal";
 chair = chair + table;
 System.out.println(chair);
}
```

- A. metal
- B. metalmetal
- C. nullmetal
- D. The code does not compile.

3. Which is correct about an instance variable of type String?

- A. It defaults to an empty string.
- B. It defaults to null.
- C. It does not have a default value.
- D. It will not compile without initializing on the declaration line.

4. Which of the following is not a valid variable name?

- A. \_blue
- B. 2blue
- C. blue\$
- D. Blue

5. Which of these class names best follows standard Java naming conventions?

- A. fooBar
- B. FooBar
- C. FOO\_BAR
- D. F\_o\_o\_B\_a\_r

6. How many of the following methods compile?

```
public String convert(int value) {
 return value.toString();
}
public String convert(Integer value) {
 return value.toString();
}
```

```
 }
 public String convert(Object value) {
 return value.toString();
 }
```

- A.** None
  - B.** One
  - C.** Two
  - D.** Three
- 7.** Which of the following does not compile?
- A.** int num = 999;
  - B.** int num = 9\_9\_9;
  - C.** int num = \_9\_99;
  - D.** None of the above; they all compile.
- 8.** Which of the following is a wrapper class?
- A.** int
  - B.** Int
  - C.** Integer
  - D.** Object
- 9.** What is the result of running this code?
- ```
public class Values {  
    integer a = Integer.valueOf("1");  
    public static void main(String[] nums) {  
        integer a = Integer.valueOf("2");  
        integer b = Integer.valueOf("3");  
        System.out.println(a + b);  
    }  
}
```
- A.** 4
 - B.** 5
 - C.** The code does not compile.
 - D.** The code compiles but throws an exception at runtime.
- 10.** Which best describes what the new keyword does?
- A.** Creates a copy of an existing object and treats it as a new one
 - B.** Creates a new primitive
 - C.** Instantiates a new object
 - D.** Switches an object reference to a new one

11. Which is the first line to trigger a compiler error?

```
double d1 = 5f;      // p1
double d2 = 5.0;    // p2
float f1 = 5f;      // p3
float f2 = 5.0;    // p4
```

- A. p1
- B. p2
- C. p3
- D. p4**

12. Which of the following lists of primitive types are presented in order from smallest to largest data type?

- A. byte, char, float, double**
- B. byte, char, double, float
- C. char, byte, float, double
- D. char, double, float, bigint

13. Which of the following is not a valid order for elements in a class?

- A. Constructor, instance variables, method names
- B. Instance variables, constructor, method names
- C. Method names, instance variables, constructor
- D. None of the above: all orders are valid.**

14. Which of the following lines contains a compiler error?

```
String title = "Weather";           // line x1
int hot, double cold;              // line x2
System.out.println(hot + " " + title); // line x3
```

- A. x1
- B. x2**
- C. x3
- D. None of the above

15. How many instance initializers are in this code?

```
1: public class Bowling {
2:     { System.out.println(); }
3:     public Bowling () {
4:         System.out.println();
5:     }
6:     static { System.out.println(); }
```

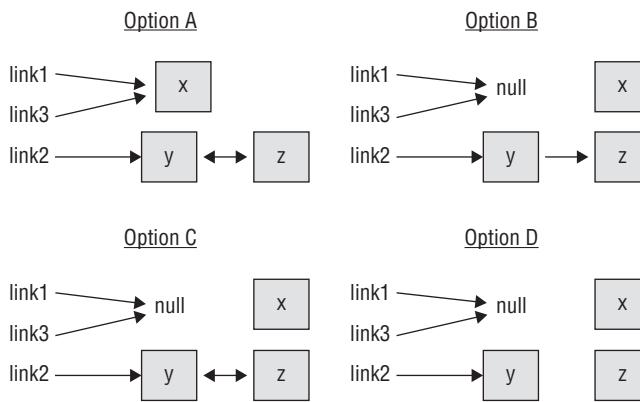
```
7:     { System.out.println(); }  
8: }
```

- A.** None
 - B.** One
 - C.** Two
 - D.** Three
- 16.** Of the types `double`, `int`, and `short`, how many could fill in the blank to have this code output 0?
- ```
public static void main(String[] args) {
 _____ defaultValue;
 System.out.println(defaultValue);
}
```
- A.** None
  - B.** One
  - C.** Two
  - D.** Three
- 17.** What is true of the `finalize()` method?
- A.** It may be called zero or one times.
  - B.** It may be called zero or more times.
  - C.** It will be called exactly once.
  - D.** It may be called one or more times.
- 18.** Which of the following is not a wrapper class?
- A.** `Double`
  - B.** `Integer`
  - C.** `Long`
  - D.** `String`
- 19.** Suppose you have the following code. Which of the images best represents the state of the references right before the end of the `main` method, assuming garbage collection hasn't run?

```
1: public class Link {
2: private String name;
3: private Link next;
4: public Link(String name, Link next) {
5: this.name = name;
6: this.next = next;
7: }
```

```

8: public void setNext(Link next) {
9: this.next = next;
10: }
11: public Link getNext() {
12: return next;
13: }
14: public static void main(String... args) {
15: Link link1 = new Link("x", null);
16: Link link2 = new Link("y", link1);
17: Link link3 = new Link("z", link2);
18: link2.setNext(link3);
19: link3.setNext(link2);
20: link1 = null;
21: link3 = null;
22: }
23: }
```



- A. Option A  
 B. Option B  
 C. Option C  
 D. Option D

20. Which type can fill in the blank?

\_\_\_\_\_ pi = 3.14;

- A. byte  
 B. float  
 C. double  
 D. short

21. What is the first line in the following code to not compile?

```
public static void main(String[] args) {
 int Integer = 0; // k1
 Integer int = 0; // k2
 Integer ++; // k3
 int++; // k4
}
```

- A. k1
- B. k2**
- C. k3
- D. k4

22. Suppose foo is a reference to an instance of a class. Which of the following is not true about foo.bar?

- A. bar is an instance variable.
- B. bar is a local variable.**
- C. It can be used to read from bar.
- D. It can be used to write to bar.

23. Which of the following is not a valid class declaration?

- A. class building {}
- B. class Cost\$ {}
- C. class 5MainSt {}**
- D. class \_Outside {}

24. Which of the following can fill in the blanks to make this code compile?

\_\_\_\_\_ d = new \_\_\_\_\_(1\_000\_000\_.00);

- A. double, double
- B. double, Double
- C. Double, double
- D. None of the above**

25. Which is correct about a local variable of type String?

- A. It defaults to an empty string.
- B. It defaults to null.
- C. It does not have a default value.**
- D. It will not compile without initializing on the declaration line.

26. Of the types double, int, long, and short, how many could fill in the blank to have this code output 0?

```
static _____ defaultValue;

public static void main(String[] args) {
 System.out.println(defaultValue);
}
double in ra 0.0
```

- A. One
- B. Two
- C. Three**
- D. Four

27. Which of the following is true about primitives?

- A. You can call methods on a primitive.
- B. You can convert a primitive to a wrapper class object simply by assigning it.**
- C. You can convert a wrapper class object to a primitive by calling valueOf().
- D. You can store a primitive directly into an ArrayList.

28. What is the output of the following?

```
Integer integer = new Integer(4);
System.out.print(integer.byteValue());

System.out.print("-");
```

```
int i = new Integer(4);
System.out.print(i.byteValue());
```

- A. 4-0
- B. 4-4
- C. The code does not compile.**
- D. The code compiles but throws an exception at runtime.

29. Given the following code, fill in the blank to have the code print bounce.

```
public class TennisBall {
 public TennisBall() {
 System.out.println("bounce");
 }
 public static void main(String[] slam) {

 }
}
```

- A. TennisBall;
  - B. TennisBall();
  - C. new TennisBall;
  - D. new TennisBall();**
30. Which of the following correctly assigns animal to both variables?
- I. String cat = "animal", dog = "animal";
  - II. String cat = "animal"; dog = "animal";
  - III. String cat, dog = "animal";
  - IV. String cat, String dog = "animal";
- A. I**
  - B. I, II
  - C. I, III
  - D. I, II, III, IV
31. Which two primitives have wrapper classes that are not merely the name of the primitive with an uppercase letter?
- A. byte and char
  - B. byte and int
  - C. char and int**
  - D. None of the above
32. Which of the following is true about String instance variables?
- A. They can be set to null.**
  - B. They can never be set from outside the class they are defined in.
  - C. They can only be set in the constructor.
  - D. They can only be set once per run of the program.
33. Which statement is true about primitives?
- A. Primitive types begin with a lowercase letter.**
  - B. Primitive types can be set to null.
  - C. String is a primitive.
  - D. You can create your own primitive types.
34. How do you force garbage collection to occur at a certain point?
- A. Call System.forceGc()**
  - B. Call System.gc()**
  - C. Call System.requireGc()
  - D. None of the above

35. How many of the `String` objects are eligible for garbage collection right before the end of the `main` method?

```
public static void main(String[] fruits) {
 String fruit1 = new String("apple");
 String fruit2 = new String("orange");
 String fruit3 = new String("pear");

 fruit3 = fruit1;
 fruit2 = fruit3;
 fruit1 = fruit2;
}
```

- A.** None
- B.** One
- C.** Two
- D.** Three

36. Which of the following can fill in the blanks to make this code compile?

```
_____ d = new _____(1_000_000.00);
```

- A.** `double, double`
- B.** `double, Double`
- C.** `Double, double`
- D.** None of the above

37. What does the following output?

```
1: public class InitOrder {
2: public String first = "instance";
3: public InitOrder() {
4: first = "constructor";
5: }
6: { first = "block"; }
7: public void print() {
8: System.out.println(first);
9: }
10: public static void main(String... args) {
11: new InitOrder().print();
12: }
13: }
```

- A. block
- B. constructor**
- C. instance
- D. The code does not compile.

38. How many of the following lines compile?

```
int i = null;
Integer in = null;
String s = null;
```

- A. None
- B. One
- C. Two**
- D. Three

39. Which pairs of statements can accurately fill in the blanks in this table?

| Variable Type | Can be called within the class from<br>what type of method |
|---------------|------------------------------------------------------------|
| Instance      | Blank 1: _____                                             |
| Static        | Blank 2: _____                                             |

- A. Blank 1: an instance method only, Blank 2: a static method only
- B. Blank 1: an instance or static method, Blank 2: a static method only
- C. Blank 1: an instance method only, Blank 2: an instance or static method**
- D. Blank 1: an instance or static method, Blank 2: an instance or static method

40. Which of the following does not compile?

- A. double num = 2.718;
- B. double num = 2.\_718;**
- C. double num = 2.7\_1\_8;
- D. None of the above; they all compile.

41. Which of the following lists of primitive numeric types is presented in order from smallest to largest data type?

- A. byte, short, int, long**
- B. int, short, byte, long
- C. short, byte, int, long
- D. short, int, byte, long

**42.** Fill in the blank to make the code compile:

```
package animal;
public class Cat {
 public String name;
 public static void main(String[] meow) {
 Cat cat = new Cat();
 _____ = "Sadie";
 }
}
```

- A.** cat.name
- B.** cat-name
- C.** cat.setName
- D.** cat[name]

**43.** Which of the following is the output of this code, assuming it runs to completion?

```
package store;
public class Toy {
 public void play() {
 System.out.print("play-");
 }
 public void finalizer() {
 System.out.print("clean-");
 }
 public static void main(String[] fun) {
 Toy car = new Toy();
 car.play();
 System.gc();
 Toy doll = new Toy();
 doll.play();
 }
}
```

- A.** play-
- B.** play-play-
- C.** play-clean-play-
- D.** play-play-clean-clean-

44. Which is the most common way to fill in the blank to implement this method?

```
public class Penguin {
 private double beakLength;
 public static void setBeakLength(Penguin p, int b) {

 }
}
```

- A. `p.beakLength = b;`
- B. `p['beakLength'] = b;`
- C. `p[beakLength] = b;`
- D. None of the above

45. Fill in the blanks to indicate whether a primitive or wrapper class can be assigned without the compiler using the autoboxing feature.

```
_____ first = Integer.parseInt("5");
_____ second = Integer.valueOf("5");
```

- A. `int, int`
- B. `int, Integer`
- C. `Integer, int`
- D. `Integer, Integer`

46. How many objects are eligible for garbage collection right before the end of the `main` method?

```
1: public class Person {
2: public Person youngestChild;
3:
4: public static void main(String... args) {
5: Person elena = new Person();
6: Person diana = new Person();
7: elena.youngestChild = diana;
8: diana = null;
9: Person zoe = new Person();
10: elena.youngestChild = zoe;
11: zoe = null;
12: }
13: }
```

- A. None
- B. One**
- C. Two
- D. Three

47. Which is a valid constructor for this class?

```
public class TennisBall {
}
```

- A. public TennisBall static create() { return new TennisBall(); }
- B. public TennisBall static newInstance() { return new TennisBall():}
- C. public TennisBall() {}**
- D. public void TennisBall() {}

48. Which of the following is not a possible output of this code, assuming it runs to completion?

```
package store;
public class Toy {
 public void play() {
 System.out.print("play-");
 }
 public void finalize() {
 System.out.print("clean-");
 }
 public static void main(String[] args) {
 Toy car = new Toy();
 car.play();
 System.gc();
 Toy doll = new Toy();
 doll.play();
 }
}
```

- A. play-**
- B. play-play-
- C. play-play-clean-
- D. play-play-clean-clean-

**49.** Which converts a primitive to a wrapper class object without using autoboxing?

- A.** Call the `asObject()` method
- B.** Call the constructor of the wrapper class
- C.** Call the `convertToObject()` method
- D.** Call the `toObject()` method

**50.** What is the output of the following?

```
package beach;
public class Sand {
 public Sand() {
 System.out.print("a");
 }
 public void Sand() {
 System.out.print("b");
 }
 public void run() {
 new Sand();
 Sand();
 }
 public static void main(String... args) {
 new Sand().run();
 }
}
```

- A.** a
- B.** ab
- C.** aab
- D.** None of the above



# Chapter 3

# Using Operators and Decision Constructs

---

**THE OCA EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Using Operators and Decision Constructs**

- Use Java operators; use parentheses to override operator precedence
- Test equality between Strings and other objects using == and equals()
- Create if and if/else and ternary constructs
- Use a switch statement

1. Which of the following variable types is not permitted in a switch statement?

- A. String
- B. double
- C. int
- D. char

2. What is the value of tip after executing the following code snippet?

```
int meal = 5;
int tip = 2;
int total = meal + (meal>6 ? ++tip : --tip);
```

- A. 1
- B. 2
- C. 3
- D. 6

3. What is the output of the following application?

```
package registration;
public class NameCheck {
 public static void main(String... data) {
 String john = "john";
 String jon = new String(john);
 System.out.print((john==jon)+" "+(john.equals(jon)));
 }
}
```

- A. true true
- B. true false
- C. false true
- D. false false

4. What is the output of the following application?

```
package planning;
public class ThePlan {
 public static void main(String[] input) {
 int plan = 1;
 plan = plan++ + --plan;
 if(plan==1) {
 System.out.print("Plan A");
 } else { if(plan==2) System.out.print("Plan B");
 } else System.out.print("Plan C");
 }
}
```

- A. Plan A
  - B. Plan B
  - C. Plan C
  - D. None of the above**
5. Which of the following statements about a `default` branch in a `switch` statement is correct?
- A. All `switch` statements must include a `default` statement.
  - B. The `default` statement is required to be placed after all `case` statements.
  - C. Unlike a `case` statement, the `default` statement does not take a value.**
  - D. A `default` statement can only be used when at least one `case` statement is present.
6. What is the value of `thatNumber` after the execution of the following code snippet?
- ```
long thatNumber = 5 >= 5 ? 1+2 : 1*1;  
if(++thatNumber < 4)  
    thatNumber += 1;
```
- A. 3**
 - B. 4**
 - C. 5
 - D. The answer cannot be determined until runtime.
7. Which statement immediately exits a `switch` statement, skipping all remaining `case` or `default` branches?
- A. `exit`
 - B. `break`**
 - C. `goto`
 - D. `continue`
8. Which statement about ternary expressions is true?
- A. In some cases, both expressions to the right of the conditional operator in a ternary expression will be evaluated at runtime.
 - B. Ternary expressions require parentheses for proper evaluation.
 - C. The ternary expressions are a convenient replacement for an `if-then-else` statement.**
 - D. Ternary expressions support `int` and `boolean` expressions for the left-most operand.
9. What is the output of the following application?

```
package voting;  
1: public class Election {  
2:     public void calculateResult(Integer candidateA, Integer candidateB) {  
3:         boolean process = candidateA == null || candidateA.intValue() < 10;  
4:         boolean value = candidateA && candidateB;  
5:         System.out.print(process || value);
```

```
6:      }
7:      public static void main(String[] unused) {
8:          new Election().calculateResult(null,203);
9:      }
10: }
```

- A.** true
- B.** false
- C.** The code does not compile.
- D.** The code compiles but throws a `NullPointerException` on line 3 at runtime.

- 10.** What is the output of the following application?

```
package dinosaur;
public class Park {
    public final static void main(String... arguments) {
        int pterodactyl = 6;
        long triceratops = 3;
        if(pterodactyl % 3 >= 1)
            triceratops++;
        triceratops--;
        System.out.print(triceratops);
    }
}
```

- A.** 2
- B.** 3
- C.** 4
- D.** The code does not compile.

- 11.** Which statement about if-then statements is true?

- A.** An if-then statement is required to have an `else` statement.
- B.** If the boolean test of an if-then statement evaluates to `false`, then the target clause of the if-then statement will still be evaluated.
- C.** An if-then statement is required to cast an object.
- D.** An if-then statement can execute a single statement or a block {}.

- 12.** What is the output of the following application?

```
package restaurant;
public class Pieces {
    public static void main(String[] info) {
        int flair = 15;
        if(flair >= 15 && flair < 37) {
            System.out.print("Not enough");
        }
    }
}
```

```
        } if(flair==37) {  
            System.out.print("Just right");  
        } else {  
            System.out.print("Too many");  
        }  
    }  
}
```

- A. Not enough
- B. Just right
- C. Too many
- D. None of the above**

13. Which statement about case statements of a switch statement is not true?

- A. A case value can be final.
- B. A case statement must be terminated with a break statement.**
- C. A case value can be a literal expression.
- D. A case value must match the data type of the switch variable, or be able to be promoted to that type.

14. Given the following truth table, which operator for the boolean expressions x and y corresponds to this relationship?

	x = true	x = false
y = true	true	false
y = false	false	false

- A. --
- B. ++
- C. ||
- D. &&**

15. What is the output of the following code snippet?

```
int hops = 0;  
int jumps = 0;  
jumps = hops++;  
if(jumps)  
    System.out.print("Jump!");  
else  
    System.out.print("Hop!");
```

- A. Jump!
- B. Hop!
- C. The code does not compile.**
- D. The code compiles but throws an exception at runtime.
16. Fill in the blanks: The _____ operator increases the value of a variable by 1 and returns the new value, while the _____ operator decreases the value of a variable by 1 and returns the original value.
- A. pre-increment [++v], pre-decrement [--v]
- B. pre-increment [++v], post-decrement [v--]**
- C. post-increment [v++], pre-decrement [--v]
- D. post-increment [v++], post-decrement [v--]
17. What is the output of the following application?
- ```
package jungle;
public class TheBigRace {
 public static void main(String[] in) {
 int tiger = 2;
 short lion = 3;
 long winner = lion+2*(tiger + lion);
 System.out.print(winner);
 }
}
```
- A. 11
- B. 13**
- C. 25
- D. None of the above

18. Given the following code snippet, assuming dayOfWeek is an int, what variable type of saturday is not permitted?

```
final ____ saturday = 6;
switch(dayOfWeek) {
 default:
 System.out.print("Another Weekday");
 break;
 case saturday:
 System.out.print("Weekend!");
}
```

- A. byte
- B. long**
- C. int
- D. None of the above

19. Given the following code snippet, what is the value of dinner after it is executed?

```
int time = 11;
int day = 4;
String dinner = time > 10 ? day ? "Takeout" : "Salad" : "Leftovers";
```

- A. Takeout
- B. Salad
- C. The code does not compile but would compile if parentheses were added.
- D. None of the above**

20. What is the output of the following application?

```
package recreation;
public class Dancing {
 public static void main(String[] vars) {
 int leaders = 10 * (2 + (1 + 2 / 5));
 int followers = leaders * 2;
 System.out.print(leaders + followers < 10 ? "Too few" : "Too many");
 }
}
```

- A. Too few
- B. Too many
- C. The code does not compile.**
- D. The code compiles but throws a division by zero error at runtime.

21. What is the output of the following application?

```
package schedule;
public class PrintWeek {
 public static final void main(String[] days) {
 System.out.print(5 + 6 + "7" + 8 + 9);
 }
}
```

- A. 56789
- B. 11789**
- C. 11717
- D. The code does not compile.

22. Fill in the blanks: The \_\_\_\_\_ operator is used to find the difference between two numbers, while the \_\_\_\_\_ operator is used to find the remainder when one number is divided by another.

- A. /, %
- B. -, %**
- C. %, <
- D. -, ||

23. What is the output of the following application?

```
package transporter;
public class Rematerialize {
 public static void main(String[] input) {
 int dog = 11;
 int cat = 3;
 int partA = dog / cat;
 int partB = dog % cat;
 int newDog = partB + partA * cat;
 System.out.print(newDog);
 }
}
```

- A. 9
- B. 11**
- C. 15
- D. The code does not compile.

24. What is the output of the following application?

```
package dessert;
public class IceCream {
 public final static void main(String... args) {
 int flavors = 30;
 int eaten = 0;
 switch(flavors) {
 case 30: eaten++;
 case 40: eaten+=2;
 default: eaten--;
 }
 System.out.print(eaten);
 }
}
```

- A. 1  
**B. 2**  
C. 3  
D. The code does not compile.
25. What is the output of the following application?
- ```
package mode;
public class Transportation {
    public static String travel(int distance) {
        return distance<1000 ? "train" : 10;
    }
    public static void main(String[] answer) {
        System.out.print(travel(500));
    }
}
```
- A. train
B. 10
C. The code does not compile.
D. The code compiles but throws an exception at runtime.
26. Fill in the blanks: Given two non-null `String` objects with reference names `apples` and `oranges`, if `apples _____ oranges` evaluates to `true`, then `apples _____ oranges` must also evaluate to `true`.
- A. `==, equals()`
B. `!=, equals()`
C. `equals(), ==`
D. `equals(), !=`
27. For a given non-null `String` `myTestVariable`, what is the resulting value of executing the statement `myTestVariable.equals(null)`?
- A. `true`
B. `false`
C. The statement does not compile.
D. The statement compiles but will produce an exception when used at runtime.
28. How many 1s are outputted when the following application is compiled and run?
- ```
package city;
public class Road {
 public static void main(String... in) {
 int intersections = 100;
```

```
int streets = 200;
if (intersections < 150) {
 System.out.print("1");
} else if (streets && intersections > 1000) {
 System.out.print("2");
} if (streets < 500)
 System.out.print("1");
else
 System.out.print("2");
}
```

- A. None
- B. One
- C. Two
- D. The code does not compile.**

**29.** Which statement about the logical operators & and && is true?

- A. The & and && operators are interchangeable, always producing the same results at runtime.
- B. The & operator always evaluates both operands, while the && operator may only evaluate the left operand.**
- C. Both expressions evaluate to true if either operand is true.
- D. The & operator always evaluates both operands, while the && operator may only evaluate the right operand.

**30.** What is the output of the following code snippet?

```
int x = 10, y = 5;
boolean w = true, z = false;
x = w ? y++ : y--;
w = !z;
System.out.print((x+y)+" "+(w ? 5 : 10));
```

- A. The code does not compile.
- B. 10 10
- C. 11 5**
- D. 12 5

**31.** What is the output of the following application?

```
package bob;
public class AreYouBob {
 public static void main(String[] unused) {
```

```
String bob = new String("bob");
String notBob = bob;
System.out.print((bob==notBob)+" "+(bob.equals(notBob)));
}
}
A. true true
B. true false
C. false true
D. false false
```

32. What is the value of  $12 + 6 * 3 \% (1 + 1)$  in Java?

- A. 0
- B. 12**
- C. 14
- D. None of the above

33. Given the following truth table, the boolean variables p and q, and the expression  $p \wedge q$ , what are the missing values in the truth table, starting with the first column?

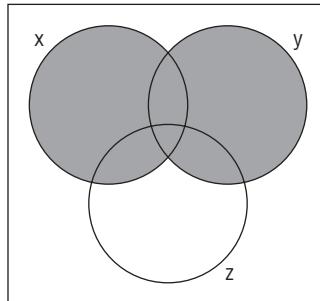
|                  | <b>p = true</b> | <b>p = false</b> |
|------------------|-----------------|------------------|
| <b>q = true</b>  | false           | true             |
| <b>q = false</b> |                 |                  |

- A. false and true
- B. false and false
- C. true and true
- D. true and false**

34. Which of the following is not a possible result of executing the following application?

```
public class ConditionallyLogical {
 public static void main(String... data) {
 if(data.length>=1
 && (data[0].equals("sound") || data[0].equals ("logic"))
 && data.length<2) {
 System.out.print(data[0]);
 }
 }
}
```

- A. Nothing is printed.  
B. sound is printed.  
**C. The application throws an exception at runtime.**  
D. logic is printed.
35. Fill in the blanks: The operators +, \_\_\_\_\_, \_\_\_\_\_, and ++ are listed in the same or increasing level of operator precedence.  
A. \*, --, /  
B. %, -, \*  
**C. /, \*, %**  
D. \*, -, /
36. What statement about the `^` operator is correct?  
A. If one of the operands of `^` is true, then the result is always true.  
B. There is a conditional form of the operator, denoted as `^^`.  
C. If both operands of `^` are true, the result is true.  
**D. The `^` operator can only be applied to boolean values.**
37. Given the following Venn diagram and the variables, x, y, and z, which Java expression most closely represents the filled-in region of the diagram?



- A. x || z**  
B. y || (y && z)  
**C. x || y**  
D. y && x
38. What variable type of red allows the following application to compile?

```
package tornado;
public class Kansas {
 public static void main(String[] args) {
 int colorOfRainbow = 10;
 _____ red = 5;
```

```
switch(colorOfRainbow) {
 default:
 System.out.print("Home");
 break;
 case red:
 System.out.print("Away");
 }
}
}
```

- A.** long  
**B.** double  
**C.** int  
**D.** None of the above
- 39.** Which two operators would be used to test if a number is equal to or greater than 5.21 but strictly less than 8.1?
- A.** > and <=   
**B.** >= and >  
**C.** < and >=   
**D.** < and >
- 40.** What is the output of the following application?
- ```
package transporter;  
public class TurtleVsHare {  
    public static void main(String[] arguments) {  
        int turtle = 10 * (2 + (3 + 2) / 5);  
        int hare = turtle < 5 ? 10 : 25;  
        System.out.print(turtle < hare ? "Hare wins!" : "Turtle wins!");  
    }  
}
```
- A.** Hare wins!
B. Turtle wins!
C. The code does not compile.
D. The code compiles but throws a division by zero error at runtime.

- 41.** What is the output of the following application?

```
public class CountEntries {  
    public static int getResult(int threshold) {  
        return threshold > 5 ? 1 : 0;
```

```
    }  
  
    public static final void main(String[] days) {  
        System.out.print(getResult(5)+getResult(1)  
            +getResult(0)+getResult(2)+"");  
    }  
}
```

- A.** 0
- B.** 1
- C.** 0000
- D.** 1000

- 42.** What is the output of the following application?

```
package yoyo;  
public class TestGame {  
    public String runTest(boolean spinner, boolean roller) {  
        if(spinner = roller) return "up";  
        else return roller ? "down" : "middle";  
    }  
    public static final void main(String pieces[]) {  
        final TestGame tester = new TestGame();  
        System.out.println(tester.runTest(false,true));  
    }  
}
```

- A.** up
- B.** middle
- C.** down
- D.** The code does not compile.

- 43.** Fill in the blanks: The _____ operator is true if either of the operands are true, while the _____ operator flips a boolean value.

- A.** +, -
- B.** &&, !
- C.** |, -
- D.** ||, !

- 44.** Given the following code snippet, what is the value of `movieRating` after it is executed?

```
int characters = 5;  
int story = 3;  
double movieRating = characters <= 4 ? 3 : story>1 ? 2 : 1;
```

- A. 2.0**
- B. 3.0**
- C. The code does not compile but would compile if parentheses were added.**
- D. None of the above**
- 45.** Fill in the blanks: A switch statement can have _____ case statements and _____ default statements.
- A. at most one, at least one**
- B. any number of, at most one**
- C. at least one, any number of**
- D. at least one, at most one**
- 46.** Which of the following is not a possible result of executing the following application?
- ```
public class OutsideLogic {
 public static void main(String... weather) {
 System.out.print(weather[0] != null
 && weather[0].equals("sunny")
 && !false
 ? "Go Outside" : "Stay Inside");
 }
}
```
- A. Nothing is printed.**
- B. The application throws an exception at runtime.**
- C. Go Outside is printed.**
- D. Stay Inside is printed.**
- 47.** What is the value of  $(5 + (!2 + 8) * 3 - 3 \% 2)/2$  in Java?
- A. 2**
- B. 11**
- C. 16**
- D. None of the above**
- 48.** Given the following truth table, the boolean variables w and z, and the expression  $w \mid\mid z$ , what are the missing values in the truth table, starting with the first row?

| <b>w = true</b>  | <b>w = false</b> |
|------------------|------------------|
| <b>z = true</b>  | true             |
| <b>z = false</b> | false            |

- A. false and false  
B. true and false  
**C. true and true**  
D. false and true
49. Fill in the blanks: The operators -, \_\_\_\_\_, \_\_\_\_\_, and % are listed in the same or increasing level of operator precedence.  
A. +, /, \*  
B. --, -, \*  
C. ++, /, \*  
D. \*, ++, %
50. What is the output of the following application?
- ```
public class Baby {  
    public static String play(int toy, int age) {  
        final String game;  
        if(toy<2)  
            game = age > 1 ? 1 : 10; // p1  
        else  
            game = age > 3 ? "Ball" : "Swim"; // p2  
        return game;  
    }  
    public static void main(String[] variables) {  
        System.out.print(play(5,2));  
    }  
}
```
- A. Ball
B. Swim
C. The code does not compile due to p1.
D. The code does not compile due to p2.

Chapter 4



Creating and Using Arrays

**THE OCA EXAM TOPICS COVERED
IN THIS PRACTICE TEST INCLUDE
THE FOLLOWING:**

✓ **Creating and Using Arrays**

- Declare, instantiate, initialize and use a one-dimensional array
- Declare, instantiate, initialize and use multi-dimensional arrays

1. What symbol is used for a varargs method parameter?

- A. ..
- B. ...
- C. --
- D. ---

2. Fill in the blank in the following code to get the first element from the varargs parameter.

```
public void toss (Frisbee... f) {  
    Frisbee first = _____;  
}
```

- A. f
- B. f[0]
- C. f[1]
- D. None of the above

3. Which of the following are primitives?

```
int[] lowercase = new int[0];  
Integer[] uppercase = new Integer[0];
```

- A. Only lowercase
- B. Only uppercase
- C. Both lowercase and uppercase
- D. Neither lowercase nor uppercase

4. How many of the following are legal declarations?

```
[]double lion;  
double[] tiger;  
double bear[];
```

- A. None
- B. One
- C. Two
- D. Three

5. Given the following two methods, which method call will not compile?

```
public void printStormName(String... names) {  
    System.out.println(Arrays.toString(names));  
}  
public void printStormNames(String[] names) {  
    System.out.println(Arrays.toString(names));  
}
```

- A. `printStormName("Arlene");`
 - B. `printStormName(new String[] { "Bret" });`
 - C. `printStormNames("Cindy");`
 - D. `printStormNames(new String[] { "Don" });`
6. How do you determine the number of elements in an array?
- A. `buses.length`
 - B. `buses.length()`
 - C. `buses.size`
 - D. `buses.size()`
7. Which of the following create an empty two-dimensional array with dimensions 2x2?
- A. `int[][] blue = new int[2, 2];`
 - B. `int[][] blue = new int[2], [2];`
 - C. `int[][] blue = new int[2][2];`
 - D. `int[][] blue = new int[2 x 2];`
8. How many lines does the following code output?
- ```
String[] days = new String[] { "Sunday", "Monday", "Tuesday",
 "Wednesday", "Thursday", "Friday", "Saturday" };
for (int i = 0; i < days.length; i++)
 System.out.println(days[i]);
```
- A. Six
  - B. Seven
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
9. What are the names of the methods to do searching and sorting respectively on arrays?
- A. `Arrays.binarySearch()` and `Arrays.linearSort()`
  - B. `Arrays.binarySearch()` and `Arrays.sort()`
  - C. `Arrays.search()` and `Arrays.linearSort()`
  - D. `Arrays.search()` and `Arrays.sort()`
10. What does this code output?
- ```
String[] nums = new String[] { "1", "9", "10" };
Arrays.sort(nums);
System.out.println(Arrays.toString(nums));
```
- A. `[1, 9, 10]`
 - B. `[1, 10, 9]`
 - C. `[10, 1, 9]`
 - D. None of the above

11. Which of the following references the first and last element in a non-empty array?

- A. `trains[0]` and `trains[trains.length]`
- B. `trains[0]` and `trains[trains.length - 1]`
- C. `trains[1]` and `trains[trains.length]`
- D. `trains[1]` and `trains[trains.length - 1]`

12. How many of the following are legal declarations?

```
String lion [] = new String[] {"lion"};
String tiger [] = new String[1] {"tiger"};
String bear [] = new String[] {};
String ohMy [] = new String[0] {};
```

- A. None
- B. One
- C. Two
- D. Three

13. How many of the following are legal declarations?

```
float[] lion = new float[];
float[] tiger = new float[1];
float[] bear = new[] float;
float[] ohMy = new[1] float;
```

- A. None
- B. One
- C. Two
- D. Three

14. Which statement most accurately represents the relationship between searching and sorting with respect to the `Arrays` class?

- A. If the array is not sorted, calling `Arrays.binarySearch()` will be accurate, but slower than if it were sorted.
- B. The array does not need to be sorted before calling `Arrays.binarySearch()` to get an accurate result.
- C. The array must be sorted before calling `Arrays.binarySearch()` to get an accurate result.
- D. None of the above

15. Which is not a true statement about an array?

- A. An array expands automatically when it is full.
- B. An array is allowed to contain duplicate values.
- C. An array understands the concept of ordered elements.
- D. An array uses a zero index to reference the first element.

16. Which line of code causes an `ArrayIndexOutOfBoundsException`?

```
String[][] matrix = new String[1][2];
matrix[0][0] = "Don't think you are, know you are.";           // m1
matrix[0][1] = "I'm trying to free your mind Neo";           // m2
matrix[1][0] = "Is all around you ";                           // m3
matrix[1][1] = "Why oh why didn't I take the BLUE pill?";   // m4
```

- A. m1
- B. m2
- C. m3
- D. m4

17. What does the following output?

```
String[] os = new String[] { "Mac", "Linux", "Windows" };
Arrays.sort(os);
System.out.println(Arrays.binarySearch(os, "Mac"));
```

- A. 0
- B. 1
- C. 2
- D. The output is not defined.

18. Which is the first line to prevent this code from compiling and running without error?

```
char[][] ticTacToe = new char[3,3];                      // r1
ticTacToe[1][3] = 'X';                                     // r2
ticTacToe[2][2] = 'X';
ticTacToe[3][1] = 'X';
System.out.println(ticTacToe.length + " in a row!"); // r3
```

- A. Line r1
- B. Line r2
- C. Line r3
- D. None of the above

19. How many objects are created when running the following code?

```
Integer[] lotto = new Integer[4];
lotto[0] = new Integer(1_000_000);
lotto[1] = new Integer(999_999);
```

- A. Two
- B. Three
- C. Four
- D. Five

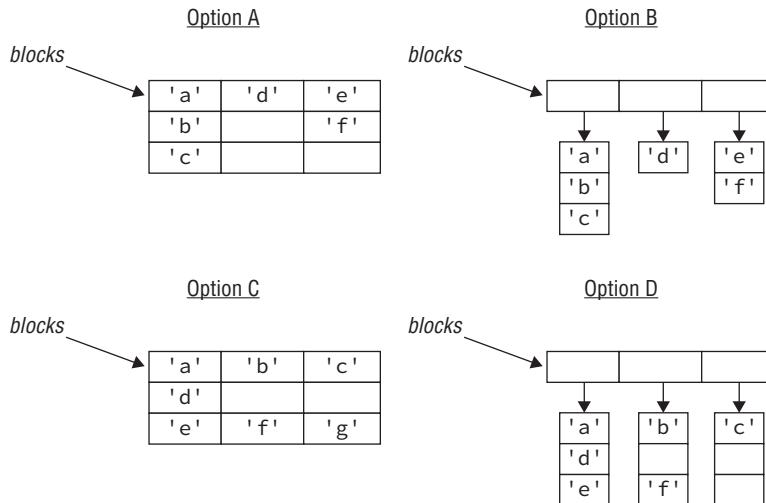
20. How many of the following are legal declarations?

```
[][] String alpha;
[] String beta;
String[][] gamma;
String[] delta[];
String epsilon[][];
```

- A. Two
- B. Three
- C. Four
- D. Five

21. Which of the options in the graphic best represent the `blocks` variable?

```
char[][] blocks = new char[][] { { 'a', 'b', 'c' }, { 'd' }, { 'e', 'f' } };
```



- A. Option A
- B. Option B
- C. Option C
- D. Option D

22. What happens when calling the following method with a non-null and non-empty array?

```
public static void addStationName(String[] names) {
    names[names.length] = "Times Square";
}
```

- A. It adds an element to the array the value of which is Times Square.
 - B. It replaces the last element in the array with the value Times Square.
 - C. It does not compile.
 - D. It throws an exception.
- 23.** How many lines does the following code output?
- ```
String[] days = new String[] { "Sunday", "Monday", "Tuesday",
 "Wednesday", "Thursday", "Friday", "Saturday" };
for (int i = 0; i < days.size(); i++)
 System.out.println(days[i]);
```
- A. Six
  - B. Seven
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
- 24.** How many dimensions does the array reference moreBools allow?
- ```
boolean[][][] bools, moreBools;
```
- A. One dimension
 - B. Two dimensions
 - C. Three dimensions
 - D. None of the above
- 25.** What is a possible output of the following code?
- ```
String[] strings = new String[2];
System.out.println(strings);
```
- A. [null, null]
  - B. [, ]
  - C. [Ljava.lang.String;@74a14482
  - D. None of the above
- 26.** Which is the first line to prevent this code from compiling and running without error?

```
char[][] ticTacToe = new char[3][3]; // r1
ticTacToe[1][3] = 'X'; // r2
ticTacToe[2][2] = 'X';
ticTacToe[3][1] = 'X';
System.out.println(ticTacToe.length + " in a row!"); // r3
```

- A. Line r1
- B. Line r2
- C. Line r3
- D. None of the above

27. What is the result of running the following as java Copier?

```
package duplicate;
public class Copier {
 public static void main(String... original) {
 String... copy = original;
 System.out.println(copy.length + " " + copy[0]);
 }
}
```

- A. 0
- B. 0 followed by an exception
- C. 1 followed by an exception
- D. The code does not compile.

28. What is the result of running the following program?

```
1: package fun;
2: public class Sudoku {
3: static int[][] game = new int[6][6];
4:
5: public static void main(String[] args) {
6: game[3][3] = 6;
7: Object[] obj = game;
8: obj[3] = "X";
9: System.out.println(game[3][3]);
10: }
11: }
```

- A. X
- B. The code does not compile.
- C. The code compiles but throws a `NullPointerException` at runtime.
- D. The code compiles but throws a different exception at runtime.

29. What does the following output?

```
String[] os = new String[] { "Mac", "Linux", "Windows" };
Arrays.sort(os);
System.out.println(Arrays.binarySearch(os, "RedHat"));
```

- A. -1
- B. -2
- C. -3
- D. The output is not defined.

30. What is the output of the following when run as `java FirstName Wolfie`?

```
public class FirstName {
 public static void main(String... names) {
 System.out.println(names[0]);
 }
}
```

- A. FirstName
- B. Wolfie
- C. The code throws an `ArrayIndexOutOfBoundsException`.
- D. The code throws a `NullPointerException`.

31. What is the output of the following when run as `java Count 1 2`?

```
public class Count {
 public static void main(String target[]) {
 System.out.println(target.length);
 }
}
```

- A. 0
- B. 1
- C. 2
- D. The code does not compile.

32. What is the output of the following when run as `java unix.EchoFirst seed flower`?

```
package unix;
import java.util.*;
public class EchoFirst {

 public static void main(String[] args) {
 String one = args[0];
 Arrays.sort(args);
 int result = Arrays.binarySearch(args, one);
 }
}
```

```

 System.out.println(result);
 }
}

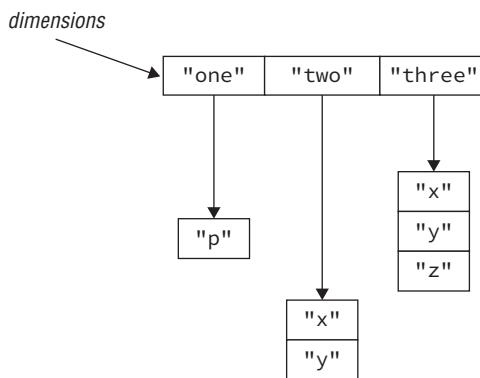
```

- A.** 0  
**B.** 1  
**C.** The code does not compile.  
**D.** The code compiles but throws an exception at runtime.

**33.** Which of these four array declarations produces a different array than the others?

- A.** `int[][] nums = new int[2][1];`  
**B.** `int[] nums[] = new int[2][1];`  
**C.** `int[] nums[] = new int[][] { { 0 }, { 0 } };`  
**D.** `int[] nums[] = new int[][] { { 0, 0 } };`

**34.** How do you access the array element with the value of "z"?



- A.** `dimensions["three"][2]`  
**B.** `dimensions["three"][3]`  
**C.** `dimensions[2][2]`  
**D.** `dimensions[3][3]`

**35.** How many lines does the following code output?

```

String[] days = new String[] { "Sunday", "Monday", "Tuesday",
 "Wednesday", "Thursday", "Friday", "Saturday" };
for (int i = 1; i <= days.length; i++)
 System.out.println(days[i]);

```

- A.** Six  
**B.** Seven  
**C.** The code does not compile.  
**D.** The code compiles but throws an exception at runtime.

36. What is the output of the following when run as `java FirstName Wolfie`?

```
public class FirstName {
 public static void main(String... names) {
 System.out.println(names[1]);
 }
}
```

- A.** FirstName
- B.** Wolfie
- C.** The code throws an `ArrayIndexOutOfBoundsException`.
- D.** The code throws a `NullPointerException`.

37. Which is the first line to prevent this code from compiling and running without error?

```
char[][] ticTacToe = new char[3][3]; // r1
ticTacToe[0][0] = 'X'; // r2
ticTacToe[1][1] = 'X';
ticTacToe[2][2] = 'X';
System.out.println(ticTacToe.length + " in a row!"); // r3
```

- A.** Line r1
- B.** Line r2
- C.** Line r3
- D.** None of the above

38. What is the output of the following when run as `java Count 1 2`?

```
public class Count {
 public static void main(String target[]) {
 System.out.println(target.length());
 }
}
```

- A.** 0
- B.** 1
- C.** 2
- D.** The code does not compile.

39. How many dimensions does the array reference `moreBools` allow?

```
boolean[][] bools[], moreBools;
```

- A.** One dimension
- B.** Two dimensions
- C.** Three dimensions
- D.** None of the above

40. What is the result of the following when called as `java counting.Binary`?

```
package counting;
import java.util.*;
public class Binary {

 public static void main(String... args) {
 Arrays.sort(args);
 System.out.println(Arrays.toString(args));
 }
}
```

- A. `null`
  - B. `[]`
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
41. What does the following output?

```
String[] os = new String[] { "Mac", "Linux", "Windows" };
System.out.println(Arrays.binarySearch(os, "Linux"));
```

- A. 0
  - B. 1
  - C. 2
  - D. The output is not defined.
42. What is the result of running the following program?

```
1: package fun;
2: public class Sudoku {
3: static int[][] game;
4:
5: public static void main(String[] args) {
6: game[3][3] = 6;
7: Object[] obj = game;
8: game[3][3] = "X";
9: System.out.println(game[3][3]);
10: }
11: }
```

- A. X
- B. The code does not compile.
- C. The code compiles but throws a `NullPointerException` at runtime.
- D. The code compiles but throws a different exception at runtime.

**43.** What is the output of the following?

```
String[][][] listing = new String[][][] { { "Book" }, { "Game", "29.99" } };
System.out.println(listing.length + " " + listing[0].length);
```

- A.** 2 1
- B.** 2 2
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

**44.** What is the output of the following when run as `java FirstName`?

```
public class FirstName {
 public static void main(String[] names) {
 System.out.println(names[0]);
 }
}
```

- A.** FirstName
- B.** The code does not compile.
- C.** The code throws an `ArrayIndexOutOfBoundsException`.
- D.** The code throws a `NullPointerException`.

**45.** How many lines does the following code output?

```
String[] days = new String[] { "Sunday", "Monday", "Tuesday",
 "Wednesday", "Thursday", "Friday", "Saturday" };
for (int i = 1; i < days.length; i++)
 System.out.println(days[i]);
```

- A.** Six
- B.** Seven
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

**46.** What is the output of the following when run as `java Count "1 2"`?

```
public class Count {
 public static void main(String target[]) {
 System.out.println(target.length);
 }
}
```

- A.** 0
- B.** 1
- C.** 2
- D.** The code does not compile.

**47.** What does the following output?

```
String[] os = new String[] { "Linux", "Mac", "Windows" };
System.out.println(Arrays.binarySearch(os, "Linux"));
```

- A.** 0
- B.** 1
- C.** 2
- D.** The output is not defined.

**48.** Which of the following statements are true?

- I.** You can always change a method signature from `call(String[] arg)` to `call(String... arg)` without causing a compiler error in the calling code.
- II.** You can always change a method signature from `call(String... arg)` to `call(String[] arg)` without causing a compiler error in the existing code.

- A.** I
- B.** II
- C.** Both I and II
- D.** Neither I nor II

**49.** Which of these four array references can point to an array that is different from the others?

- A.** `int[][][][] nums1a, nums1b;`
- B.** `int[][][] nums2a[], nums2b;`
- C.** `int[][] nums3a[][][], nums3b[][][];`
- D.** `int[] nums4a[][][], nums4b[][][];`

**50.** What is the output of the following when run as `java unix.EchoFirst seed flower?`

```
package unix;
import java.util.*;
public class EchoFirst {

 public static void main(String[] args) {
 Arrays.sort(args);
 String result = Arrays.binarySearch(args, args[0]);
 System.out.println(result);
 }
}
```

- A.** 0
- B.** 1
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

# Chapter 5



## Using Loop Constructs

---

**THE OCA EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Using Loop Constructs**

- Create and use while loops
- Create and use for loops including the enhanced for loop
- Create and use do/while loops
- Compare loop constructs
- Use break and continue

1. Which type of loop is best known for its boolean condition that controls entry to the loop?
  - A. do-while loop
  - B. for (traditional)
  - C. for-each
  - D. while
2. Which type of loop is best known for using an index or counter?
  - A. do-while loop
  - B. for (traditional)
  - C. for-each
  - D. while
3. Which type of loop is guaranteed to have the body execute at least once?
  - A. do-while loop
  - B. for (traditional)
  - C. for-each
  - D. while
4. Which of the following can loop through an array without referring to the elements by index?
  - A. do-while loop
  - B. for (traditional)
  - C. for-each
  - D. while
5. What keyword is used to end the current loop iteration and proceed execution with the next iteration of that loop?
  - A. break
  - B. continue
  - C. end
  - D. skip
6. What keyword is used to proceed with execution immediately after a loop?
  - A. break
  - B. continue
  - C. end
  - D. skip
7. Which type of loop has three segments within parentheses?
  - A. do-while loop
  - B. for (traditional)
  - C. for-each
  - D. while

- 8.** Which of the following statements is/are true?
- I.** A traditional for loop can iterate through an array starting from index 0.
  - II.** A traditional for loop can iterate through an array starting from the end.
- A.** Only I
  - B.** Only II
  - C.** Both statements
  - D.** Neither statement
- 9.** Which of the following statements is/are true?
- I.** A for-each loop can iterate through an array starting from index 0.
  - II.** A for-each loop can iterate through an array starting from the end.
- A.** Only I
  - B.** Only II
  - C.** Both statements
  - D.** Neither statement
- 10.** Which type of loop has a boolean condition that is first checked after a single iteration through the loop?
- A.** do-while loop
  - B.** for (traditional)
  - C.** for-each
  - D.** while
- 11.** What does the following code output?
- ```
int singer = 0;
while (singer)
    System.out.println(singer++);
```
- A.** 0
 - B.** The code does not compile.
 - C.** The loops complete with no output.
 - D.** This is an infinite loop.
- 12.** What does the following code output?
- ```
List<String> drinks = Arrays.asList("can", "cup");
for (int container = drinks.size() - 1; container >= 0; container--)
 System.out.print(drinks.get(container) + ",");
```
- A.** can,cup,
  - B.** cup,can,
  - C.** The code does not compile.
  - D.** None of the above

**13.** What does the following code output?

```
public static void main(String[] args) {
 List<String> bottles = Arrays.asList("glass", "plastic");
 for (int type = 0; type < bottles.size();) {
 System.out.print(bottles.get(type) + ",");
 break;
 }
 System.out.print("end");
}
```

- A.** glass,end
- B.** glass,plastic,end
- C.** The code does not compile.
- D.** None of the above

**14.** What does the following code output?

```
String letters = "";
while (letters.length() != 2)
 letters+="a";
System.out.println(letters);
```

- A.** aa
- B.** aaa
- C.** The loops complete with no output.
- D.** This is an infinite loop.

**15.** What is the result of the following when run with  
java peregrine.TimeLoop September 3 1940?

```
package peregrine;
public class TimeLoop {
 public static void main(String[] args) {
 for (int i = args.length; i>=0; i++)
 System.out.println("args");
 }
}
```

- A.** args
- B.** argsargs
- C.** The code does not compile.
- D.** None of the above

16. What is the output of the following code?

```
package chicago;
public class Loop {
 private static int count;
 private static String[] stops = new String[] { "Washington",
 "Monroe", "Jackson", "LaSalle" };
 public static void main(String[] args) {
 while (count < stops.length) {
 if (stops[count++].length() < 8) {
 break;
 }
 }
 System.out.println(count);
 }
}
```

- A. 1
- B. 2
- C. 4
- D. The code does not compile.

17. What is the result of the following code?

```
do {
 int count = 0;
 do {
 count++;
 } while (count < 2);
 break;
} while (true);
System.out.println(count);
```

- A. 2
- B. 3
- C. The code does not compile.
- D. This is an infinite loop.

18. Which of the following segments of a `for` loop can be left blank?

```
for (segmentA; segmentB; segmentC) {
}
```

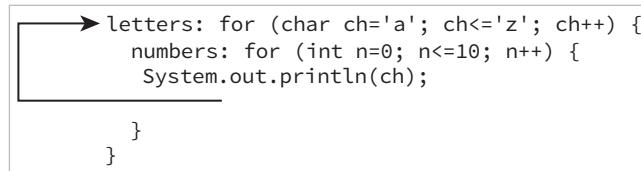
- A. segmentA
  - B. segmentB
  - C. segmentC
  - D. All of the above
19. How many of the loop types (while, do while, traditional for, and enhanced for) allow you to write code that creates an infinite loop?
- A. One
  - B. Two
  - C. Three
  - D. Four
20. What is the output of the following?
- ```
List<String> drinks = Arrays.asList("can", "cup");
for (int container = 0; container < drinks.size(); container++)
    System.out.print(drinks.get(container) + ",");

A. can,cup,
B. cup,can,
C. The code does not compile.
D. None of the above
```
21. What happens when running the following code?
- ```
do (
 System.out.println("helium");
) while (false);

A. It completes successfully without output.
B. It outputs helium once.
C. It keeps outputting helium.
D. The code does not compile.
```
22. Which of the following is equivalent to this code snippet given an array of String objects?
- ```
for (int i=0; i<fun.length; i++)
    System.out.println(fun[i]);
```
- A. for (String f = fun) System.out.println(f);
 - B. for (String f : fun) System.out.println(f);
 - C. for (String = fun) System.out.println(it);
 - D. None of the above

23. How many of these statements can be inserted after the `println` to have the code flow follow the arrow in this diagram?

```
break;  
break letters;  
break numbers;
```



- A. None
 - B. One
 - C. Two
 - D. Three
24. Using the diagram in the previous question, how many of these statements can be inserted after the `println` to have the code flow follow the arrow in the diagram?

```
continue;  
continue letters;  
continue numbers;
```

- A. None
- B. One
- C. Two
- D. Three

25. What does the following code output?

```
int singer = 0;  
while (singer > 0)  
    System.out.println(singer++);
```

- A. 0
- B. The code does not compile.
- C. The loops completes with no output.
- D. This is an infinite loop.

26. Which of the following types is `taxis` not allowed to be in order for this code to compile?

```
for (Object obj : taxis) {  
}
```

- A. `ArrayList<Integer>`
- B. `int[]`
- C. `StringBuilder`
- D. All of these are allowed.

27. What is the output of the following?

```
boolean balloonInflated = false;
do {
    if (!balloonInflated) {
        balloonInflated = true;
        System.out.print("inflate-");
    }
} while (! balloonInflated);
System.out.println("done");
```

- A. `done`
- B. `inflate-done`
- C. The code does not compile.
- D. This is an infinite loop.

28. What does the following code output?

```
String letters = "";
while (letters.length() != 3)
    letters+="ab";
System.out.println(letters);
```

- A. `ab`
- B. `abab`
- C. The loop completes with no output.
- D. This is an infinite loop.

29. What describes the order in which the three expressions appear in a `for` loop?

- A. boolean conditional, initialization expression, update statement
- B. initialization expression, boolean conditional, update statement
- C. initialization expression, update statement, boolean conditional
- D. None of the above

30. What is the result of the following?

```
int count = 10;
List<Character> chars = new ArrayList<>();
do {
```

```
chars.add('a');
for (Character x : chars) count -=1;
} while (count > 0);
System.out.println(chars.size());
```

- A.** 3
 - B.** 4
 - C.** The code does not compile.
 - D.** None of the above
- 31.** What is the result of the following?
- ```
int k = 0;
for (int i = 10; i > 0; i--) {
 while (i > 3) i -= 3;
 k += 1;
}
System.out.println(k);
```
- A.** 1
  - B.** 2
  - C.** 3
  - D.** 4
- 32.** Which of the following is equivalent to this code snippet given an array of `String` objects?
- ```
for (int i=fun.length-1; i>=0; i--)
    System.out.println(fun[i]);
```
- A.** `for (String f = fun) System.out.println(f);`
 - B.** `for (String f : fun) System.out.println(f);`
 - C.** `for (String f fun) System.out.println(it);`
 - D.** None of the above
- 33.** What does the following code output?

```
public static void main(String[] args) {
    List<String> bottles = Arrays.asList("glass", "plastic");
    for (int type = 0; type < bottles.size();)
        System.out.print(bottles.get(type) + ",");
        break;
    System.out.print("end");
}
```

- A. glass,end
- B. glass,plastic,end
- C. The code does not compile.
- D. None of the above

34. What is the result of the following?

```
String[] nycTourLoops = new String[] { "Downtown", "Uptown", "Brooklyn" };
String[] times = new String[] { "Day", "Night" };
for (int i = 0, j = 0; i < nycTourLoops.length
    && j < times.length; i++; j++)
{
    System.out.print(nycTourLoops[i] + " " + times[j] + "-");
}
```

- A. Downtown Day-
- B. Downtown Day-Uptown Night-
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

35. What is the result of the following when run with
java peregrine.TimeLoop September 3 1940?

```
package peregrine;
public class TimeLoop {
    public static void main(String[] args) {
        for (int i = args.length; i>=0; i--)
            System.out.println(args[i]);
    }
}
```

- A. September
- B. 1940
- C. The code does not compile.
- D. None of the above

36. What is the output of the following?

```
public class Shoelaces {
    public static void main(String[] args) {
        String tie = null;
        while (tie == null)
            tie = "shoelace";
        System.out.print(tie);
    }
}
```

- A. null
 - B. shoelace
 - C. shoelaceshoelace
 - D. None of the above
37. The following code outputs a single letter x. What happens if you remove lines 25 and 28?
- ```
23: String race = "";
24: loop:
25: do {
26: race += "x";
27: break loop;
28: } while (true);
29: System.out.println(race);
```
- A. It prints an empty string.
  - B. It still outputs a single letter x.
  - C. It no longer compiles.
  - D. It becomes an infinite loop.
38. What is the output of the following code?
- ```
package chicago;
public class Loop {
    private static int count;
    private static String[] stops = new String[] { "Washington",
        "Monroe", "Jackson", "LaSalle" };
    public static void main(String[] args) {
        while (count < stops.length) {
            if (stops[count++].length() < 8) {
                continue;
            }
        }
        System.out.println(count);
    }
}
```
- A. 1
 - B. 2
 - C. 4
 - D. The code does not compile.

39. What is the output of the following?

```
StringBuilder builder = new StringBuilder();
String str = new String("Leaves growing");
do {
    System.out.println(str);
} while (builder);
System.out.println(builder);
```

- A.** Leaves growing
- B.** This is an infinite loop.
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

40. What is the result of the following code?

```
6: int count = 0;
7: do {
8:     do {
9:         count++;
10:    } while (count < 2);
11:    break;
12: } while (true);
13: System.out.println(count);
```

- A.** 2
- B.** 3
- C.** The code does not compile.
- D.** This is an infinite loop.

41. Fill in the blank so this code compiles and does not cause an infinite loop.

```
t: while (true) {
    f: while(true) {
        _____
    }
}
```

- A.** break;
- B.** break f;
- C.** break t;
- D.** None of the above

42. What is the result of the following?

```
String[] nycTourLoops = new String[] { "Downtown", "Uptown", "Brooklyn" };
String[] times = new String[] { "Day", "Night" };
for (int i = 0, j = 0; i < nycTourLoops.length
    && j < times.length; i++, j++)
{
    System.out.print(nycTourLoops[i] + " " + times[j] + "-");
}
```

- A.** Downtown Day-
- B.** Downtown Day-Uptown Night-
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

43. How many lines does the following code output?

```
import java.util.*;
public class Exams {
    public static void main(String[] args) {
        List<String> exams = Arrays.asList("OCA", "OCP");
        for (String e1 : exams)
            for (String e2 : exams)
                System.out.println(e1 + " " + e2);
    }
}
```

- A.** One
- B.** Four
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

44. Which of the following best describes the flow of execution in this `for` loop if beta always returns false?

```
for (alpha; beta; gamma) {
    delta;
}
```

- A.** alpha
- B.** alpha, beta
- C.** alpha, beta, gamma
- D.** None of the above

- 45.** Which of the following best describes the flow of execution in this `for` loop if the loop body is run exactly once?

```
for (alpha; beta; gamma) {  
    delta;  
}
```

- A.** alpha, delta, gamma, beta
- B.** alpha, beta, delta, gamma, beta
- C.** alpha, delta, gamma, alpha, beta
- D.** alpha, beta, delta, gamma, alpha, beta

- 46.** Which of the following iterates a different number of times than the others?

- A.** `for (int k=0; k < 5; k++) {}`
- B.** `for (int k=1; k <= 5; k++) {}`
- C.** `int k=0; do {} while(k++ < 5)`
- D.** `int k=0; while (k++ < 5) {}`

- 47.** What is the output of the following?

```
public class Shoelaces {  
    public static void main(String[] args) {  
        String tie = null;  
        while (tie == null);  
        tie = "shoelace";  
        System.out.print(tie);  
    }  
}
```

- A.** null
- B.** shoelace
- C.** shoelaceshoelace
- D.** None of the above

- 48.** What is the output of the following?

```
12: int result = 8;  
13: for: while (result > 7) {  
14:     result++;  
15:     do {  
16:         result--;  
17:     } while (result > 5);  
18:     break for;  
19: }  
20: System.out.println(result);
```

- A. 5
- B. 8
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

49. What is the output of the following?

```
boolean balloonInflated = false;
do {
    if (!balloonInflated) {
        balloonInflated = true;
        System.out.print("inflate-");
    }
} while (balloonInflated);
System.out.println("done");
```

- A. done
- B. inflate-done
- C. The code does not compile.
- D. This is an infinite loop.

50. Which of the following can fill in the blank to have the code compile successfully?

```
package nyc;
public class TouristBus {
    public static void main(String... args) {
        String[] nycTourLoops = new String[] { "Downtown", "Uptown", "Brooklyn" };
        String[] times = new String[] { "Day", "Night" };
        for (_____ i < 1; i++, j++)
            System.out.println(nycTourLoops[i] + " " + times[j]);
    }
}
A. int i=0; j=0;
B. int i=0, j=0;
C. int i=0; int j=0;
D. int i=0, int j=0;
```


Chapter 6



Working with Methods and Encapsulation

**THE OCA EXAM TOPICS COVERED IN
THIS PRACTICE TEST INCLUDE THE
FOLLOWING:**

✓ **Working with Methods and Encapsulation**

- Create methods with arguments and return values; including overloaded methods
- Apply the static keyword to methods and fields
- Create and overload constructors; differentiate between default and user defined constructors
- Apply access modifiers
- Apply encapsulation principles to a class
- Determine the effect upon object references and primitive values when they are passed into methods that change the values

1. Fill in the blanks: The _____ access modifier allows access to everything the _____ access modifier does and more.
 - A. package-private, protected
 - B. protected, public
 - C. protected, package-private
 - D. private, package-private
2. What is the command to call one constructor from another constructor in the same class?
 - A. super()
 - B. this()
 - C. that()
 - D. construct()
3. What is the output of the following application?

```
package stocks;
public class Bond {
    private static int price = 5;
    public boolean sell() {
        if(price<10) {
            price++;
            return true;
        } else if(price>=10) {
            return false;
        }
    }
    public static void main(String[] cash) {
        new Bond().sell();
        new Bond().sell();
        new Bond().sell();
        System.out.print(price);
    }
}
```

- A. 5
- B. 6
- C. 8
- D. The code does not compile.

4. What is true about the following program?

```
package figures;
public class Dolls {
    public void nested() { nested(2,true); } // g1
    public int nested(int level, boolean height) { return nested(level); }
    public int nested(int level) { return level+1; }; // g2

    public static void main(String[] outOfTheBox) {
        System.out.print(new Dolls().nested());
    }
}
```

- A.** It compiles successfully and prints 3 at runtime.
 - B.** It does not compile because of line g1.
 - C.** It does not compile because of line g2.
 - D.** It does not compile for some other reason.
5. Fill in the blank: Java uses _____ to send data into a method.
- A.** pass-by-null
 - B.** pass-by-value
 - C.** both pass-by-value and pass-by-reference
 - D.** pass-by-reference
6. Which of the following is a valid JavaBean method signature?
- A.** public void getArrow()
 - B.** public void setBow()
 - C.** public void setRange(int range)
 - D.** public String addTarget(String target)
7. Which of the following statements about calling `this()` in a constructor is not true?
- A.** If `this()` is used, it must be the first line of the constructor.
 - B.** If `super()` and `this()` are both used in the same constructor, `super()` must appear on the line immediately after `this()`.
 - C.** If arguments are provided to `this()`, then there must be a constructor in the class able to take those arguments.
 - D.** If the no-argument `this()` is called, then the class must explicitly implement the no-argument constructor.

8. Which of the following can fill in the blank to make the class compile?

```
package ai;
public class Robot {
    _____ compute() { return 10; }
}
```

- A. Public int
- B. Long
- C. void
- D. private String

9. Fill in the blank: A _____ variable is always available to all instances of the class.

- A. public
- B. local
- C. static
- D. instance

10. Which line of code, inserted at line p1, causes the application to print 5?

```
package games;
public class Jump {
    private int rope = 1;
    protected boolean outside;
    public Jump() {
        // p1
        outside = true;
    }
    public Jump(int rope) {
        this.rope = outside ? rope : rope+1;
    }
    public static void main(String[] bounce) {
        System.out.print(new Jump().rope);
    }
}
```

- A. this(4);
- B. new Jump(4);
- C. this(5);
- D. rope = 4;

11. Which of the following statements is not true?
- A. An instance of one class may access an instance of another class's attributes if it has a reference to the instance and the attributes are declared `public`.
 - B. An instance of one class may access package-private attributes in a parent class, provided the parent class is not in the same package.
 - C. Two instances of the same class may access each other's `private` attributes.
 - D. An instance of one class may access an instance of another class's attributes if both classes are located in the same package and marked `protected`.

12. Given the following class, what should be inserted into the two blanks to ensure the class data is properly encapsulated?

```
package storage;
public class Box {
    public String stuff;
    _____ String _____() {
        return stuff;
    }

    public void setStuff(String stuff) {
        this.stuff = stuff;
    }
}
```

- A. `public` and `getStuff`
- B. `private` and `isStuff`
- C. `public` and `setStuff`
- D. None of the above

13. Which statement about a no-argument constructor is true?

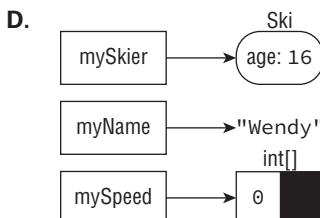
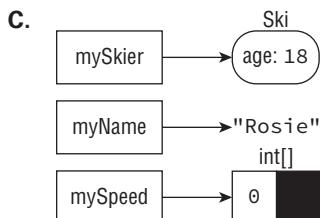
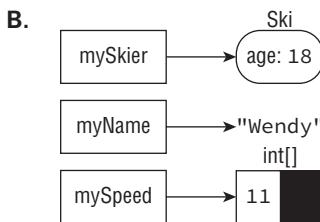
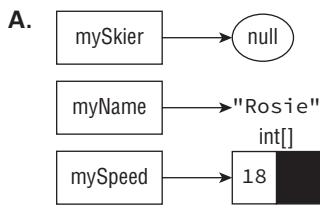
- A. The Java compiler will always insert a default no-argument constructor if you do not define a no-argument constructor in your class.
- B. In order for a class to call `super()` in one of its constructors, its parent class must explicitly implement a no-argument constructor.
- C. If a class extends another class that has only one constructor that takes a value, then the child class must explicitly declare at least one constructor.
- D. A class may contain more than one no-argument constructor.

14. Which of the following method signatures does not contain a compiler error?

- A. `public void sing(String key, String... harmonies)`
- B. `public void sing(int note, String... sound, int music)`
- C. `public void sing(String... keys, String... pitches)`
- D. `public void sing(String... notes, String melodies)`

15. Given the following application, which diagram best represents the state of the `mySkier`, `mySpeed`, and `myName` variables in the `main()` method after the call to the `slalom()` method?

```
package slopes;
public class Ski {
    private int age = 18;
    private static void slalom(Ski racer, int[] speed, String name) {
        racer.age = 18;
        name = "Wendy";
        speed = new int[1];
        speed[0] = 11;
        racer = null;
    }
    public static void main(String... mountain) {
        final Ski mySkier = new Ski();
        mySkier.age = 16;
        final int[] mySpeed = new int[1];
        final String myName = "Rosie";
        slalom(mySkier, mySpeed, myName);
    }
}
```



16. Given the class below, which method signature could be successfully added to the class as an overloaded version of the `findAverage()` method?

```
public class Calculations {  
    public Integer findAverage(int sum) { return sum; }  
}
```

- A. `public Long findAverage(int sum)`
 - B. `public Long findAverage(int sum, int divisor)`
 - C. `public Integer average(int sum)`
 - D. `private void findAverage(int sum)`
17. Which of the following is not a reason to use encapsulation when designing a class?
- A. Promote usability by other developers.
 - B. Maintain class data integrity of data elements.
 - C. Prevent users from modifying the internal attributes of a class.
 - D. Increase concurrency and improve performance.
18. Which of the following data types can be modified after they are passed to a method as an argument?
- A. `int[]`
 - B. `String`
 - C. `long`
 - D. `boolean`
19. What is the best way to call the following method from another class in the same package, assuming the class using the method does not have any `static` imports?

```
package useful;  
public class MathHelper {  
    public static int roundValue(double d) {  
        // Implementation omitted  
    }  
}
```

- A. `MathHelper:roundValue(5.92)`
- B. `MathHelper.roundValue(3.1)`
- C. `roundValue(4.1)`
- D. `useful.MathHelper.roundValue(65.3)`

- 20.** Given a method with one of the following return types, which data type prevents the return statement from being used within the method?
- A.** byte
 - B.** String
 - C.** void
 - D.** None of the above
- 21.** How many `final` modifiers would need to be removed for this application to compile?
- ```
package end;
public final class Games {
 public final static int finish(final int score) {
 final int win = 3;
 final int result = score++ < 5 ? 2 : win;
 return result+=win;
 }
 public static void main(final String[] v) {
 System.out.print(finish(Integer.parseInt(v[0])));
 }
}
```
- A.** None
  - B.** One
  - C.** Two
  - D.** The code will not compile regardless of the number of `final` modifiers that are removed.
- 22.** Fill in the blanks: \_\_\_\_\_ is used to call a constructor in the parent class, while \_\_\_\_\_ is used to reference a member of the parent class.
- A.** `super` and `this()`
  - B.** `super` and `super()`
  - C.** `super()` and `this`
  - D.** `super()` and `super`
- 23.** Given the following method signature, which classes can call it?
- ```
void run(String government)
```
- A.** Classes in other packages
 - B.** Classes in the same package
 - C.** Subclasses in a different package
 - D.** All classes

- 24.** Which statement(s) about the following class would help to properly encapsulate the data in the class?

```
package shield;
public class Protect {
    private String material;
    protected int strength;

    public int getStrength() {
        return strength;
    }
    public void setStrength(int strength) {
        this.strength = strength;
    }
}
```

- I.** Change the access modifier of `strength` to `private`.
 - II.** Add a getter method for `material`.
 - III.** Add a setter method for `material`.
- A.** I
- B.** II and III
- C.** I, II, and III
- D.** None, the data in the class is already encapsulated.
- 25.** Which of the following is a valid method name in Java?
- A.** `Go_$Outside$2()`
- B.** `have-Fun()`
- C.** `new()`
- D.** `9enjoyTheWeather()`
- 26.** Which of the following lines of code can be inserted in the line below that would allow the class to compile?

```
package farm;
public class Coop {
    public final int static getNumberOfChickens() {
        // INSERT CODE HERE
    }
}
```

- A. `return 3.0;`
 - B. `return 5L;`
 - C. `return 10;`
 - D. None of the above
27. Which of the following is a true statement about passing data to a method?
- A. A change made to a primitive value passed to a method is reflected in the calling method.
 - B. A change made to the data within an object passed to a method is reflected in the calling method.
 - C. Reassigning an object reference passed to a method is reflected in the calling method.
 - D. A change made to a boolean value passed to a method is reflected in the calling method.
28. What is a possible output of the following application?

```
package wrap;
public class Gift {
    private final Object contents;
    protected Object getContents() {
        return contents;
    }
    protected void setContents(Object contents) {
        this.contents = contents;
    }
    public void showPresent() {
        System.out.print("Your gift: "+contents);
    }
    public static void main(String[] treats) {
        Gift gift = new Gift();
        gift.setContents(gift);
        gift.showPresent();
    }
}
```

- A. `Your gift: wrap.Gift@29ca2745`
- B. `Your gift: Your gift:`
- C. It does not compile.
- D. It compiles but throws an exception at runtime.

- 29.** Which of the following is a valid JavaBean method prefix?
- A.** is
 - B.** gimme
 - C.** request
 - D.** put
- 30.** Given the following two classes, each in a different package, which line inserted below allows the second class to compile?

```
package clothes;
public class Store {
    public static String getClothes() { return "dress"; }
}
```

```
package wardrobe;
// INSERT CODE HERE
public class Closet {
    public void borrow() {
        System.out.print("Borrowing clothes: "+getClothes());
    }
}
```

- A.** static import clothes.Store.getClothes;
 - B.** import clothes.Store.*;
 - C.** import static clothes.Store.getClothes;
 - D.** import static clothes.Store;
- 31.** What access modifier is used to mark class members package-private?
- A.** private
 - B.** default
 - C.** protected
 - D.** None of the above
- 32.** How many lines of the following program contain compilation errors?

```
package sky;
public class Stars {
    private int inThe = 4;
    public void Stars() {
        super();
    }
}
```

```
public Stars(int inThe) {  
    this.inThe = this.inThe;  
}  
public static void main(String[] endless) {  
    System.out.print(new sky.Stars(2).inThe);  
}  
}
```

- A.** None
 - B.** One
 - C.** Two
 - D.** Three
- 33.** Which of the following statements is true?
- A.** An instance method is allowed to reference a `static` variable.
 - B.** A `static` method is allowed to reference an instance variable.
 - C.** A `static` initialization block is allowed to reference an instance variable.
 - D.** A `final static` variable may be set in a constructor.
- 34.** Given the following method declaration, which line can be inserted to make the code compile?
- ```
public short calculateDistance(double lat1, double lon1,
 double lat2, double lon2) {
 // INSERT CODE HERE
}
```
- A.** `return new Integer(3);`
  - B.** `return new Byte((byte)6);`
  - C.** `return 5L;`
  - D.** `return new Short(4).longValue();`
- 35.** Which of the following statements about overloaded methods are true?
- I.** Overloaded methods must have the same name.
  - II.** Overloaded methods must have the same return type.
  - III.** Overloaded methods must have a different list of parameters.
- A.** I
  - B.** I and II
  - C.** I and III
  - D.** I, II, and III

36. How many lines of code would need to be removed for the following class to compile?

```
package work;
public class Week {
 private static final String monday;
 String tuesday;
 final static wednesday = 3;
 final protected int thursday = 4;
}
```

- A. One
  - B. Two
  - C. Three
  - D. The code will not compile regardless of the number of lines removed.
37. What is the output of the following application?

```
package pet;
public class Puppy {
 public static int wag = 5; // q1
 public void Puppy(int wag) { // q2
 this.wag = wag;
 }
 public static void main(String[] tail) {
 System.out.print(new Puppy(2).wag); // q3
 }
}
```

- A. 2
  - B. It does not compile because of line q1.
  - C. It does not compile because of line q2.
  - D. It does not compile because of line q3.
38. Fill in the blanks: The \_\_\_\_\_ access modifier allows access to everything the \_\_\_\_\_ access modifier does and more.
- A. public, private
  - B. private, package-private
  - C. package-private, protected
  - D. private, public

39. What is the output of the following application?

```
package ship;
public class Phone {
 private int size;
 public Phone(int size) {this.size=size;

 public static void sendHome(Phone p, int newSize) {
 p = new Phone(newSize);
 p.size = 4;
 }
 public static final void main(String... params) {
 final Phone phone = new Phone(3);
 sendHome(phone,7);
 System.out.print(phone.size);
 }
}
```

- A.** 3
- B.** 4
- C.** 7
- D.** The code does not compile.

40. Given the following class, which line of code when inserted below would prevent the class from compiling?

```
public class Drink {
 public static void water() {}
 public void get() {
 // INSERT CODE HERE
 }
}
```

- A.** water();
- B.** this.Drink.water();
- C.** this.water();
- D.** Drink.water();

41. Given the following method declaration signature, which of the following is a valid call of this method?

```
public void call(int count, String me, String... data)
```

- A.** call(9,"me",10,"Al")
- B.** call(5)

- C. `call(2,"home","sweet")`
  - D. `call("answering","service")`
- 42.** Which statement about a `static` variable is true?
- A. The value of a `static` variable must be set when the variable is declared or in a `static` initialization block.
  - B. It is not possible to read `static final` variables outside the class in which they are defined.
  - C. It is not possible to reference `static` methods using `static` imports.
  - D. A `static` variable is always available in all instances of the class.
- 43.** Which of the following is not a true statement?
- A. The first line of every constructor is a call to the parent constructor via the `super()` command.
  - B. A class does not have to have a constructor explicitly defined.
  - C. A constructor may pass arguments to the parent constructor.
  - D. A `final` instance variable whose value is not set when they are declared or in an initialization block should be set by the constructor.
- 44.** How many `final` modifiers would need to be removed for this application to compile?

```
package park;
public class Tree {
 public final static long numberoftrees;
 public final double height;
 static {}
 { final int initheight = 2;
 height = initheight;
 }
 static {
 numberoftrees = 100;
 height = 4;
 }
}
```

- A. None
- B. One
- C. Two
- D. The code will not compile regardless of the number of `final` modifiers removed.

**45.** What is the output of the following application?

```
package jungle;
public class RainForest extends Forest {
 public RainForest(long treeCount) {
 this.treeCount = treeCount+1;
 }
 public static void main(String[] birds) {
 System.out.print(new RainForest(5).treeCount);
 }
}
class Forest {
 public long treeCount;
 public Forest(long treeCount) {
 this.treeCount = treeCount+2;
 }
}
```

- A.** 5
- B.** 6
- C.** 8
- D.** The code does not compile.

**46.** What is the output of the following application?

```
public class ChooseWisely {
 public ChooseWisely() { super(); }
 public int choose(int choice) { return 5; }
 public int choose(short choice) { return 2; }
 public int choose(long choice) { return 11; }
 public static void main(String[] path) {
 System.out.print(new ChooseWisely().choose((byte)2+1));
 }
}
```

- A.** 5
- B.** 2
- C.** 11
- D.** The code does not compile.

47. What is the output of the following application?

```
package sports;
public class Football {
 public static Long getScore(Long timeRemaining) {
 return 2*timeRemaining; // m1
 }
 public static void main(String[] refs) {
 final int startTime = 4;
 System.out.print(getScore(startTime)); // m2
 }
}
```

- A. 8
  - B. The code does not compile because of line m1.
  - C. The code does not compile because of line m2.
  - D. The code compiles but throws an exception at runtime.
48. Which of the following is a valid method name in Java?
- A. \$sprint()
  - B. \jog13()
  - C. walk#()
  - D. %run()
49. Assume there is a class `Bouncer` with a protected variable. Methods in which class can access this variable?
- A. Only subclasses of `Bouncer`
  - B. Any subclass of `Bouncer` or any class in the same package as `Bouncer`
  - C. Only classes in the same package as `Bouncer`
  - D. Any superclass of `Bouncer`
50. Given the following two classes, each in a different package, which line inserted below allows the second class to compile?

```
package commerce;
public class Bank {
 public void withdrawal(int amountInCents) {}
 public void deposit(int amountInCents) {}
}

package employee;
// INSERT CODE HERE
public class Teller {
 public void processAccount(int depositSlip, int withdrawalSlip) {
```

```
 withdrawal(withdrawalSlip);
 deposit(depositSlip);
}
}
```

- A.** import static commerce.Bank.\*;
- B.** static import commerce.Bank.\*;
- C.** import static commerce.Bank;
- D.** None of the above

# Chapter 7

# Working with Inheritance

---

**THE OCA EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Working with Inheritance**

- Describe inheritance and its benefits
- Develop code that makes use of polymorphism; develop code that overrides methods; differentiate between the type of a reference and the type of an object
- Determine when casting is necessary
- Use super and this to access objects and constructors
- Use abstract classes and interfaces

1. How many lines of the following program contain compilation errors?

```
package theater;
class Cinema {
 private String name;
 public Cinema(String name) {this.name = name;}
}
public class Movie extends Cinema {
 public Movie(String movie) {}
 public static void main(String[] showing) {
 System.out.print(new Movie("Another Trilogy").name);
 }
}
```

- A. None
  - B. One
  - C. Two
  - D. Three
2. Which modifier can be applied to an abstract interface method?
- A. `protected`
  - B. `static`
  - C. `final`
  - D. `public`
3. What is the output of the following application?

```
package radio;
public class Song {
 public void playMusic() {
 System.out.print("Play!");
 }
 private static int playMusic() {
 System.out.print("Music!");
 }
 public static void main(String[] tracks) {
 new Song().playMusic();
 }
}
```

- A. Play!
- B. Music!
- C. The code does not compile.
- D. The code compiles but the answer cannot be determined until runtime.

4. Which of the following statements about inheritance is true?
- A. Inheritance allows objects to access commonly used attributes and methods.
  - B. Inheritance always leads to simpler code.
  - C. All primitives and objects inherit a set of methods.
  - D. Inheritance allows you to write methods that reference themselves.
5. Given the class declaration below, which value cannot be inserted into the blank line that would allow the code to compile?
- ```
package mammal;
interface Pet {}
public class Canine implements Pet {
    public _____ getDoggy() {
        return this;
    }
}
```
- A. Class
 - B. Pet
 - C. Canine
 - D. Object
6. Imagine you are working with another team to build an application. You are developing code that uses a class that the other team has not finished writing yet. Which element of Java would best facilitate this development, allowing easy integration once the other team's code is complete?
- A. An abstract class
 - B. An interface
 - C. static methods
 - D. An access modifier
7. What is the output of the following application?

```
package vehicles;
class Automobile {
    private final String drive() { return "Driving vehicle"; }
}
class Car extends Automobile {
    protected String drive() { return "Driving car"; }
}
public class ElectricCar extends Car {
    public final String drive() { return "Driving electric car"; }
    public static void main(String[] wheels) {
        final Car car = new ElectricCar();
```

```
        System.out.print(car.drive());
    }
}
```

- A.** Driving vehicle
 - B.** Driving electric car
 - C.** Driving car
 - D.** The code does not compile.
- 8.** Which of the following statements about inheritance is correct?
- A.** Java does not support multiple inheritance.
 - B.** Java allows multiple inheritance using abstract classes.
 - C.** Java allows multiple inheritance using non-abstract classes.
 - D.** Java allows multiple inheritance using interfaces.
- 9.** How many changes need to be made to the classes below to properly override the `watch()` method?
- ```
package entertainment;
class Television {
 protected final void watch() {}
}
public class LCD extends Television {
 Object watch() {}
}
```
- A.** One
  - B.** Two
  - C.** Three
  - D.** None; the code compiles as is.
- 10.** Which of the following statements about overriding a method is incorrect?
- A.** The return types must be covariant.
  - B.** The access modifier of the method in the child class must be the same or broader than the method in the superclass.
  - C.** A checked exception thrown by the method in the parent class must be thrown by the method in the child class.
  - D.** A checked exception thrown by a method in the child class must be the same or narrower than the exception thrown by the method in the parent class.
- 11.** What is the output of the following application?

```
package machines;
class Computer {
 protected final int process() { return 5; }
```

```
 }
 public class Laptop extends Computer {
 public final int process() { return 3; }
 public static void main(String[] chips) {
 System.out.print(new Laptop().process());
 }
 }
```

- A.** 5
  - B.** 3
  - C.** The code does not compile.
  - D.** The code compiles but throws an exception at runtime.
- 12.** Given that `FileNotFoundException` is a subclass of `IOException`, what is the output of the following application?

```
package edu;
import java.io.*;
class School {
 public int getNumberOfStudentsPerClassroom(String... students)
 throws IOException {
 return 3;
 }
 public int getNumberOfStudentsPerClassroom() throws IOException {
 return 9;
 }
}
public class HighSchool extends School {
 public int getNumberOfStudentsPerClassroom() throws FileNotFoundException {
 return 2;
 }
 public static void main(String[] students) throws IOException {
 School school = new HighSchool();
 System.out.print(school.getNumberOfStudentsPerClassroom());
 }
}
A. 2
B. 3
C. 9
D. The code does not compile.
```

**13.** Which modifier can be applied to an interface method?

- A.** protected
- B.** static
- C.** private
- D.** final

**14.** What is the output of the following application?

```
package track;
interface Run {
 default void walk() {
 System.out.print("Walking and running!");
 }
}
interface Jog {
 default void walk() {
 System.out.print("Walking and jogging!");
 }
}
public class Sprint implements Run, Jog {
 public void walk() {
 System.out.print("Sprinting!");
 }
 public static void main() {
 new Sprint().walk();
 }
}
```

- A.** Walking and running!
- B.** Walking and jogging!
- C.** Sprinting!
- D.** The code does not compile.

**15.** Which of the following statements about interfaces is not true?

- A.** An interface can extend another interface.
- B.** An interface can implement another interface.
- C.** A class can implement two interfaces.
- D.** A class can extend another class.

16. What is the output of the following application?

```
package transport;

class Ship {
 protected int weight = 3;
 private int height = 5;
 public int getWeight() { return weight; }
 public int getHeight() { return height; }
}

public class Rocket extends Ship {
 public int weight = 2;
 public int height = 4;
 public void printDetails() {
 System.out.print(super.getWeight() + ", " + super.height);
 }
 public static final void main(String[] fuel) {
 new Rocket().printDetails();
 }
}

A. 2,5
B. 3,4
C. 3,5
D. The code does not compile.
```

17. Fill in the blanks: Excluding `default` and `static` methods, a(n) \_\_\_\_\_ can contain both abstract and concrete methods, while a(n) \_\_\_\_\_ contains only abstract methods.

- A. concrete class, abstract class
- B. concrete class, interface
- C. interface, abstract class
- D. abstract class, interface

18. Which statement about the following class is correct?

```
package shapes;
abstract class Triangle {
 abstract String getDescription();
}
```

```
class RightTriangle extends Triangle {
 protected String getDescription() { return "rt"; } // g1
}
public abstract class IsoscelesRightTriangle extends RightTriangle { // g2
 public String getDescription() { return "irt"; }
 public static void main(String[] edges) {
 final Triangle shape = new IsoscelesRightTriangle(); // g3
 System.out.print(shape.getDescription());
 }
}
```

- A.** The code does not compile due to line g1.
  - B.** The code does not compile due to line g2.
  - C.** The code does not compile due to line g3.
  - D.** The code compiles and runs without issue.
- 19.** Given that `Short` and `Integer` extend `Number`, what type can be used to fill in the blank in the class below to allow it to compile?

```
package band;

interface Horn { public Integer play(); }
abstract class Woodwind { public Short play() {return 3;} }
public final class Saxophone extends Woodwind implements Horn {
 public _____ play() {
 return null;
 }
}
```

- A.** `Integer`
  - B.** `Short`
  - C.** `Number`
  - D.** None of the above
- 20.** Fill in the blanks: A class \_\_\_\_\_ an interface, while a class \_\_\_\_\_ an abstract class.
- A.** extends, implements
  - B.** extends, extends
  - C.** implements, extends
  - D.** implements, implements

21. What is the output of the following application?

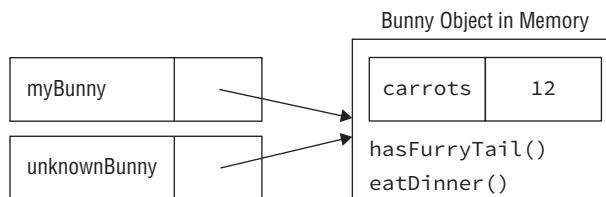
```
package paper;

abstract class Book {
 protected static String material = "papyrus";
 public Book() {}
 public Book(String material) {this.material = material;}
}

public class Encyclopedia extends Book {
 public static String material = "cellulose";
 public Encyclopedia() {super();}
 public String getMaterial() {return super.material;}
 public static void main(String[] pages) {
 System.out.print(new Encyclopedia().getMaterial());
 }
}
```

- A. papyrus
- B. cellulose
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

22. The following diagram shows two reference variables pointing to the same Bunny object in memory. The reference variable `myBunny` is of type `Bunny`, while `unknownBunny` is of an unknown data type. Which statement about the reference variables is not true? For this question, assume the instance methods and variables shown in the diagram are marked `public`.



- A. If the `unknownBunny` reference does not have access to the same variables and methods that `myBunny` has access to, it can be explicitly cast to a reference type that does.
- B. The data type of `unknownBunny` must be `Bunny` or a subclass of `Bunny`.
- C. If the data type of `unknownBunny` is `Bunny`, it has access to all of the same methods and variables as `myBunny`.
- D. The data type of `unknownBunny` could be an interface, class, or abstract class.

**23.** Which of the following modifiers can be applied to an abstract method?

- A.** final
- B.** private
- C.** default
- D.** protected

**24.** What is the output of the following application?

```
package space;

interface Sphere {
 default String getName() { return "Unknown"; }
}
abstract class Planet {
 abstract String getName();
}
public class Mars extends Sphere implements Planet {
 public Mars() {
 super();
 }
 public String getName() { return "Mars"; }
 public static void main(final String[] probe) {
 System.out.print(((Planet)new Mars()).getName());
 }
}
```

- A.** Mars
- B.** Unknown
- C.** The code does not compile due to the declaration of Sphere.
- D.** The code does not compile for another reason.

**25.** Which of the following statements is correct?

- A.** A reference to a class can be assigned to a subclass reference without an explicit cast.
- B.** A reference to a class can be assigned to a superclass reference without an explicit cast.
- C.** A reference to an interface can be assigned to a reference of a class that implements the interface without an explicit cast.
- D.** A reference to a class that implements an interface can be assigned to an interface reference only with an explicit cast.

**26.** Of the following four modifiers, choose the one that is not implicitly applied to all interface variables.

- A.** final
- B.** abstract
- C.** static
- D.** public

27. What is the output of the following application?

```
package race;
abstract class Car {
 static { System.out.print("1"); }
 public Car(String name) {
 super();
 System.out.print("2");
 }
 { System.out.print("3"); }
}
public class BlueCar extends Car {
 { System.out.print("4"); }
 public BlueCar() {
 super("blue");
 System.out.print("5");
 }
 public static void main(String[] gears) {
 new BlueCar();
 }
}
A. 23451
B. 12354
C. 13245
D. The code does not compile.
```

28. Fill in the blank: Overloaded and overridden methods always have \_\_\_\_\_.

- A. the same parameter list
- B. different return types
- C. the same method name
- D. covariant return types

29. What is the output of the following application?

```
package sports;
abstract class Ball {
 protected final int size;
 public Ball(int size) {
 this.size = size;
 }
}
```

```
interface Equipment {}
public class SoccerBall extends Ball implements Equipment {
 public SoccerBall() {
 super(5);
 }
 public Ball get() { return this; }
 public static void main(String[] passes) {
 Equipment equipment = (Equipment)(Ball)new SoccerBall().get();
 System.out.print(((SoccerBall)equipment).size());
 }
}
```

- A. 5  
B. The code does not compile due an invalid cast.  
C. The code does not compile for a different reason.  
D. The code compiles but throws a `ClassCastException` at runtime.

30. Fill in the blanks: A class that defines an instance variable with the same name as a variable in the parent class is referred to as \_\_\_\_\_ a variable, while a class that defines a `static` method with the same signature as a `static` method in a parent class is referred to as \_\_\_\_\_ a method.

- A. hiding, overriding  
B. overriding, hiding  
C. hiding, hiding  
D. replacing, overriding

31. Which statement about the following class is correct?

```
package shapes;

abstract class Parallelogram {
 private int getEqualSides() {return 0;}
}
abstract class Rectangle extends Parallelogram {
 public static int getEqualSides() {return 2;} // x1
}
public final class Square extends Rectangle {
 public int getEqualSides() {return 4;} // x2
 public static void main(String[] corners) {
 final Square myFigure = new Square(); // x3
 System.out.print(myFigure.getEqualSides());
 }
}
```

- A. The code does not compile due to line x1.
  - B. The code does not compile due to line x2.
  - C. The code does not compile due to line x3.
  - D. The code compiles and runs without issue.
32. What is the output of the following application?

```
package flying;

class Rotorcraft {
 protected final int height = 5;
 abstract int fly();
}

public class Helicopter extends Rotorcraft {
 private int height = 10;
 protected int fly() {
 return super.height;
 }
 public static void main(String[] unused) {
 Helicopter h = (Helicopter) new Rotorcraft();
 System.out.print(h.fly());
 }
}
```

- A. 5
  - B. 10
  - C. The code does not compile.
  - D. The code compiles but produces a `ClassCastException` at runtime.
33. Fill in the blanks: A class may be assigned to a(n) \_\_\_\_\_ reference variable automatically but requires an explicit cast when assigned to a(n) \_\_\_\_\_ reference variable.
- A. subclass, outer class
  - B. superclass, subclass
  - C. subclass, superclass
  - D. abstract class, concrete class
34. Fill in the blank: A(n) \_\_\_\_\_ is the first non-abstract subclass that is required to implement all of the inherited abstract methods.
- A. abstract class
  - B. abstraction
  - C. concrete class
  - D. interface

**35.** How many compiler errors does the following code contain?

```
package animal;
interface CanFly {
 public void fly() {}
}
final class Bird {
 public int fly(int speed) {}
}
public class Eagle extends Bird implements CanFly {
 public void fly() {}
}
```

- A.** None
- B.** One
- C.** Two
- D.** Three

**36.** Which of the following is not an attribute common to both abstract classes and interfaces?

- A.** They both can contain `static` variables.
- B.** They both can contain `default` methods.
- C.** They both can contain `static` methods.
- D.** They both can contain `abstract` methods.

**37.** What is the output of the following application?

```
package musical;
interface SpeakDialogue { default int talk() { return 7; } }
interface SingMonologue { default int talk() { return 5; } }
public class Performance implements SpeakDialogue, SingMonologue {
 public int talk(String... x) {
 return x.length;
 }
 public static void main(String[] notes) {
 System.out.print(new Performance().talk(notes));
 }
}
```

- A.** 7
- B.** 5
- C.** The code does not compile.
- D.** The code compiles without issue, but the output cannot be determined until runtime.

- 38.** Which of the following is a virtual method?
- A.** protected instance methods
  - B.** static methods
  - C.** private instance methods
  - D.** final instance methods
- 39.** Fill in the blanks: An interface \_\_\_\_\_ another interface, while a class \_\_\_\_\_ another class.
- A.** implements, extends
  - B.** extends, extends
  - C.** implements, implements
  - D.** extends, implements
- 40.** What is the output of the following application?
- ```
class Math {  
    public final double secret = 2;  
}  
class ComplexMath extends Math {  
    public final double secret = 4;  
}  
public class InfiniteMath extends ComplexMath {  
    public final double secret = 8;  
    public static void main(String[] numbers) {  
        Math math = new InfiniteMath();  
        System.out.print(math.secret);  
    }  
}
```
- A.** 2
 - B.** 4
 - C.** 8
 - D.** The code does not compile.

- 41.** Given the following method and the fact that `FileNotFoundException` is a subclass of `IOException`, which of the following method signatures is a valid override by a subclass?

```
protected void dance() throws FileNotFoundException {}
```

- A.** void dance() throws IOException
- B.** public void dance() throws IOException
- C.** private void dance() throws FileNotFoundException
- D.** public final void dance()

- 42.** Given the class definitions below, which value, when inserted into the blank line, does not allow the class to compile?

```
public class Canine {}
public class Dog extends Canine {}
public class Wolf extends Canine {}
public final class Husky extends Dog {}
public class Zoologist {
    Canine animal;
    public final void setAnimal(Dog animal) { this.animal = animal; }
    public static void main(String[] furryFriends) {
        new Zoologist().setAnimal(______);
    }
}
```

- A.** new Husky()
- B.** new Dog()
- C.** new Wolf()
- D.** null

- 43.** Which of the following modifiers cannot be applied to an interface method?

- A.** final
- B.** default
- C.** static
- D.** abstract

- 44.** Which statement about the following application is true?

```
package party;

abstract class House {
    protected abstract Object getSpace();
}
abstract class Room extends House {
    abstract Object getSpace(Object list);
}
abstract public class Ballroom extends House {
    protected abstract Object getSpace();
    public static void main(String[] squareFootage) {
        System.out.print("Let's start the party!");
    }
}
```

- A. It compiles and at runtime prints Let's start the party!
 - B. It does not compile for one reason.
 - C. It does not compile for two reasons.
 - D. It does not compile for three reasons.
45. Fill in the blanks: _____ methods must have a different list of parameters, while _____ methods must have the exact same return type.
- A. Overloaded, overridden
 - B. Inherited, overridden
 - C. Overridden, overloaded
 - D. None of the above
46. Which of the following statements about no-argument constructors is correct?
- A. If a parent class does not include a no-argument constructor, a child class cannot declare one.
 - B. If a parent class does not include a no-argument constructor (nor a default one inserted by the compiler), a child class must contain at least one constructor definition.
 - C. If a parent class contains a no-argument constructor, a child class must contain a no-argument constructor.
 - D. If a parent class contains a no-argument constructor, a child class must contain at least one constructor.
47. Fill in the blanks: The _____ determines which attributes exist in memory, while the _____ determines which attributes are accessible by the caller.
- A. reference type, signature
 - B. object type, superclass
 - C. reference type, object type
 - D. object type, reference type
48. Given that Integer and Long are subclasses of Number, what type can be used to fill in the blank in the class below to allow it to compile?

```
package orchestra;
interface MusicCreator { public Number play(); }
abstract class StringInstrument { public Long play() {return 3L;} }
public class Violin extends StringInstrument implements MusicCreator {
    public _____ play() {
        return 12;
    }
}
```

- A. Long
 - B. Integer
 - C. Long or Integer
 - D. Long or Number
49. Which of the following is the best reason for creating a default interface method?
- A. Allow interface methods to be inherited.
 - B. Add backward compatibility to existing interfaces.
 - C. Give an interface the ability to create concrete methods.
 - D. Allow an interface to define a method at the class level.
50. Given that `EOFException` is a subclass of `IOException`, what is the output of the following application?

```
package ai;
import java.io.*;
class Machine {
    public boolean turnOn() throws EOFException {return true;}
}
public class Robot extends Machine {
    public boolean turnOn() throws IOException {return false;}
    public static void main(String[] doesNotCompute) throws Exception {
        Machine m = new Robot();
        System.out.print(m.turnOn());
    }
}


- A. true
- B. false
- C. The code does not compile.
- D. The code compiles but produces an exception at runtime.

```

Chapter 8



Handling Exceptions

**THE OCA EXAM TOPICS COVERED
IN THIS PRACTICE TEST INCLUDE
THE FOLLOWING:**

✓ **Handling Exceptions**

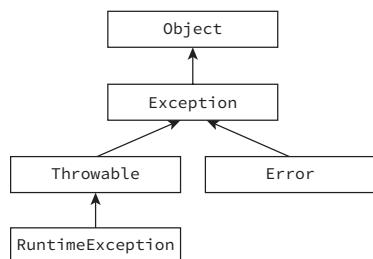
- Differentiate among checked exceptions, unchecked exceptions, and Errors
- Create a try-catch block and determine how exceptions alter normal program flow
- Describe the advantages of Exception handling
- Create and invoke a method that throws an exception
- Recognize common exception classes (such as `NullPointerException`, `ArithmaticException`, `ArrayIndexOutOfBoundsException`, `ClassCastException`)

1. What is the result of compiling and executing the following application?

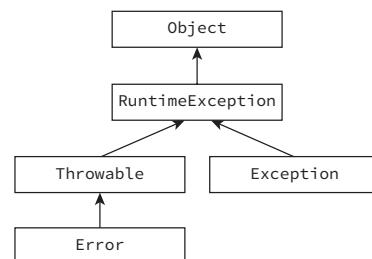
```
package mind;
public class Remember {
    public static void think() throws Exception { // k1
        try {
            throw new Exception();
        }
    }
    public static void main(String... ideas) throws Exception {
        think();
    }
}
```

- A. The code compiles and runs without printing anything.
 - B. The code compiles but a stack trace is printed at runtime.
 - C. The code does not compile because of line k1.
 - D. The code does not compile for another reason.
2. Choose the answer that lists the keywords in the order that they would be used together.
- A. catch, try, finally
 - B. try, catch, finally
 - C. finally, catch, try
 - D. try, finally, catch
3. Which of the following diagrams of `java.lang` classes shows the inheritance model properly?

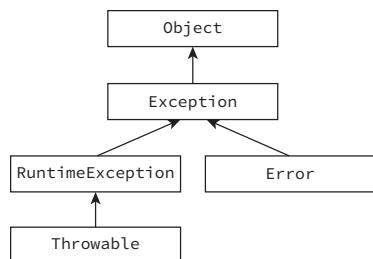
A.



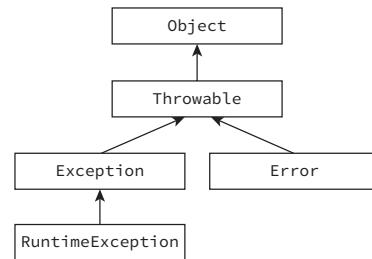
B.



C.



D.



4. Which of the following `Throwable` types is it recommended not to catch in a Java application?

- A. `Error`
- B. `CheckedException`
- C. `Exception`
- D. `RuntimeException`

5. What is the output of the following application?

```
package game;
public class Baseball {
    public static void main(String... teams) {
        try {
            int score = 1;
            System.out.print(score++);
        } catch (Throwable t) {
            System.out.print(score++);
        } finally {
            System.out.print(score++);
        }
        System.out.print(score++);
    }
}
```

- A. 123
- B. 124
- C. 12
- D. None of the above

6. Which of the following is a checked exception?

- A. `ClassCastException`
- B. `IOException`
- C. `ArrayIndexOutOfBoundsException`
- D. `IllegalArgumentException`

7. Fill in the blanks: The _____ keyword is used in method declarations, while the _____ keyword is used to throw an exception to the surrounding process.

- A. `throws, throw`
- B. `catch, throw`
- C. `throw, throws`
- D. `throws, catch`

8. If a try statement has catch blocks for both Exception and IOException, then which of the following statements is correct?
- A. The catch block for Exception must appear before the catch block for IOException.
 - B. The catch block for IOException must appear before the catch block for Exception.
 - C. The catch blocks for these two exception types can be declared in any order.
 - D. A try statement cannot be declared with these two catch block types because they are incompatible.

9. What is the output of the following application?

```
package game;
public class Football {
    public static void main(String officials[]) {
        try {
            System.out.print('A');
            throw new RuntimeException("Out of bounds!");
        } catch (ArrayIndexOutOfBoundsException aioobe) {
            System.out.print('B');
            throw t;
        } finally {
            System.out.print('C');
        }
    }
}
```

- A. ABC
- B. ABC, followed by a stack trace for a RuntimeException
- C. AC, followed by a stack trace for a RuntimeException
- D. None of the above

10. What is the result of compiling and running the following application?

```
package castles;
public class Fortress {
    public void openDrawbridge() throws Exception { // p1
        try {
            throw new Exception("Circle");
        } catch (Exception e) {
            System.out.print("Opening!");
        } finally {
            System.out.print("Walls"); // p2
        }
    }
}
```

- ```
public static void main(String[] moat) {
 new Fortress().openDrawbridge(); // p3
}
}

A. The code does not compile because of line p1.
B. The code does not compile because of line p2.
C. The code does not compile because of line p3.
D. The code compiles, but a stack trace is printed at runtime.

11. Which of the following exception types must be handled or declared by the method in which they are thrown?

A. NullPointerException
B. Exception
C. RuntimeException
D. ArithmeticException

12. What is the output of the following application?


```
package game;  
public class BasketBall {  
    public static void main(String[] dribble) {  
        try {  
            System.out.print(1);  
            throw new ClassCastException();  
        } catch (ArrayIndexOutOfBoundsException ex) {  
            System.out.print(2);  
        } catch (Throwable ex) {  
            System.out.print(3);  
        } finally {  
            System.out.print(4);  
        }  
        System.out.print(5);  
    }  
}
```


A. 1345
B. 1235
C. The code does not compile.
D. The code compiles but throws an exception at runtime.


```

- 13.** Which of the following statements about a `finally` block is true?
- A.** Every line of the `finally` block is guaranteed to be executed.
  - B.** The `finally` block is executed only if the related `catch` block is also executed.
  - C.** The `finally` statement requires brackets `{}`.
  - D.** The `finally` block cannot throw an exception.
- 14.** Given that `FileNotFoundException` is a subclass of `IOException`, what is the output of the following application?

```
package office;
import java.io.*;
public class Printer {
 public void print() {
 try {
 throw new FileNotFoundException();
 } catch (IOException exception) {
 System.out.print("Z");
 } catch (FileNotFoundException enfe) {
 System.out.print("X");
 } finally {
 System.out.print("Y");
 }
 }
 public static void main(String... ink) {
 new Printer().print();
 }
}
```

- A.** XY
  - B.** ZY
  - C.** The code does not compile.
  - D.** The code compiles but a stack trace is printed at runtime.
- 15.** Which keywords are required with a `try` statement?
- I.** `catch`
  - II.** `finalize`
  - III.** `finally`
- A.** I only
  - B.** II only
  - C.** I or III, or both
  - D.** None of these statements are required with a `try` statement.

- 16.** Which statement about the role of exceptions in Java is incorrect?
- A.** Exceptions are often used when things “go wrong” or deviate from the expected path.
  - B.** An application that throws an exception will terminate.
  - C.** Some exceptions can be avoided programmatically.
  - D.** An application that can properly handle its exception may recover from unexpected problems.

- 17.** What is the output of the following application?

```
package harbor;
class CapsizedException extends Exception {}
class Transport {
 public int travel() throws CapsizedException { return 2; }
}
public class Boat {
 public int travel() throws Exception { return 4; } // j1
 public static void main(String... distance) throws Exception{
 try {
 System.out.print(new Boat().travel());
 } catch (Exception e) {
 System.out.print(8);
 }
 }
}
```

- A.** 4
- B.** 8
- C.** The code does not compile due to line j1.
- D.** The code does not compile for another reason.

- 18.** Which of following method signatures would not be allowed in a class implementing the `Printer` interface?

```
class PrintException extends Exception {}
class PaperPrintException extends PrintException {}
public interface Printer {
 abstract int printData() throws PrintException;
}

- A. public int printData() throws PaperPrintException
- B. public int printData() throws Exception
- C. public int printData()
- D. None of the above

```

- 19.** Which import statement is required to be declared in order to use the `Exception`, `RuntimeException`, and `Throwable` classes in an application?

- A.** `import java.exception.*;`
- B.** `import java.util.exception.*;`
- C.** `import java.lang.*;`
- D.** None of the above

- 20.** Which statement about the following classes is correct?

```
class GasException extends Exception {}
class Element {
 public int getSymbol() throws GasException { return -1; } // g1
}
public class Oxygen extends Element {
 public int getSymbol() { return 8; } // g2
 public void printData() {
 try {
 System.out.print(getSymbol());
 } catch { // g3
 System.out.print("Unable to read data");
 }
 }
}
```

- A.** The code does not compile because of line g1.
- B.** The code does not compile because of line g2.
- C.** The code does not compile because of line g3.
- D.** None of the above

- 21.** Fill in the blanks: A program must handle or declare \_\_\_\_\_ but should never handle \_\_\_\_\_.

- A.** `java.lang.Error`, unchecked exceptions
- B.** checked exceptions, `java.lang.Error`
- C.** `java.lang.Throwable`, `java.lang.Error`
- D.** unchecked exceptions, `java.lang.Exception`

- 22.** What is the result of compiling and running the following application?

```
package castles;
class CastleUnderSiegeException extends Exception {}
class KnightAttackingException extends CastleUnderSiegeException {}
public class Citadel {
 public void openDrawbridge() throws RuntimeException { // q1
```

```
try {
 throw new KnightAttackingException();
} catch (Exception e) {
 throw new ClassCastException();
} finally {
 throw new CastleUnderSiegeException(); // q2
}
}
public static void main(String[] moat) {
 new Citadel().openDrawbridge(); // q3
}
}
```

- A. The code does not compile because of line q1.
  - B. The code does not compile because of line q2.
  - C. The code does not compile because of line q3.
  - D. The code compiles, but a stack trace is printed at runtime.
23. If an exception matches two or more catch blocks, which catch block is executed?
- A. The first one that matches is executed.
  - B. The last one that matches is executed.
  - C. All matched blocks are executed.
  - D. It is not possible to write code like this.
24. What is the output of the following application?

```
package system;
public class Computer {
 public void compute() throws Exception {
 throw new RuntimeException("Error processing request");
 }
 public static void main(String[] bits) {
 try {
 new Computer().compute();
 System.out.print("Ping");
 } catch (NullPointerException e) {
 System.out.print("Pong");
 throw e;
 }
 }
}
```

- A. Ping
  - B. Pong
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
25. In the following application, the value of `list` has been omitted. Assuming the code compiles without issue, which one of the following is not a possible output of executing this class?

```
package checkboard;

public class Attendance {
 private Boolean[] list = // value omitted
 public int printTodaysCount() {
 int count=0;
 for(int i=0; i<10; i++) {
 if(list[i]) ++count;
 }
 return count;
 }
 public static void main(String[] roster) {
 new Attendance().printTodaysCount();
 }
}
```

- A. A stack trace for `NullPointerException` is printed.
  - B. A stack trace for `ArrayIndexOutOfBoundsException` is printed.
  - C. A stack trace for `ClassCastException` is printed.
  - D. None of the above
26. Fill in the blanks: A \_\_\_\_\_ occurs when a program recurses too deeply into an infinite loop, while a(n) \_\_\_\_\_ occurs when a reference to a nonexistent object is acted upon.
- A. `NoClassDefFoundError`, `StackOverflowError`
  - B. `StackOverflowError`, `NullPointerException`
  - C. `ClassCastException`, `IllegalArgumentException`
  - D. `StackOverflowError`, `IllegalArgumentException`
27. Which of the following is not a reason to add checked exceptions to a method signature?
- A. To force a caller to handle or declare its exceptions
  - B. To notify the caller of potential types of problems
  - C. To ensure that exceptions never cause the application to terminate
  - D. To give the caller a chance to recover from a problem

**28.** What is the output of the following application?

```
package peculiar;
public class Stranger {
 public static String getFullName(String firstName, String lastName) {
 try {
 return firstName.toString() + " " + lastName.toString();
 } finally {
 System.out.print("Finished!");
 } catch (NullPointerException npe) {
 System.out.print("Problem?");
 }
 return null;
 }
 public static void main(String[] things) {
 System.out.print(getFullName("Joyce", "Hopper"));
 }
}
```

- A.** Joyce Hopper
- B.** Finished!Joyce Hopper
- C.** Problem?Finished!null
- D.** None of the above

**29.** Fill in the blanks: A try statement has \_\_\_\_\_ finally block(s) and \_\_\_\_\_ catch blocks.

- A.** zero or one, zero or more
- B.** one, one or more
- C.** zero or one, zero or one
- D.** one or more, zero or one

**30.** What is the output of the following application?

```
package pond;
abstract class Duck {
 protected int count;
 public abstract int getDuckies();
}
public class Ducklings extends Duck {
 private int age;
 public Ducklings(int age) { this.age = age; }
 public int getDuckies() { return this.age/count; }
 public static void main(String[] pondInfo) {
```

```
 Duck itQuacks = new Ducklings(5);
 System.out.print(itQuacks.getDuckies());
 }
}
```

- A.** 0
- B.** 5
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

- 31.** Given a `try` statement, if both the `catch` block and the `finally` block each throw an exception, what does the caller see?
- A.** The exception from the `catch` block
  - B.** The exception from the `finally` block
  - C.** Both the exception from the `catch` block and the exception from the `finally` block
  - D.** None of the above
- 32.** What is the output of the following application?

```
package zoo;
class BigCat {
 void roar(int level) throw RuntimeException { // m1
 if(level<3) throw new IllegalArgumentException("Incomplete");
 System.out.print("Roar!");
 }
}
public class Lion extends BigCat {
 public void roar() { // m2
 System.out.print("Roar!!!");
 }
}
public static void main(String[] cubs) {
 final BigCat kitty = new Lion(); // m3
 kitty.roar(2);
}
```

- A.** The code does not compile because of line m1.
- B.** The code does not compile because of line m2.
- C.** The code does not compile because of line m3.
- D.** The code compiles but a stack trace is printed at runtime.

33. Given the following code snippet, which specific exception will be thrown?

```
final Object exception = new Exception();
final Exception data = (RuntimeException)exception;
System.out.print(data);
```

- A. ClassCastException
- B. RuntimeException
- C. NullPointerException
- D. None of the above

34. Which of the following classes will handle all types in a catch block?

- A. Exception
- B. Error
- C. Throwable
- D. RuntimeException

35. In the following application, the values of `street` and `city` have been omitted. Which one of the following is a possible output of executing this class?

- I. 350 5th Ave - New York
- II. Posted:350 5th Ave - New York

```
package registration;
public class Address {
 public String getAddress(String street, String city) {
 try {
 return street.toString() + " : " + city.toString();
 } finally {
 System.out.print("Posted:");
 }
 }
 public static void main(String[] form) {
 String street = // value omitted
 String city = // value omitted
 System.out.print(new Address().getAddress(street,city));
 }
}
```

- A. I only
- B. II only
- C. I and II
- D. None of the above

- 36.** If a try statement has catch blocks for both `ClassCastException` and `RuntimeException`, then which of the following statements is correct?
- A.** The catch block for `ClassCastException` must appear before the catch block for `RuntimeException`.
  - B.** The catch block for `RuntimeException` must appear before the catch block for `ClassCastException`.
  - C.** The catch blocks for these two exception types can be declared in any order.
  - D.** A try statement cannot be declared with these two catch block types because they are incompatible.
- 37.** Which of the following is the best scenario to use an exception?
- A.** The computer caught fire.
  - B.** The code does not compile.
  - C.** A caller passes invalid data to a method.
  - D.** A method finishes sooner than expected.
- 38.** What is the output of the following application?

```
package body;
class Organ {
 public void operate() throws RuntimeException {
 throw new RuntimeException("Not supported");
 }
}
public class Heart extends Organ {
 public void operate() throws Exception {
 System.out.print("beat");
 }
 public static void main(String... cholesterol) throws Exception {
 try {
 new Heart().operate();
 } finally {
 }
 }
}
A. beat
B. Not supported
C. The code does not compile.
D. The code compiles but a stack trace is printed at runtime.
```

39. Which statement about the following exception statement is correct?

```
throw new NullPointerException();
```

- A. The code where this is called must include a try-catch block that handles this exception.
- B. The method where this is called must declare a compatible exception.
- C. This exception cannot be handled.
- D. This exception can be handled with a try-catch block or ignored altogether by the surrounding method.

40. What is the output of the following application?

```
package clothing;
public class Coat {
 public Long zipper() throws Exception {
 try {
 String checkZipper = (String) new Object();
 } catch (Exception e) {
 throw RuntimeException("Broken!");
 }
 return null;
 }
 public static void main(String... warmth) {
 try {
 new Coat().zipper();
 System.out.print("Finished!");
 } catch (Throwable t) {}
 }
}
```

- A. Finished!
- B. Finished!, followed by a stack trace
- C. The application does not produce any output at runtime.
- D. The code does not compile.

41. Given the following application, which type of exception will be printed in the stack trace at runtime?

```
package carnival;
public class WhackAnException {
 public static void main(String... hammer) {
 try {
 throw new ClassCastException();
 }
 }
}
```

```
 } catch (IllegalArgumentException e) {
 throw new IllegalArgumentException();
 } catch (RuntimeException e) {
 throw new NullPointerException();
 } finally {
 throw new RuntimeException();
 }
 }
}
```

- A. `IllegalArgumentException`
  - B. `NullPointerException`
  - C. `RuntimeException`
  - D. The code does not compile.
42. Which of these method signatures is allowed in a class implementing the `Outfielder` interface?
- ```
class OutOfBoundsException extends BadCatchException {}
class BadCatchException extends Exception {}

public interface Outfielder {
    public void catchBall() throws OutOfBoundsException;
}
```
- A. `public int catchBall() throws OutOfBoundsException`
 - B. `public int catchBall() throws BadCatchException`
 - C. `public int catchBall() throws Exception`
 - D. None of the above
43. What is the output of the following application?

```
package city;
public class Street {
    public static void dancing() throws RuntimeException {
        try {
            throw new IllegalArgumentException();
        } catch (Error) {
            System.out.print("Unable!");
        }
    }
    public static void main(String... count) throws RuntimeException {
        dancing();
    }
}
```

- A. Unable!
 - B. The application does not produce any output.
 - C. The application compiles but produces a stack trace at runtime.
 - D. The code does not compile.
44. What is the result of compiling and running the following application?
- ```
package castles;
class DragonException extends Exception {}
public class Lair {
 public void openDrawbridge() throws Exception { // r1
 try {
 throw new Exception("This Exception");
 } catch (RuntimeException e) {
 throw new DragonException(); // r2
 } finally {
 throw new RuntimeException("Or maybe this one");
 }
 }
 public static void main(String[] moat) throws Exception {
 new Lair().openDrawbridge(); // r3
 }
}
```
- A. The code does not compile because of line r1.
  - B. The code does not compile because of line r2.
  - C. The code does not compile because of line r3.
  - D. The code compiles, but a stack trace is printed at runtime.
45. If a try statement has catch blocks for both `IllegalArgumentException` and `ClassCastException`, then which of the following statements is correct?
- A. The catch block for `IllegalArgumentException` must appear before the catch block for `ClassCastException`.
  - B. The catch block for `ClassCastException` must appear before the catch block for `IllegalArgumentException`.
  - C. The catch blocks for these two exception types can be declared in any order.
  - D. A try statement cannot be declared with these two catch block types because they are incompatible.

**46.** What is the output of the following application?

```
package broken;
class Problem implements RuntimeException {}
public class BiggerProblem extends Problem {
 public static void main(String uhOh[]) {
 try {
 throw new BiggerProblem();
 } catch (BiggerProblem re) {
 System.out.print("Problem?");
 } catch (Problem e) {
 System.out.print("Handled");
 } finally {
 System.out.print("Fixed!");
 }
 }
}
```

- A.** Problem?Fixed!
- B.** Handled.Fixed!
- C.** Problem?Handled.Fixed!
- D.** The code does not compile.

**47.** What is the output of the following application?

```
package lighting;
interface Source {
 void flipSwitch() throws Exception;
}
public class LightBulb implements Source {
 public void flipSwitch() {
 try {
 throws new RuntimeException("Circuit Break!");
 } finally {
 System.out.print("Flipped!");
 }
 }
 public static void main(String... electricity) throws Throwable {
 final Source bulb = new LightBulb();
 bulb.flipSwitch();
 }
}
```

- A. A stack trace for a `RuntimeException`
  - B. Flipped!, followed by a stack trace for a `RuntimeException`
  - C. The code does not compile because `flipSwitch()` is an invalid method override.
  - D. The code does not compile for another reason.
48. Given an application that hosts a website, which of the following would most likely result in a `java.lang.Error` being thrown?
- A. Two users try to register an account at the same time.
  - B. The application temporarily loses connection to the network.
  - C. A user enters their password incorrectly.
  - D. The application runs out of memory.
49. Given that `FileNotFoundException` is a subclass of `IOException`, what is the output of the following application?

```
package storage;
import java.io.*;
public class Backup {
 public void performBackup() {
 try {
 throw new IOException("Disk not found");
 } catch (Exception e) {
 try {
 throw new FileNotFoundException("File not found");
 } catch (FileNotFoundException e) { // z1
 System.out.print("Failed");
 }
 }
 }
 public static void main(String... files) {
 new Backup().performBackup(); // z2
 }
}
```

- A. Failed
- B. The application compiles but a stack trace is printed at runtime.
- C. The code does not compile because of line z1.
- D. The code does not compile because of line z2.

**50.** What is the output of the following application?

```
package bed;
public class Sleep {
 public static void snore() {
 try {
 String sheep[] = new String[3];
 System.out.print(sheep[3]);
 } catch (RuntimeException e) {
 System.out.print("Awake!");
 } finally {
 throw new Exception(); // x1
 }
 }
 public static void main(String... sheep) { // x2
 new Sleep().snore(); // x3
 }
}
```

- A.** Awake!, followed by a stack trace
- B.** The code does not compile because of line x1.
- C.** The code does not compile because of line x2.
- D.** The code does not compile because of line x3.

# Chapter 9



## Working with Selected Classes from the Java API

---

**THE OCA EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Working with Selected classes from the Java API**

- Manipulate data using the `StringBuilder` class and its methods
- Create and manipulate `String`s
- Create and manipulate calendar data using classes from `java.time.LocalDateTime`, `java.time.LocalDate`, `java.time.LocalTime`, `java.time.format.DateTimeFormatter`, `java.time.Period`
- Declare and use an `ArrayList` of a given type
- Write a simple Lambda expression that consumes a Lambda Predicate expression

1. What is the best reason for using `StringBuilder` instead of `String`?
  - A. `StringBuilder` adds support for multiple threads.
  - B. `StringBuilder` can use `==` to compare values.
  - C. `StringBuilder` saves memory by reducing the number of objects created.
  - D. `StringBuilder` supports different languages and encodings.
2. What is not true about a `String`?
  - A. It can be created without coding a call to a constructor.
  - B. It can be reused via the string pool.
  - C. It is final.
  - D. It is mutable.
3. Which of the following creates a `StringBuilder` with a different value than the other options?
  - A. `new StringBuilder().append("clown")`
  - B. `new StringBuilder("clown")`
  - C. `new StringBuilder("cl").insert(2, "own")`
  - D. All of them create the same value.
4. What is the output of the following?

```
StringBuilder teams = new StringBuilder("333");
teams.append(" 806");
teams.append(" 1601");
System.out.print(teams);
```

  - A. 333
  - B. 333 806 1601
  - C. The code compiles but outputs something else.
  - D. The code does not compile.
5. How many of the types `ArrayList`, `List`, and `Object` can fill in the blank to produce code that compiles?

```
List frisbees = new _____();
```

- A. None
- B. One
- C. Two
- D. Three

6. What is the output of the following?

```
List<String> tools = new ArrayList<>();
tools.add("hammer");
tools.add("nail");
tools.add("hex key");
System.out.println(tools.get(1));
```

- A. hammer
- B. hex key
- C. nail
- D. None of the above

7. What is the result of the following code?

```
StringBuilder sb = new StringBuilder("radical")
 .insert(sb.length(), "robots");
System.out.println(sb);
```

- A. radicarobots
- B. radicalrobots
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

8. What is the output of the following?

```
List<String> museums = new ArrayList<>(1);
museums.add("Natural History");
museums.add("Science");
museums.add("Art");
museums.remove(2);
System.out.println(museums);
```

- A. [Natural History, Science]
- B. [Natural History, Art, Science]
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

9. What is the output of the following?

```
12: StringBuilder b = new StringBuilder("12");
13: b = b.append("3");
14: b.reverse();
15: System.out.println(b.toString());
```

- A.** 12  
**B.** 123  
**C.** 321  
**D.** The code does not compile.
- 10.** What is the main benefit of a lambda expression?
- A.** It allows you to convert a primitive to a wrapper class.  
**B.** It allows you to change the bytecode while the application is running.  
**C.** It allows you to inherit from multiple classes.  
**D.** It allows you to write code that has the execution deferred.
- 11.** What is the output of the following?
- ```
5: StringBuilder line = new StringBuilder("-");  
6: StringBuilder anotherLine = line.append("-");  
7: System.out.print(line == anotherLine);  
8: System.out.print(" ");  
9: System.out.print(line.length());
```
- A.** false 1
B. false 2
C. true 1
D. true 2
- 12.** The author of this method forgot to include the data type. Which of the following reference types can fill in the blank to complete this method?
- ```
public static void secret(____ mystery) {
 mystery.add("metal");
 String str = mystery.get(0);
 int num = mystery.length();
}
```
- A.** ArrayList  
**B.** ArrayList<String>  
**C.** StringBuilder  
**D.** None of the above
- 13.** Which portion of code can be removed so that this line of code continues to compile?
- ```
Predicate<StringBuilder> p = (StringBuilder b) -> {return true;};
```
- A.** Remove StringBuilder b
B. Remove ->
C. Remove { and ;}
D. Remove { return and ;}

14. What is the output of the following?

```
20: List<Character> chars = new ArrayList<>();  
21: chars.add('a');  
22: chars.add('b');  
23: chars.set(1, 'c');  
24: chars.remove(0);  
25: System.out.print(chars.size() + " " + chars.contains('b'));  
  
A. 1 false  
B. 1 true  
C. 2 false  
D. 2 true
```

15. What is the output of the following?

```
12: String b = "12";  
13: b += "3";  
14: b.reverse();  
15: System.out.println(b.toString());  
  
A. 12  
B. 123  
C. 321  
D. The code does not compile.
```

16. How many of these lines fail to compile?

```
Predicate<String> pred1 = s -> false;  
Predicate<String> pred2 = (s) -> false;  
Predicate<String> pred3 = String s -> false;  
Predicate<String> pred4 = (String s) -> false;
```

- A.** One
- B.** Two
- C.** Three
- D.** Four

17. What does the following do?

```
public class Shoot {  
    interface Target {  
        boolean needToAim(double angle);  
    }  
    static void prepare(double angle, Target t) {  
        boolean ready = t.needToAim(angle); // k1
```

```
        System.out.println(ready);
    }
    public static void main(String[] args) {
        prepare(45, d -> d > 5 || d < -5); // k2
    }
}
```

- A.** It prints true.
B. It prints false.
C. It doesn't compile due to line k1.
D. It doesn't compile due to line k2.
- 18.** What is the output of the following?
- ```
String teams = new String("694");
teams.concat(" 1155");
teams.concat(" 2265");
teams.concat(" 2869");
System.out.println(teams);
```
- A.** 694  
**B.** 694 1155 2265 2869  
**C.** The code compiles but outputs something else.  
**D.** The code does not compile.
- 19.** Which of these classes are in the `java.util` package?
- I.** `ArrayList`  
**II.** `LocalDate`  
**III.** `String`
- A.** I only  
**B.** II only  
**C.** I and II  
**D.** I, II, and III
- 20.** Which of the answer choices results in a different value being output than the other three choices?

```
StringBuilder sb = new StringBuilder("radical ");
sb = _____;
System.out.print(sb);
```

- A. `new StringBuilder("radical ")  
 .append("robots")`
- B. `new StringBuilder("radical ")  
 .delete(1, 100)  
 .append("obots")  
 .insert(1, "adical r")`
- C. `new StringBuilder("radical ")  
 .insert(7, "robots")`
- D. `new StringBuilder("radical ")  
 .insert(sb.length(), "robots")`
- 21.** What is the output of the following?
- ```
String[] array = {"Natural History", "Science"};  
List<String> museums = Arrays.asList(array);  
museums.set(0, "Art");  
System.out.println(museums.contains("Art"));
```
- A. `true`
- B. `false`
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.
- 22.** Which is a true statement?
- A. If `s.contains("abc")` is true, then `s.equals("abc")` is also true.
- B. If `s.contains("abc")` is true, then `s.startsWith("abc")` is also true.
- C. If `s.startsWith("abc")` is true, then `s.equals("abc")` is also true.
- D. If `s.startsWith("abc")` is true, then `s.contains("abc")` is also true.
- 23.** What is the output of the following?
- ```
20: List<Character> chars = new ArrayList<>();
21: chars.add('a');
22: chars.add('b');
23: chars.set(1, 'c');
24: chars.remove(0);
25: System.out.print(chars.length());
```
- A. `0`
- B. `1`
- C. `2`
- D. None of the above

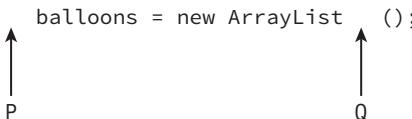
24. The author of this method forgot to include the data type. Which of the following reference types can fill in the blank to complete this method?

```
public static void secret(____ mystery) {
 mystery = mystery.replace("1", "8");
 mystery.startsWith("paper");
 String s = mystery.toString();
}
```

- A. `ArrayList`
- B. `String`
- C. `StringBuilder`
- D. None of the above

25. Which statement is true about the following figure while ensuring the code continues to compile?

```
List balloons = new ArrayList();
```



- A. `<>` can be inserted at position P without making any other changes.
- B. `<>` can be inserted at position Q without making any other changes.
- C. `<>` can be inserted at both positions P and Q.
- D. None of the above

26. Which of the following can fill in the blank to make the code compile?

```
import java.util.function.*;
public class Card {
 public static void main(String[] s) {
 Predicate<String> pred = _____ -> true;
 }
}
```

- A. `(Integer i)`
- B. `(Object o)`
- C. `(String s)`
- D. None of the above

27. What is the output of the following?

```
5: String line = new String("-");
6: String anotherLine = line.concat("-");
7: System.out.print(line == anotherLine);
8: System.out.print(" ");
9: System.out.print(line.length());
```

- A. false 1  
B. false 2  
C. true 1  
D. true 2
28. What does the following output?
- ```
Predicate dash = c -> c.startsWith("-");  
System.out.println(dash.test("-"));
```
- A. true
B. false
C. The code does not compile.
D. The code compiles but throws an exception at runtime.
29. Of the classes `LocalDate`, `LocalDateTime`, `LocalTime`, and `LocalTimeStamp`, how many include hours, minutes, and seconds?
- A. One
B. Two
C. Three
D. Four
30. What is the output of the following class?
- ```
1: package rocket;
2: public class Countdown {
3: public static void main(String[] args) {
4: String builder = "54321";
5: builder = builder.substring(4);
6: System.out.println(builder.charAt(2));
7: }
8: }
```
- A. 2  
B. 3  
C. 4  
D. None of the above
31. Which equivalent code can replace `i -> i != 0` in the following line?
- ```
Predicate<Integer> ip = i -> i != 0;
```
- A. `i -> { i != 0 }`
B. `i -> { i != 0; }`
C. `i -> { return i != 0 }`
D. `i -> { return i != 0; }`

32. What is the output of the following?

```
LocalDate xmas = LocalDate.of(2016, 12, 25);
xmas.plusDays(-1);
System.out.println(xmas.getDayOfMonth());
```

- A.** 24
- B.** 25
- C.** 26
- D.** None of the above

33. What is the output of the following?

```
1:  public class Legos {
2:      public static void main(String[] args) {
3:          StringBuilder sb = new StringBuilder();
4:          sb.append("red");
5:          sb.deleteCharAt(0);
6:          sb.delete(1, 2);
7:          System.out.println(sb);
8:      }
9:  }
```

- A.** e
- B.** d
- C.** ed
- D.** None of the above

34. What does the following output?

```
Predicate clear = c -> c.equals("clear");
System.out.println(clear.test("pink"));
```

- A.** true
- B.** false
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

35. Which starts counting from one rather than zero?

- A.** Array indexes
- B.** The index used by `charAt` in a `String`
- C.** The months in a `LocalDateTime`
- D.** The months in a `LocalTime`

- 36.** Which statement is not true of `Predicate`?
- A.** A `boolean` is returned from the method it declares.
 - B.** It is an interface.
 - C.** The method it declares accepts two parameters.
 - D.** The method it declares is named `test`.

- 37.** Which of these periods represents a larger amount of time?

```
Period period1 = Period.ofWeeks(1).ofDays(3);
Period period2 = Period.ofDays(10);
```

- A.** `period1`
- B.** `period2`
- C.** They represent the same length of time.
- D.** None of the above. This code does not compile.

- 38.** What is the result of the following?

```
import java.time.*;
import java.time.format.*;

public class HowLong {
    public static void main(String[] args) {
        LocalDate newYears = LocalDate.of(2017, 1, 1);
        Period period = Period.ofDays(1);
        DateTimeFormatter format = DateTimeFormatter.ofPattern("MM-dd-yyyy");
        System.out.print(format.format(newYears.minus(period)));
    }
}
```

- A.** 01-01-2017
- B.** 12-31-2016
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

- 39.** Which of the following can fill in the blank so the following code prints `true`?

```
String happy = " :) - (: ";
String really = happy.trim();
String question = _____;
System.out.println(really.equals(question));
```

- A. happy.substring(0, happy.length() - 1)
- B. happy.substring(0, happy.length())
- C. happy.substring(1, happy.length() - 1)
- D. happy.substring(1, happy.length())

40. Which is not a true statement about the Period class?

- A. A Period is immutable.
- B. A Period is typically used for adding or subtracting time from dates.
- C. You can create a Period representing 2 minutes.
- D. You can create a Period representing 5 years.

41. What is the output of the following class?

```
1: package rocket;
2: public class Countdown {
3:     public static void main(String[] args) {
4:         StringBuilder builder = new StringBuilder("54321");
5:         builder.substring(2);
6:         System.out.println(builder.charAt(1));
7:     }
8: }
```

- A. 1
- B. 2
- C. 3
- D. 4

42. What does the following output?

```
List<Integer> pennies = new ArrayList<>();
pennies.add(3);
pennies.add(2);
pennies.add(1);
pennies.remove(2);
System.out.println(pennies);
```

- A. [3, 1]
- B. [3, 2]
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

- 43.** The author of this method forgot to include the data type. Which of the following reference types can best fill in the blank to complete this method?

```
public static void secret(____ mystery) {  
    char ch = mystery.charAt(3);  
    mystery = mystery.insert(1, "more");  
    int num = mystery.length();  
}
```

- A.** `ArrayList`
- B.** `String`
- C.** `StringBuilder`
- D.** None of the above

- 44.** What is the smallest unit you can add to a `LocalTime` object?

- A.** `Second`
- B.** `Millisecond`
- C.** `Nanosecond`
- D.** `Picosecond`

- 45.** What is the result of the following?

```
import java.time.*;  
import java.time.format.*;  
  
public class HowLong {  
    public static void main(String[] args) {  
        LocalDate newYears = LocalDate.of(2017, 1, 1);  
        Period period = Period.ofDays(1);  
        DateTimeFormatter format = DateTimeFormatter.ofPattern("mm-dd-yyyy");  
        System.out.print(format.format(newYears.minus(period)));  
    }  
}
```

- A.** 01-01-2017
- B.** 12-31-2016
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

46. Which of the following types can you pass as a parameter to the `replace()` method on the `String` class?

- I.** `char`
 - II.** `String`
 - III.** `StringBuilder`
- A.** I
 - B.** I and II
 - C.** II and III
 - D.** I, II, and III

47. How many lines does this code output?

```
import java.util.*;  
import java.util.function.*;  
  
public class PrintNegative {  
    public static void main(String[] args) {  
        List<String> list = new ArrayList<>();  
        list.add("-5");  
        list.add("0");  
        list.add("5");  
        print(list, e -> e < 0);  
    }  
    public static void print(List<String> list, Predicate<Integer> p) {  
        for (String num : list)  
            if (p.test(Integer.parseInt(num)))  
                System.out.println(num);  
    }  
}
```

- A.** One
- B.** Two
- C.** None. The code does not compile.
- D.** None. The code throws an exception at runtime.

48. What is the output of the following?

```
12: List<String> magazines = new ArrayList();  
13: magazines.add("Readers Digest");  
14: magazines.add("People");  
15: magazines.clear();  
16: magazines.add("The Economist");  
17: magazines.remove(1);  
18: System.out.println(magazines.size());
```

- A.** 0
- B.** 1
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

49. What is the output of the following?

```
public class Costume {  
    public static void main(String[] black) {  
        String witch = 'b';  
        String tail = "lack";  
        witch = witch.concat(tail);  
        System.out.println(witch);  
    }  
}
```

- A.** b
- B.** black
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

50. What is the result of the following?

```
LocalDate xmas = LocalDate.of(2016, 12, 25);  
xmas.setYear(2017);  
System.out.println(xmas.getYear());
```

- A.** 2016
- B.** 2017
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

Chapter 10



OCA Practice Exam

This chapter contains 80 questions and is designed to simulate a real OCA exam. While previous chapters were focused on a specific set of objectives, this chapter covers all of the objectives on the exam. We recommend you take this exam only after you score well on the questions in the individual chapters.

For this chapter, you should try to simulate the real exam experience as much as possible. This means setting aside 150 minutes of uninterrupted time to complete the test, as well as not looking at any reference material while taking the exam. If you don't know an answer to a question, complete it as best you can and move on to the next question, just as you would on a real exam.

Remember, the exam permits writing material, such as a whiteboard. If you do not have a whiteboard handy, you can just use blank sheets of paper and a pencil. If you do well on this test, then you are hopefully ready to take the real exam. With that said, good luck!

1. What is the output if this class is run with `java Indexing cars carts`?

```
public class Indexing {  
    public static void main(String... books) {  
        StringBuilder sb = new StringBuilder();  
        for (String book : books)  
            sb.insert(sb.indexOf("c"), book);  
        System.out.println(sb);  
    }  
}
```

- A. cars
- B. cars carts
- C. ccars arts
- D. The code does not compile.
- E. The code compiles but throws an exception at runtime.

2. Fill in the blanks: The operators `+=`, `—`, `—`, `—`, `—`, and `++` are listed in increasing or the same level of operator precedence. (Choose two.)

- A. `—, +, =, --`
- B. `%, *, /, +`
- C. `=, +, /, *`
- D. `^, *, -, ==`
- E. `*, /, %, --`

3. Which of the following are valid JavaBean signatures? (Choose three.)

- A. `public byte getNose(String nose)`
- B. `public void setHead(int head)`
- C. `public String getShoulders()`
- D. `public long isMouth()`
- E. `public void gimmeEars()`
- F. `public boolean isToes()`

4. Which of the following are true? (Choose two.)

```
20: int[] crossword [] = new int[10][20];  
21: for (int i = 0; i < crossword.length; i++)  
22:     for (int j = 0; j < crossword.length; j++)  
23:         crossword[i][j] = 'x';  
24: System.out.println(crossword.size());
```

- A.** One line needs to be changed for this code to compile.
 - B.** Two lines need to be changed for this code to compile.
 - C.** Three lines need to be changed for this code to compile.
 - D.** If the code is fixed to compile, none of the cells in the 2D array have a value of 0.
 - E.** If the code is fixed to compile, half of the cells in the 2D array have a value of 0.
 - F.** If the code is fixed to compile, all of the cells in the 2D array have a value of 0.
- 5.** Which of the following statements about `java.lang.Error` are most accurate? (Choose two.)
- A.** An `Error` should be thrown if a file system resource becomes temporarily unavailable.
 - B.** An application should never catch an `Error`.
 - C.** `Error` is a subclass of `Exception`, making it a checked exception.
 - D.** It is possible to catch and handle an `Error` thrown in an application.
 - E.** An `Error` should be thrown if a user enters invalid input.
- 6.** Given a class that uses the following `import` statements, which class would be automatically accessible without using its full package name? (Choose three.)
- ```
import forest.Bird;
import jungle.tree.*;
import savana.*;
```
- A.** `forest.Bird`
  - B.** `savana.sand.Wave`
  - C.** `jungle.tree.Huicungo`
  - D.** `java.lang.Object`
  - E.** `forest.Sloth`
  - F.** `forest.ape.bonobo`
- 7.** How many of the following variables represent immutable objects?
- ```
ArrayList l = new ArrayList();
String s = new String();
StringBuilder sb = new StringBuilder();
LocalDateTime t = LocalDateTime.now();
```
- A.** None
 - B.** One
 - C.** Two
 - D.** Three
 - E.** Four
 - F.** None of the above—this code doesn't compile.

8. What is the output of the following?

```
StringBuilder builder = new StringBuilder("Leaves growing");
do {
    builder.delete(0, 5);
} while (builder.length() > 5);
System.out.println(builder);
```

- A. Leaves growing
- B. ing
- C. wing
- D. The code does not compile.
- E. The code compiles but throws an exception at runtime.

9. What is the output of the following application?

```
package reality;
public class Equivalency {
    public static void main(String[] edges) {
        final String ceiling = "up";
        String floor = new String("up");
        final String wall = new String(floor);
        System.out.print((ceiling==wall)
            +" "+(floor==wall)
            +" "+ceiling.equals(wall));
    }
}
```

- A. false false false
- B. true true true
- C. false true true
- D. false false true
- E. It does not compile.

10. How many times does the following code print true?

```
1:  public class Giggles {
2:      public static void main(String[] args) {
3:          String lol = "lol";
4:          System.out.println(lol.toUpperCase() == lol);
5:          System.out.println(lol.toUpperCase() == lol.toUpperCase());
6:          System.out.println(lol.toUpperCase().equals(lol));
7:          System.out.println(lol.toUpperCase().equals(lol.toUpperCase())));
8:          System.out.println(lol.toUpperCase().equalsIgnoreCase(lol));
```

```
9:         System.out.println(lol.toUpperCase())
10:            .equalsIgnoreCase(lol.toUpperCase()));
11:    } }
```

- A.** One
- B.** Two
- C.** Three
- D.** Four
- E.** Five
- F.** None. The code does not compile.

- 11.** Which lines can be removed together without stopping the code from compiling and while printing the same output? (Choose three.)

```
14: String race = "";
15: outer:
16: do {
17: inner:
18:   do {
19:     race += "x";
20:   } while (race.length() <= 4);
21: } while (race.length() < 4);
22: System.out.println(race);
```

- A.** Lines 15 and 17
- B.** Lines 15, 16, and 21
- C.** Line 17
- D.** Lines 17, 18, and 20
- E.** Line 20
- F.** Line 21

- 12.** Which of the following do not compile when filling in the blank? (Choose two.)

```
long bigNum = _____;
```

- A.** 1234
- B.** 1234.0
- C.** 1234.0L
- D.** 1234l
- E.** 1234L
- F.** 1_234

13. How many lines does this program print?

```
import java.time.*;
public class OnePlusOne {
    public static void main(String... nums) {
        LocalTime time = LocalTime.of(1, 11);
        while (time.getHour() < 1) {
            time.plusHours(1);
            System.out.println("in loop");
        }
    }
}
```

- A.** None
- B.** One
- C.** Two
- D.** This is an infinite loop.
- E.** The code does not compile.

14. What is the result of running the following program?

```
1: package fun;
2: public class Sudoku {
3:     static int[][] game;
4:
5:     public static void main(String args[]) {
6:         game[3][3] = 6;
7:         Object[] obj = game;
8:         obj[3] = 'X';
9:         System.out.println(game[3][3]);
10:    }
11: }
```

- A.** 6
- B.** X
- C.** The code does not compile.
- D.** The code compiles but throws a `NullPointerException` at runtime.
- E.** The code compiles but throws a different exception at runtime.
- F.** The output is not guaranteed.

15. Which of the following use generics and compile without warnings? (Choose two.)

- A. `List<String> a = new ArrayList();`
- B. `List<> b = new ArrayList();`
- C. `List<String> c = new ArrayList<>();`
- D. `List<> d = new ArrayList<>();`
- E. `List<String> e = new ArrayList<String>();`
- F. `List<> f = new ArrayList<String>();`

16. Which of the following are true right before the `main()` method ends? (Choose two.)

```
public static void main(String[] args) {  
    String shoe1 = new String("sandal");  
    String shoe2 = new String("flip flop");  
    String shoe3 = new String("croc");  
  
    shoe1 = shoe2;  
    shoe2 = shoe3;  
    shoe3 = shoe1;  
}
```

- A. No objects are eligible for garbage collection.
- B. One object is eligible for garbage collection.
- C. Two objects are eligible for garbage collection.
- D. No objects are guaranteed to be garbage collected.
- E. One object is guaranteed to be garbage collected.
- F. Two objects are guaranteed to be garbage collected.

17. How many lines of the following application do not compile?

```
package ocean;  
class BubbleException extends Exception {}  
class Fish {  
    Fish getFish() throws BubbleException {  
        throw new RuntimeException("fish!");  
    }  
}  
public final class Clownfish extends Fish {  
    public final Clownfish getFish() {  
        throw new RuntimeException("clown!");  
    }  
}  
public static void main(String[] bubbles) {  
    final Fish f = new Clownfish();
```

```
        f.getFish();
        System.out.println("swim!");
    }
}
```

- A. None. The code compiles and prints `swim!`.
- B. None. The code compiles and prints a stack trace.
- C. One
- D. Two
- E. Three
- 18.** How many lines does this code output?

```
import java.util.*;
import java.util.function.*;

public class PrintNegative {

    public static void main(String[] args) {
        List<Integer> list= new ArrayList<>();
        list.add(-5);
        list.add(0);
        list.add(5);
        print(list, e -> e < 0);
    }

    public static void print(List<Integer> list, Predicate<Integer> p) {
        for (Integer num : list)
            if (p.test(num))
                System.out.println(num);
    }
}
```

- A. One
- B. Two
- C. Three
- D. None. It doesn't compile.
- E. None. It throws an exception at runtime.
- 19.** Which keywords are required with a `try` statement?
- I. `finalize`
 - II. `catch`
 - III. `throws`
 - IV. `finally`

- A.** I only
 - B.** II only
 - C.** III only
 - D.** IV only
 - E.** I or II, or both
 - F.** None of the above
- 20.** What is the output of the following?
- ```
12: int result = 8;
13: loop: while (result > 7) {
14: result++;
15: do {
16: result--;
17: } while (result > 5);
18: break loop;
19: }
20: System.out.println(result);
```
- A.** 5
  - B.** 7
  - C.** 8
  - D.** The code does not compile.
  - E.** The code compiles but throws an exception at runtime.
- 21.** What is the result of compiling and executing the following application?

```
package reptile;
public class Alligator {
 static int teeth;
 double scaleToughness;
 public Alligator() {
 teeth++;
 }
 public void snap(int teeth) {
 System.out.print(teeth+" ");
 teeth--;
 }
 public static void main(String[] unused) {
 new Alligator().snap(teeth);
 new Alligator().snap(teeth);
 }
}
```

- A.** 0 1
  - B.** 1 1
  - C.** 1 2
  - D.** 2 2
  - E.** The code does not compile.
  - F.** The code compiles but produces an exception at runtime.
- 22.** What is the output of the following?

```
public class Costume {
 public static void main(String[] black) {
 String witch = "b";
 String tail = "lack";
 witch.concat(tail);
 System.out.println(witch);
 }
}
```

- A.** b
  - B.** black
  - C.** lack
  - D.** The code does not compile.
  - E.** The code compiles but throws an exception at runtime.
- 23.** Which modifiers can be independently applied to an interface method? (Choose three.)

- A.** default
- B.** protected
- C.** static
- D.** private
- E.** final
- F.** abstract

- 24.** What is the output of the following?

```
public class Shoelaces {
 public static void main(String[] args) {
 String tie = null;
 while (tie = null)
 tie = "shoelace";
 System.out.print(tie);
 }
}
```

- A.** null
  - B.** shoelace
  - C.** shoelaceshoelace
  - D.** The code does not compile.
  - E.** This is an infinite loop.
  - F.** The code compiles but throws an exception at runtime.
- 25.** What statements are true about compiling a Java class file? (Choose two.)
- A.** If the file does not contain a package statement, then the compiler considers the class part of the `java.lang` package.
  - B.** The compiler assumes every class implicitly imports the `java.lang.*` package.
  - C.** The compiler assumes every class implicitly imports the `java.util.*` package.
  - D.** Java requires every file to declare a package statement.
  - E.** Java requires every file to declare at least one `import` statement.
  - F.** If the class declaration does not extend another class, then it implicitly extends the `java.lang.Object` class.
- 26.** What is the output of the following application?

```
package woods;
interface Plant {
 default String grow() { return "Grow!"; }
}
interface Living {
 public default String grow() { return "Growing!"; }
}
public class Tree implements Plant, Living { // m1
 public String grow(int height) { return "Super Growing!"; }
 public static void main(String[] leaves) {
 Plant p = new Tree(); // m2
 System.out.print(((Living)p).grow()); // m3
 }
}
```

- A.** Grow!
- B.** Growing!
- C.** Super Growing!
- D.** It does not compile because of line m1.
- E.** It does not compile because of line m2.
- F.** It does not compile because of line m3.

**27.** What is the result of the following?

```
public static void main(String... args) {
 String name = "Desiree";
 int _number = 694;
 boolean profit$$$;
 System.out.println(name + " won. "
 + _number + " profit? " + profit$$$);
}
```

- A.** The declaration of name does not compile.
  - B.** The declaration of \_number does not compile.
  - C.** The declaration of profit\$\$\$ does not compile.
  - D.** The println statement does not compile.
  - E.** The code compiles and runs successfully.
  - F.** The code compiles and throws an exception at runtime.
- 28.** Fill in the blanks: Given a variable x, \_\_\_\_\_ decreases the value of x by 1 and returns the original value, while \_\_\_\_\_ increases the value of x by 1 and returns the new value.
- A.** x--, ++x
  - B.** x--, x++
  - C.** --x, x++
  - D.** --x, ++x
- 29.** Given the following two classes in the same package, which constructors contain compiler errors? (Choose three.)

```
public class Big {
 public Big(boolean stillIn) {
 super();
 }
}

public class Trouble extends Big {
 public Trouble() {}
 public Trouble(int deep) {
 super(false);
 this();
 }
 public Trouble(String now, int... deep) {
 this(3);
 }
}
```

```
 }
 public Trouble(long deep) {
 this("check",deep);
 }
 public Trouble(double test) {
 super(test>5 ? true : false);
 }
}
```

- A.** public Big(boolean stillIn)
  - B.** public Trouble()
  - C.** public Trouble(int deep)
  - D.** public Trouble(String now, int... deep)
  - E.** public Trouble(long deep)
  - F.** public Trouble(double test)
- 30.** Which of the following can replace the comment so this code outputs 100? (Choose two.)
- ```
public class Stats {
    // INSERT CODE
    public static void main(String[] math) {
        System.out.println(max - min);
    }
}
```
- A.** final int min, max = 100;
 - B.** final int min = 0, max = 100;
 - C.** int min, max = 100;
 - D.** int min = 0, max = 100;
 - E.** static int min, max = 100;
 - F.** static int min = 0, max = 100;
- 31.** Which of the following statements are true about Java operators and statements? (Choose two.)
- A.** Both right-hand sides of the ternary expression will be evaluated at runtime.
 - B.** A switch statement may contain at most one default statement.
 - C.** A single if-then statement can have multiple else statements.
 - D.** The | and || operator are interchangeable, always producing the same results at runtime.
 - E.** The ! operator may not be applied to numeric expressions.

32. What is the output of the following?

```
1:  public class Legos {  
2:      public static void main(String[] args) {  
3:          StringBuilder sb = new StringBuilder();  
4:          sb.append("red");  
5:          sb.deleteCharAt(0);  
6:          sb.delete(1, 1);  
7:          System.out.println(sb);  
8:      }  
9:  }
```

- A.** r
- B.** e
- C.** ed
- D.** red
- E.** The code does not compile.
- F.** The code compiles but throws an exception at runtime.

33. Which of the following is a valid method name in Java? (Choose two.)

- A.** _____()
- B.** %run()
- C.** check-Activity()
- D.** \$Hum2()
- E.** sing\\3()
- F.** po#ut ()

34. Which of the following statements about inheritance are true? (Choose two.)

- A.** Inheritance is better than using `static` methods for accessing data in other classes.
- B.** Inheritance allows a method to be overridden in a subclass, possibly changing the expected behavior of other methods in a superclass.
- C.** Inheritance allows objects to inherit commonly used attributes and methods.
- D.** It is possible to create a Java class that does not inherit from any other.
- E.** Inheritance tends to make applications more complicated.

35. Which of the following statements about Java are true?

- I.** The `java` command uses `.` to separate packages.
- II.** Java supports functional programming.
- III.** Java is object oriented.
- IV.** Java supports polymorphism.

- A.** I only
 - B.** II only
 - C.** II and III
 - D.** I, III, and IV
 - E.** I, II, III, and IV
 - F.** None are true.
- 36.** What is the output of the following?
- ```
String[][] listing = new String[][] { { "Book", "34.99" },
 { "Game", "29.99" }, { "Pen", ".99" } };
System.out.println(listing.length + " " + listing[0].length);
```
- A.** 2 2
  - B.** 2 3
  - C.** 3 2
  - D.** 3 3
  - E.** The code does not compile.
  - F.** The code compiles but throws an exception at runtime.
- 37.** Which of the following variable types is permitted in a `switch` statement? (Choose three.)
- A.** Character
  - B.** Byte
  - C.** Double
  - D.** long
  - E.** String
  - F.** Object

- 38.** What does the following do?

```
public class Shoot {
 interface Target {
 boolean needToAim(double angle);
 }
 static void prepare(double angle, Target t) {
 boolean ready = t.needToAim(angle); // k1
 System.out.println(ready);
 }
 public static void main(String[] args) {
 prepare(45, d => d > 5 || d < -5); // k2
 }
}
```

- A. It prints true.
  - B. It prints false.
  - C. It doesn't compile due to line k1.
  - D. It doesn't compile due to line k2.
  - E. It doesn't compile due to another line.
39. Which of the following is a valid code comment in Java? (Choose three.)
- A. `/** Insert */ in next method **/`
  - B. `***** Find the kitty cat */`
  - C. `// Is this a bug?`
  - D. `/ Begin method - performStart() /`
  - E. `/** TODO: Call grandma **/`
  - F. `# Updated code by Patti`
40. Given the following two classes, each in a different package, which lines allow the second class to compile when inserted independently? (Choose two.)

```
package food;
public class Grass {
 public static int seeds = 10;
 public static Grass getGrass() {return new Grass();}
}
```

```
package woods;
```

```
// INSERT CODE HERE
public class Deer {
 public void eat() {
 getGrass();
 System.out.print(seeds);
 }
}
```

- A. `import static food.Grass.getGrass;`  
`import static food.Grass.seeds;`
- B. `import static food.*;`
- C. `static import food.Grass.*;`
- D. `import food.Grass.*;`
- E. `static import food.Grass.getGrass;`  
`static import food.Grass.seeds;`
- F. `import static food.Grass.*;`

41. What is the result of the following?

```
import java.util.*;
public class Museums {
 public static void main(String[] args) {
 String[] array = {"Natural History", "Science", "Art"};
 List<String> museums = Arrays.asList(array);
 museums.remove(2);
 System.out.println(museums);
 }
}
```

- A. [Natural History, Science]
- B. [Natural History, Science, Art]
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

42. Which of the following substitutions will compile? (Choose two.)

```
public class Underscores {
 public String name = "Sherrin";
 public void massage() {
 int zip = 10017;
 }
}
```

- A. Change name to \_name
- B. Change 10017 to \_10017
- C. Change 10017 to 10017\_
- D. Change 10017 to 10\_0\_17
- E. Change int to \_int

43. What is the result of the following when called as `java counting.Binary`?

```
package counting;
import java.util.*;
public class Binary {

 public static void main(String[] args) {
 args = new String[] {"0", "1", "01", "10" };

 Arrays.sort(args);
 System.out.println(Arrays.toString(args));
 }
}
```

- A.** []  
**B.** [0, 01, 1, 10]  
**C.** [0, 01, 10, 1]  
**D.** [0, 1, 01, 10]  
**E.** The code does not compile.  
**F.** The code compiles but throws an exception at runtime.
- 44.** Fill in the blanks: Using the \_\_\_\_\_ and \_\_\_\_\_ modifiers together allows a variable to be accessed from any class, without requiring an instance variable.
- A.** final, package-private  
**B.** class, static  
**C.** protected, instance  
**D.** public, static  
**E.** default, public
- 45.** How many lines does the following code output?
- ```
import java.util.*;
public class Exams {
    public static void main(String[] args) {
        List<String> exams = Arrays.asList("OCA", "OCP");
        for (String e1 : exams)
            for (String e2 : exams)
                System.out.print(e1 + " " + e2);
                System.out.println();
    }
}
```
- A.** One
B. Four
C. Five
D. The code does not compile.
E. The code compiles but throws an exception at runtime.
- 46.** Which of the following are true statements? (Choose two.)
- A.** The javac command compiles a source text file into a set of machine instructions.
B. The java command compiles a .class file into a .java file.
C. The javac command compiles a .java file into a .class file.
D. The javac command compiles a source text file into a bytecode file.
E. The java command compiles a .java file into a .class file.
F. The javac command compiles a .class file into a .java file.

47. How many of the following lines of code compile?

```
char one = Integer.parseInt("1");
Character two = Integer.parseInt("2");
int three = Integer.parseInt("3");
Integer four = Integer.parseInt("4");
short five = Integer.parseInt("5");
Short six = Integer.parseInt("6");
```

- A. None
- B. One
- C. Two
- D. Three
- E. Four
- F. Five

48. Given the application below, what data types can be inserted into the blank that would allow the code to print 3? (Choose three.)

```
public class Highway {
    public int drive(long car) { return 2; }
    public int drive(double car) { return 3; }
    public int drive(int car) { return 5; }
    public int drive(short car) { return 3; }
    public static void main(String[] gears) {
        _____ value = 5;
        System.out.print(new Highway().drive(value));
    }
}
```

- A. boolean
- B. short
- C. int
- D. byte
- E. long
- F. float

49. How many times does this code print true?

```
import java.time.*;
public class Equality {
    public void main(String[] args) {
        System.out.println(new StringBuilder("zelda")
            == new StringBuilder("zelda"));
```

```
        System.out.println(3 == 3);
        System.out.println("bart" == "bart");
        System.out.println(new int[0] == new int[0]);
        System.out.println(LocalTime.now() == LocalTime.now());
    }
}
```

- A.** None
- B.** One
- C.** Two
- D.** Three
- E.** The code does not compile.

- 50.** What is the output of the following application?

```
package ballroom;
public class Dance {
    public static void swing(int... beats) throws ClassCastException {
        try {
            System.out.print("1"+beats[2]); // p1
        } catch (RuntimeException e) {
            System.out.print("2");
        } catch (Exception e) {
            System.out.print("3");
        } finally {
            System.out.print("4");
        }
    }
    public static void main(String... music) {
        new Dance().swing(0,0); // p2
        System.out.print("5");
    }
}
```

- A.** 145
- B.** 1045
- C.** 24, followed by a stack trace
- D.** 245
- E.** The code does not compile because of line p1.
- F.** The code does not compile because of line p2.

51. What is the output of the following?

```
List<String> drinks = Arrays.asList("can", "cup");
for (int container = drinks.size(); container > 0; container++) {
    System.out.print(drinks.get(container-1) + ",");
}
```

- A. can,cup,
- B. cup,can,
- C. The code does not compile.
- D. This is an infinite loop.
- E. The code compiles but throws an exception at runtime.

52. Which of the following method signatures are valid declarations of an entry point in a Java application? (Choose three.)

- A. public static void main(String... widgets)
- B. public static void main(String sprockets)
- C. protected static void main(String[] args)
- D. public static int void main(String[] arg)
- E. public static final void main(String []a)
- F. public static void main(String[] data)

53. Given the application below and the choices available, which lines must all be removed to allow the code to compile? (Choose three.)

```
1: package year;
2: public class Seasons {
3:     public static void main(String[] time) {
4:         final long winter = 10;
5:         final byte season = 2;
6:         int fall = 4;
7:         final short summer = 3;
8:         switch(season) {
9:             case 1:
10:                 case winter: System.out.print("winter");
11:                 default:
12:                     case fall: System.out.print("fall");
13:                     case summer: System.out.print("summer");
14:                     default:
15:                 }
16:     }
17: }
```

- A.** Line 8
- B.** Line 9
- C.** Line 10
- D.** Line 11
- E.** Line 12
- F.** Line 13

- 54.** Given the application below, which lines do not compile? (Choose three.)

```
package furryfriends;
interface Friend {
    protected String getName(); // h1
}
class Cat implements Friend {
    String getName() { // h2
        return "Kitty";
    }
}
public class Dog implements Friend {
    String getName() throws RuntimeException { // h3
        return "Doggy";
    }
    public static void main(String[] adoption) {
        Friend friend = new Dog(); // h4
        System.out.print(((Cat)friend).getName()); // h5
        System.out.print(((Dog)null).getName()); // h6
    }
}
```

- A.** Line h1
- B.** Line h2
- C.** Line h3
- D.** Line h4
- E.** Line h5
- F.** Line h6

- 55.** Which of the following are unchecked exceptions? (Choose three.)

- A.** FileNotFoundException
- B.** ArithmeticException
- C.** IOException
- D.** Exception
- E.** IllegalArgumentException
- F.** RuntimeException

56. What is the result of compiling and executing the following application?

```
package ranch;
public class Cowboy {
    private int space = 5;
    private double ship = space < 2 ? 1 : 10; // g1
    public void printMessage() {
        if(ship>1) {
            System.out.println("Goodbye");
        } if(ship<10 && space>=2) System.out.println("Hello"); // g2
        else System.out.println("See you again");
    }
    public static final void main(String... stars) {
        new Cowboy().printMessage();
    }
}
```

- A. It only prints Hello.
 - B. It only prints Goodbye.
 - C. It only prints See you again.
 - D. It does not compile because of line g1.
 - E. It does not compile because of line g2.
 - F. None of the above
57. Given the following three class declarations, which sets of access modifiers can be inserted, in order, into the blank lines below that would allow all of the classes to compile? (Choose three.)

```
package wake;
public class Alarm {
    _____ static int clock;
    _____ long getTime() {return clock;}
}

package wake;
public class Coffee {
    private boolean bringCoffee() { return new Alarm().clock<10;}
}

package sleep;
public class Snooze extends wake.Alarm {
    private boolean checkTime() { return getTime()>10;}
}
```

- A. protected and package-private (blank)
 - B. public and public
 - C. package-private (blank) and protected
 - D. protected and protected
 - E. private and public
 - F. package-private (blank) and package-private (blank)
58. Given that `FileNotFoundException` is a subclass of `IOException` and `Long` is a subclass of `Number`, what is the output of the following application?

```
package materials;

import java.io.*;

class CarbonStructure {
    protected long count;
    public abstract Number getCount() throws IOException; // q1
    public CarbonStructure(int count) { this.count = count; }
}

public class Diamond extends CarbonStructure {
    public Diamond() { super(15); }
    public Long getCount() throws FileNotFoundException { // q2
        return count;
    }
    public static void main(String[] cost) {
        try {
            final CarbonStructure ring = new Diamond(); // q3
            System.out.print(ring.getCount()); // q4
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- A. 15
- B. It does not compile because of line q1.
- C. It does not compile because of line q2.
- D. It does not compile because of line q3.
- E. It does not compile because of line q4.
- F. It compiles but throws an exception at runtime.

59. How many lines contain a compile error?

```
1: import java.time.*;
2: import java.time.format.*;
3:
4: public class HowLong {
5:     public void main(String h) {
6:         LocalDate newYears = new LocalDate(2017, 1, 1);
7:         Period period = Period.ofYears(1).ofDays(1);
8:         DateTimeFormat format = DateTimeFormat.ofPattern("MM-dd-yyyy");
9:         System.out.print(format.format(newYears.minus(period)));
10:    }
11: }
```

- A. None
- B. One
- C. Two
- D. Three
- E. Four
- F. Five

60. Which of the following statements about try-catch blocks are correct? (Choose two.)

- A. A catch block can never appear after a finally block.
- B. A try block must be followed by a catch block.
- C. A finally block can never appear after a catch block.
- D. A try block must be followed by a finally block.
- E. A try block can have zero or more catch blocks.
- F. A try block can have zero or more finally blocks.

61. What is printed by the following code snippet?

```
int fish = 1 + 2 * 5>=2 ? 4 : 2;
int mammals = 3 < 3 ? 1 : 5>=5 ? 9 : 7;
System.out.print(fish+mammals+"");
```

- A. 49
- B. 13
- C. 18
- D. 99
- E. It does not compile.

62. Which of the following statements about objects, reference types, and casting are correct? (Choose three.)

- A.** An object can be assigned to an inherited interface reference variable without an explicit cast.
- B.** The compiler can prevent all explicit casts that lead to an exception at runtime.
- C.** Casting an object to a reference variable does not modify the object in memory.
- D.** An object can be assigned to a subclass reference variable without an explicit cast.
- E.** An object can be assigned to a superclass reference variable without an explicit cast.
- F.** An implicit cast of an object to one of its inherited types can sometimes lead to a `ClassCastException` at runtime.

63. What is the output of the following when run as `java EchoFirst seed flower plant?`

```
package unix;

import java.util.*;

public class EchoFirst {

    public static void main(String[] args) {
        int result = Arrays.binarySearch(args, args[0]);
        System.out.println(result);
    }
}
```

- A.** 0
- B.** 1
- C.** 2
- D.** The code does not compile.
- E.** The code compiles but throws an exception at runtime.
- F.** The output is not guaranteed.

64. How many objects are eligible for garbage collection at the end of the `main()` method?

```
package store;
public class Shoes {

    static String shoe1 = new String("sandal");
    static String shoe2 = new String("flip flop");

    public static void shopping() {
        String shoe3 = new String("croc");
    }
}
```

```
        shoe2 = shoe1;
        shoe1 = shoe3;
    }

    public static void main(String... args) {
        shopping();
    }
}
```

- A.** None
 - B.** One
 - C.** Two
 - D.** Three
 - E.** The code does not compile.
- 65.** Fill in the blanks: The _____ keyword is used in method declarations, the _____ keyword is used to guarantee a statement will execute even if an exception is thrown, and the _____ keyword is used to throw an exception to the surrounding process.
- A.** throw, finally, throws
 - B.** throws, catch, throw
 - C.** catch, finally, throw
 - D.** finally, catch, throw
 - E.** throws, finally, throw
- 66.** Which statements best describe the result of this code? (Choose two.)

```
package nyc;
public class TouristBus {

    public static void main(String... args) {
        String[] nycTourLoops = new String[] { "Downtown", "Uptown", "Brooklyn" };
        String[] times = new String[] { "Day", "Night" };
        for (int i = 0, j = 0; i < nycTourLoops.length; i++, j++)
            System.out.println(nycTourLoops[i] + " " + times[j]);
    }
}
```

- A.** The println causes one line of output.
- B.** The println causes two lines of output.
- C.** The println causes three lines of output.
- D.** The code terminates successfully.
- E.** The code throws an exception at runtime.

67. Fill in the blanks: Because of _____, it is possible to _____ a method, which allows Java to support _____.

- A.** abstract methods, override, inheritance
- B.** concrete methods, overload, inheritance
- C.** virtual methods, overload, interfaces
- D.** inheritance, abstract, polymorphism
- E.** virtual methods, override, polymorphism.

68. What is the result of the following?

```
package calendar;
public class Seasons {

    public static void seasons(String... names) {
        int l = names[1].length();           // s1
        System.out.println(names[l]);       // s2
    }

    public static void main(String[] args) {
        seasons("Summer", "Fall", "Winter", "Spring");
    }
}
```

- A.** Fall
- B.** Spring
- C.** The code does not compile.
- D.** The code throws an exception on line s1.
- E.** The code throws an exception on line s2.

69. How many lines of the following application contain compilation errors?

```
1: package percussion;
2:
3: interface MakesNoise {}
4: abstract class Instrument implements MakesNoise {
5:     public Instrument(int beats) {}
6:     public void play() {}
7: }
8: public class Drum extends Instrument {
9:     public void play(int count) {}
10:    public void concert() {
11:        super.play(5);
```

```
12:      }
13:      public static void main(String[] beats) {
14:          MakesNoise mn = new Drum();
15:          mn.concert();
16:      }
17: }
```

- A.** None. The code compiles and runs without issue.
B. One
C. Two
D. Three
E. Four
- 70.** What is the output of the following application?

```
package fly;
public class Helicopter {
    public int adjustPropellers(int length, String[] type) {
        length++;
        type[0] = "LONG";
        return length;
    }
    public static void main(String[] climb) {
        final Helicopter h = new Helicopter();
        int length = 5;
        String[] type = new String[1];
        length = h.adjustPropellers(length, type);
        System.out.print(length+","+type[0]);
    }
}
```

- A.** 5, LONG
B. 6, LONG
C. 5, null
D. 6, null
E. The code does not compile.
F. The code compiles but throws an exception at runtime.
- 71.** How many lines of the following application do not compile?

```
package castles;
class OpenDoorException extends Exception {}
class CableSnapException extends OpenDoorException {}
```

```
public class Palace {  
    public void openDrawbridge() throws Exception {  
        try {  
            throw new Exception("Problem");  
        } catch (OpenDoorException e) {  
            throw new OpenDoorException();  
        } catch (CableSnapException ex) {  
            try {  
                throw new OpenDoorException();  
            } catch (Exception ex) {  
            } finally {  
                System.out.println("Almost done");  
            }  
        } finally {  
            throw new RuntimeException("Unending problem");  
        }  
    }  
    public static void main(String[] moat) throws IllegalArgumentException {  
        new Palace().openDrawbridge();  
    }  
}
```

- A. None. The code compiles and produces a stack trace at runtime.
- B. One
- C. Two
- D. Three
- E. Four
- F. Five
72. Choose the best answer: _____ and _____ are two properties that go hand in hand to improve class design by structuring a class with related attributes and actions while protecting the underlying data from access by other classes.
- A. Optimization and platform independence
- B. Platform independence and encapsulation
- C. Platform independence and inheritance
- D. Object orientation and encapsulation
- E. Inheritance and polymorphism
73. What is the output of the following?

```
string bike1 = "speedy";  
string bike2 = new String("speedy");
```

```
boolean test1 = bike1 == bike2;
boolean test2 = bike1.equals(bike2);
System.out.println(test1 + " " + test2);
```

- A.** false false
 - B.** false true
 - C.** true false
 - D.** true true
 - E.** The code does not compile.
 - F.** The code compiles but throws an exception at runtime.
- 74.** What is the output of the following when run as `java EchoFirst seed flower plant?`

```
package unix;

import java.util.*;

public class EchoFirst {

    public static void main(String[] args) {
        Arrays.sort(args);
        int result = Arrays.binarySearch(args, args[0]);
        System.out.println(result);
    }
}
```

- A.** 0
 - B.** 1
 - C.** 2
 - D.** The code does not compile.
 - E.** The code compiles but throws an exception at runtime.
 - F.** The output is not guaranteed.
- 75.** Which are true statements? (Choose three.)
- A.** Every do-while loop can be rewritten as a for-each loop.
 - B.** Every for-each loop can be rewritten as a do-while loop.
 - C.** Every for-each loop can be rewritten as a traditional for loop.
 - D.** Every for-each loop can be rewritten as a while loop.
 - E.** Every traditional for loop can be rewritten as a for-each loop.
 - F.** Every while loop can be rewritten as a for-each loop.

76. How many lines does this program print?

```
import java.time.*;
public class OnePlusOne {
    public static void main(String... nums) {
        LocalDate time = LocalDate.of(1, 11);
        while (time.getHour() < 1) {
            time.plusHours(1);
            System.out.println("in loop");
        }
    }
}
```

- A.** None
- B.** One
- C.** Two
- D.** This is an infinite loop.
- E.** The code does not compile.

77. How many objects are eligible for garbage collection immediately before the end of the `main()` method?

```
public class Tennis {
    public static void main(String[] game) {
        String[] balls = new String[1];
        int[] scores = new int[1];
        balls = null;
        scores = null;
    }
}
```

- A.** None
- B.** One
- C.** Two
- D.** Three
- E.** Four

78. What is the output of the following?

```
14: int count = 0;
15: LocalDate date = LocalDate.of(2017, Month.JANUARY, 1);
16: while (date.getMonth() != Month.APRIL)
17:     date = date.minusMonths(1);
18:     count++;
19: System.out.println(count);
```

- A.** 0
- B.** 1
- C.** 3
- D.** 9
- E.** This is an infinite loop.
- F.** The code does not compile.

79. How many lines of the following class do not compile?

```
1: package arctic;
2: abstract class Bear {
3:     protected int sing;
4:     protected abstract int grunt();
5:     int sing() {
6:         return sing;
7:     }
8: }
9: public class PolarBear extends Bear {
10:    int grunt() {
11:        sing() += 10;
12:        return super.grunt()+1;
13:        return 10;
14:    }
15: }
```

- A.** None, the class compiles without issue.
- B.** One
- C.** Two
- D.** Three
- E.** Four
- F.** Five

80. In which places is the `default` keyword permitted to be used? (Choose two.)

- A.** Access modifier in a class
- B.** Execution path in a `switch` statement
- C.** Method name
- D.** Modifier in an `abstract` interface method
- E.** Modifier in an interface method with a body
- F.** Variable name

OCP

PART

||



Chapter 11

Java Class Design

**THE OCP EXAM TOPICS COVERED
IN THIS PRACTICE TEST INCLUDE
THE FOLLOWING:**

✓ **Java Class Design**

- Implement encapsulation
- Implement inheritance including visibility modifiers and composition
- Implement polymorphism
- Override hashCode, equals, and toString methods from Object class
- Create and use singleton classes and immutable classes
- Develop code that uses static keyword on initialize blocks, variables, methods, and classes

1. Which answer choice can replace line 6 so the code continues to produce the same output?

```
3: List<String> rug = new ArrayList<>();  
4: rug.add("circle");  
5: rug.add("square");  
6: System.out.println(rug);
```

- A. System.out.println(rug.asString());
- B. System.out.println(rug.toString());
- C. System.out.println(rug.toString());
- D. System.out.println(rug.toString());

2. Which best describes this code?

```
class Stats {  
    private int data;  
    public int getData() {  
        return data;  
    }  
    public void setData(int data) {  
        this.data = data;  
    }  
}
```

- A. It is a singleton.
- B. It is well encapsulated.
- C. It is immutable.
- D. It is both well encapsulated and immutable.

3. What design pattern or principle ensures that there will be no more than one instance of a class?

- A. Encapsulation
- B. Immutability
- C. Singleton
- D. Static

4. What is the output of this code?

```
class Laptop extends Computer {  
    public void startup() {  
        System.out.print("laptop-");  
    }  
}  
public class Computer {
```

```
public void startup() {  
    System.out.print("computer-");  
}  
public static void main(String[] args) {  
    Computer computer = new Laptop();  
    Laptop laptop = new Laptop();  
    computer.startup();  
    laptop.startup();  
}
```

- A. computer-laptop-
B. laptop-computer-
C. laptop-laptop-
D. None of the above
5. Which method can be correctly inserted into this class to meet the contract of the `equals()` method? You may assume that `text` is not `null`.

```
class Button {  
    private String text;  
  
    public int hashCode() {  
        return text.hashCode();  
    }  
}
```

- A.
- ```
public boolean equals(Object o) {
 if (o == null) return true;
 if (! (o instanceof Button)) return false;
 return text.equals(o.text);
}
```
- B.
- ```
public boolean equals(Object o) {  
    if ( o == null ) return true;  
    Button b = (Button) o;  
    return text.equals(b.text);  
}
```

C.

```
public boolean equals(Object o) {  
    if (!(o instanceof Button)) return false;  
    return text.equals(o.text);  
}
```

D.

```
public boolean equals(Object o) {  
    if (!(o instanceof Button)) return false;  
    Button b = (Button) o;  
    return text.equals(b.text);  
}
```

6. Fill in the blanks: _____ means the state of an object cannot be changed while _____ means that it can.

- A. Immutability, mutability
- B. Rigidity, flexibility
- C. Static, instance
- D. None of the above

7. Which is the first line to fail to compile?

```
class Tool {  
    void use() { }      // r1  
}  
  
class Hammer extends Tool {  
    private void use() { } // r2  
    public void bang() { } // r3  
}
```

- A. r1
- B. r2
- C. r3
- D. None of the above

8. Which of these classes properly implement(s) the singleton pattern?

```
class ExamAnswers {  
    private static ExamAnswers instance = new ExamAnswers();  
    private List<String> answers = new ArrayList<>();
```

```
public static List<String> getAnswers() {
    return instance.answers;
}
}
class TestAnswers {
    private static TestAnswers instance = new TestAnswers();
    private List<String> answers = new ArrayList<>();
    public static TestAnswers getTestAnswers() {
        return instance;
    }
    public List<String> getAnswers() {
        return answers;
    }
}
```

- A.** ExamAnswers
 - B.** TestAnswers
 - C.** Both classes
 - D.** Neither class
- 9.** What does the following print?

```
public class Transport {
    static interface Vehicle {}
    static class Bus implements Vehicle {}

    public static void main(String[] args) {
        Bus bus = new Bus();
        boolean n = null instanceof Bus;
        boolean v = bus instanceof Vehicle;
        boolean b = bus instanceof Bus;
        System.out.println(n + " " + v + " " + b);
    }
}
```

- A.** true true true
- B.** false true true
- C.** false false false
- D.** None of the above

- 10.** What technique allows multiple variables from the same class to be shared across all instances of a class?
- A.** Encapsulation
 - B.** Immutability
 - C.** Singleton
 - D.** Static
- 11.** Which is not a requirement for a class to be immutable?
- A.** A private constructor is provided.
 - B.** Any instance variables are private.
 - C.** Methods cannot be overridden.
 - D.** There are no setter methods.
- 12.** Which statement is true about encapsulation while providing the broadest access allowed?
- A.** Variables are **public** and methods are **private**.
 - B.** Variables are **public** and methods are **public**.
 - C.** Variables are **private** and methods are **public**.
 - D.** Variables are **private** and methods are **private**.
- 13.** What does the following print?
- ```
class Laptop extends Computer {
 String type = "laptop";
}
public class Computer {
 String type = "computer";
 public static void main(String[] args) {
 Computer computer = new Laptop();
 Laptop laptop = new Laptop();
 System.out.print(computer.type + "," + laptop.type);
 }
}
```
- A.** computer,laptop
  - B.** laptop,computer
  - C.** laptop,laptop
  - D.** None of the above
- 14.** Which of these classes is/are immutable?

```
public final class Flower {
 private final String name;
 private final List<Integer> counts;
 public Flower(String name, List<Integer> counts) {
```

```
 this.name = name;
 this.counts = counts;
 }
 public String getName() {
 return name;
 }
 public List<Integer> getCounts() {
 return counts;
 }
}

public final class Plant {
 private final String name;
 private final List<Integer> counts;
 public Plant(String name, List<Integer> counts) {
 this.name = name;
 this.counts = new ArrayList<>(counts);
 }
 public String getName() {
 return name;
 }
 public List<Integer> getCounts() {
 return new ArrayList<>(counts);
 }
}
```

A. Flower

B. Plant

C. Both classes

D. Neither class

15. Which methods compile?

```
private static int numShovels;
private int numRakes;

public int getNumShovels() {
 return numShovels;
}

public int getNumRakes() {
 return numRakes;
}
```

- A. Just `getNumRakes()`
  - B. Just `getNumShovels()`
  - C. Both methods
  - D. Neither method
- 16.** Which methods compile?
- ```
private static int numShovels;  
private int numRakes;  
  
public static int getNumShovels() {  
    return numShovels;  
}  
  
public static int getNumRakes() {  
    return numRakes;  
}
```
- A. Just `getNumRakes()`
 - B. Just `getNumShovels()`
 - C. Both methods
 - D. Neither method
- 17.** How many lines of the main method fail to compile?

```
11: static interface Vehicle {}  
12: static class Bus implements Vehicle {}  
13:  
14: public static void main(String[] args) {  
15:     Bus bus = new Bus();  
16:  
17:     System.out.println(null instanceof Bus);  
18:     System.out.println(bus instanceof Vehicle);  
19:     System.out.println(bus instanceof Bus);  
20:     System.out.println(bus instanceof ArrayList);  
21:     System.out.println(bus instanceof Collection);  
22: }
```

- A. One
- B. Two
- C. Three
- D. Four

18. Which variable declaration is the first line not to compile?

```
class Building {}
class House extends Building{}

public void convert() {
    Building b = new Building();
    House h = new House();
    Building bh = new House();
    Building p = (House) b;
    House q = (Building) h;
    Building r = (Building) bh;
    House s = (House) bh;
}
```

- A. p
- B. q
- C. r
- D. s

19. Which statement is true about the code that can fill in the blank?

```
class Sticker {
    public int hashCode() {
        return 1;
    }
    public boolean equals(Object o) {
        return _____;
    }
}
```

- A. It must return false.
- B. It must return true.
- C. It can return either true or false.
- D. None of the above.

20. What change is needed to make `Secret` well encapsulated?

```
import java.util.*;

public class Secret {

    private int number = new Random().nextInt(10);
    public boolean guess(int candidate) {
        return number == candidate;
    }
}
```

- A. Change number to use a public access modifier.
 - B. Declare a private constructor.
 - C. Remove the guess method.
 - D. None. It is already well encapsulated.
21. Which of these classes best implement(s) the singleton pattern?

```
class ExamAnswers {  
    private static ExamAnswers instance = new ExamAnswers();  
    private List<String> answers = new ArrayList<>();  
    private ExamAnswers() {}  
    public ExamAnswers getExamAnswers() {  
        return instance;  
    }  
    public List<String> getAnswers() {  
        return answers;  
    }  
}  
class TestAnswers {  
    private static TestAnswers instance = new TestAnswers();  
    private List<String> answers = new ArrayList<>();  
    private TestAnswers() {}  
    public static TestAnswers getTestAnswers() {  
        return instance;  
    }  
    public List<String> getAnswers() {  
        return answers;  
    }  
}
```

- A. ExamAnswers
- B. TestAnswers
- C. Both classes
- D. Neither class

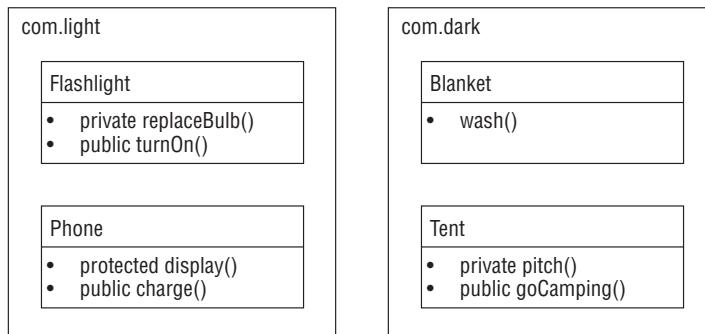
22. How many lines does the following code output?

```
public class Cars {  
    static {  
        System.out.println("static");  
    }  
    private static void drive() {
```

```
        System.out.println("fast");
    }
public static void main(String[] args) {
    drive();
    drive();
}
}
```

- A. One
- B. Two
- C. Three
- D. None of the above. The code does not compile.

23. Which is not a true statement given this diagram?



- A. Instance methods in the Blanket class can call the Flashlight class's turnOn().
 - B. Instance methods in the Flashlight class can call the Flashlight class's replaceBulb().
 - C. Instance methods in the Phone class can call the Blanket class's wash().
 - D. Instance methods in the Tent class can call the Tent class's pitch().
24. Given the diagram in the previous question, how many of the classes can call the display() method?
- A. One
 - B. Two
 - C. Three
 - D. Four
25. What does the following print?
- ```
1: class SmartWatch extends Watch {
2: private String getType() { return "smart watch"; }
3: public String getName(String suffix) {
```

```
4: return getType() + suffix;
5: }
6: }
7: public class Watch {
8: private String getType() { return "watch"; }
9: public String getName(String suffix) {
10: return getType() + suffix;
11: }
12: public static void main(String[] args) {
13: Watch watch = new Watch();
14: SmartWatch smartWatch = new SmartWatch();
15: System.out.print(watch.getName(","));
16: System.out.print(smartWatch.getName(""));
17: }
18: }
```

- A. smart watch,watch
- B. watch,smart watch
- C. watch,watch
- D. None of the above

26. What does the following print?

```
public class Transport {
 static interface Vehicle {}
 static class Bus implements Vehicle {}
 static class Van extends Bus {}

 public static void main(String[] args) {
 Bus bus = new Van();
 Van van = new Van();
 Van[] vans = new Van[0];

 boolean b = bus instanceof Vehicle;
 boolean v = van instanceof Vehicle;
 boolean a = vans instanceof Vehicle[];

 System.out.println(b + " " + v + " " + a);
 }
}
```

- A. true true true
- B. false true true
- C. true false false
- D. None of the above. The code does not compile

27. Which of the following correctly fills in the blank so this code compiles and prints true?

```
public class Button {
 private String text;
 public int hashCode() {
 return text.hashCode();
 }
 public boolean equals(Object o) {
 if (_____) return false;
 Button b = (Button) o;
 return text.equals(b.text);
 }
 public static void main(String[] args) {
 Button b1 = new Button();
 Button b2 = new Button();
 b1.text = "mickey";
 b2.text = "mickey";
 System.out.println(b1.equals(b2));
 }
}
```

- A. (o instanceof Button)
- B. (o instanceOf Button)
- C. !(o instanceof Button)
- D. !(o instanceOf Button)

28. Which is the first line to fail to compile?

```
class Tool {
 void use() { } // r1
}

class Hammer extends Tool {
 private void use(String s) { } // r2
 public void bang() { } // r3
}
```

- A. r1
- B. r2
- C. r3
- D. None of the above

**29.** What is lazy instantiation?

- A. A technique that can be used in an immutable class to wait until the first use to create the object
- B. A technique that can be used in a singleton to wait until the first use to create the object
- C. A technique that can be used in an immutable class to save memory when creating the object
- D. A technique that can be used in a singleton to save memory when creating the object

**30.** Which variable declaration is the first line not to compile?

```
30: class Building {}
31: class House extends Building{}
32:
33: public void convert() {
34: Building b = new Building();
35: House h = new House();
36: Building bh = new House();
37: House p = (House) b;
38: House q = (House) h;
39: House r = (House) bh;
40: }
```

- A. p
- B. q
- C. r
- D. None of the above

**31.** Which statement about encapsulation is not true?

- A. Encapsulation allows putting extra logic in the getter and setter methods.
- B. Encapsulation can use immutable instance variables in the implementation.
- C. Encapsulation causes two classes to be more tightly tied together.
- D. Encapsulation makes it easier to change the instance variables in the future.

**32.** Which of these classes is/are immutable?

```
public class Flower {
 private final String name;
```

```
private final List<Integer> counts;
public Flower(String name, List<Integer> counts) {
 this.name = name;
 this.counts = new ArrayList<>(counts);
}
public final String getName() {
 return name;
}
public final List<Integer> getCounts() {
 return new ArrayList<>(counts);
}
}

public class Plant {
 private final String name;
 private final List<Integer> counts;
 public Plant(String name, List<Integer> counts) {
 this.name = name;
 this.counts = new ArrayList<>(counts);
 }
 public String getName() {
 return name;
 }
 public List<Integer> getCounts() {
 return new ArrayList<>(counts);
 }
}
```

- A.** Flower
- B.** Plant
- C.** Both classes
- D.** Neither class

**33.** How many lines does the following code output?

```
public class Cars {
 private static void drive() {
 static {
 System.out.println("static");
 }
 System.out.println("fast");
```

```
 }
 public static void main(String[] args) {
 drive();
 drive();
 }
}
```

- A.** One  
**B.** Two  
**C.** Three  
**D.** None of the above. The code does not compile.
- 34.** How many of the following pairs of values can fill in the blanks to comply with the contract of the `hashCode()` and `equals()` methods?

```
class Sticker {
 public int hashCode() {
 return _____;
 }
 public boolean equals(Object o) {
 return _____;
 }
}
```

- I.** 1, false  
**II.** 1, true  
**III.** new Random().nextInt(), false  
**IV.** new Random().nextInt(), true
- A.** None  
**B.** One  
**C.** Two  
**D.** Three
- 35.** How do you change the value of an instance variable in an immutable class?
- A.** Call the setter method.  
**B.** Remove the final modifier and set the instance variable directly.  
**C.** Use a method other than Option A or B.  
**D.** You can't.

**36.** Which technique or pattern requires instance variables to implement?

- A.** Is-a
- B.** Object composition
- C.** Singleton
- D.** None of the above

**37.** How many lines of output does the following generate?

```
public class HowMany {
 static {
 System.out.println("any");
 }
 {
 System.out.println("more");
 }
 public static void main(String[] args) {
 new HowMany();
 new HowMany();
 }
}
```

- A.** Two
- B.** Three
- C.** Four
- D.** None of the above. The code does not compile.

**38.** Which is the first line to fail to compile?

```
class Tool {
 default void use() { } // r1
}
```

```
class Hammer extends Tool {
 public void use() { } // r2
 public void bang() { } // r3
}
```

- A.** r1
- B.** r2
- C.** r3
- D.** None of the above

**39.** Which variable declaration is the first line to throw a `ClassCastException` at runtime?

```
class Building {}
class House extends Building{}

public void convert() {
 Building b = new Building();
 House h = new House();
 Building bh = new House();
 House p = (House) b;
 House q = (House) h;
 House r = (House) bh;
}
```

- A.** p
- B.** q
- C.** r
- D.** None of the above

**40.** Which of the following values can fill in the blank for the class to be correctly implemented?

```
class Sticker {
 public int hashCode(Object o) {
 return _____;
 }
 public boolean equals(Object o) {
 return true;
 }
}
```

- I.** -1
  - II.** 5
  - III.** `new Random().nextInt()`
- A.** I
  - B.** I and II
  - C.** I, II, and III
  - D.** I and III

# Chapter 12



# Advanced Java Class Design

---

**THE OCP EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Advanced Java Class**

- Develop code that uses abstract classes and methods
- Develop code that uses the final keyword
- Create inner classes including static inner class, local class, nested class, and anonymous inner class
- Use enumerated types including methods, and constructors in an enum type
- Develop code that declares, implements and/or extends interfaces and use the @Override annotation
- Create and use Lambda expressions

1. Which of the following is required for all valid lambda expressions?

- A. ()
- B. ->
- C. {}
- D. Parameter data type(s)

2. What is the output of the following application?

```
package holiday;
enum DaysOff {
 Thanksgiving, PresidentsDay, ValentinesDay
}
public class Vacation {
 public static void main(String... unused) {
 final DaysOff input = DaysOff.Thanksgiving;
 switch(input) {
 default:
 case DaysOff.ValentinesDay:
 System.out.print("1");
 case DaysOff.PresidentsDay:
 System.out.print("2");
 }
 }
}
```

- A. 1
- B. 2
- C. 12
- D. None of the above

3. Fill in the blanks: A functional interface must contain or inherit \_\_\_\_\_ and may optionally include \_\_\_\_\_.

- A. at least one abstract method, the @Override annotation
- B. exactly one method, static methods
- C. exactly one abstract method, the @FunctionalInterface annotation
- D. at least one static method, at most one default method

4. Which of the following class types cannot be marked final or abstract?

- A. Static nested class
- B. Local inner class
- C. Anonymous inner class
- D. Member inner class

5. Which of the following is a valid lambda expression?
- A. `r -> {return 1==2}`
  - B. `(q) -> true`
  - C. `(x,y) -> {int test; return test>0;}`
  - D. `a,b -> true`
6. Which of the following properties of an enum can be marked abstract?
- A. The enum class definition
  - B. An enum method
  - C. An enum value
  - D. None of the above
7. What is the output of the following application?

```
package world;
public class Matrix {
 private int level = 1;
 class Deep {
 private int level = 2;
 class Deeper {
 private int level = 5;
 public void printReality() {
 System.out.print(level);
 System.out.print(" "+Matrix.Deep.this.level);
 System.out.print(" "+Deep.this.level);
 }
 }
 }
 public static void main(String[] bots) {
 Matrix.Deep.Deeper simulation = new Matrix().new Deep().new Deeper();
 simulation.printReality();
 }
}
A. 1 1 2
B. 5 2 2
C. 5 2 1
D. The code does not compile.
```

**8.** A local inner class can access which type of local variables?

- I. final
  - II. private
  - III. effectively final
- A. I only
  - B. I and II
  - C. III only
  - D. I and III

**9.** What is the output of the following application?

```
package finance;

enum Currency {
 DOLLAR, YEN, EURO
}
abstract class Provider {
 protected Currency c = Currency.EURO;
}
public class Bank extends Provider {
 protected Currency c = Currency.DOLLAR;
 public static void main(String[] pennies) {
 int value = 0;
 switch(new Bank().c) {
 case 0:
 value--; break;
 case 1:
 value++; break;
 }
 System.out.print(value);
 }
}
```

- A. 0
- B. 1
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

- 10.** What statement best describes the notion of effectively final in Java?
- A.** A local variable that is marked `final`
  - B.** A `static` variable that is marked `final`
  - C.** A local variable that is not marked `final` but whose primitive value or object reference does not change after it is initialized
  - D.** A local variable that is not marked `final` but whose primitive value or object reference does not change after a certain point in the method

- 11.** What is the output of the following application?

```
package race;

interface Drive {
 int SPEED = 5;
 default int getSpeed() { return SPEED; }
}

interface Hover {
 int MAX_SPEED = 5;
 default int getSpeed() { return MAX_SPEED; }
}

public class Car implements Drive, Hover {
 public static void main(String[] gears) {
 class RaceCar extends Car {
 @Override public int getSpeed() { return 10; }
 };
 System.out.print(new RaceCar().getSpeed());
 }
}
```

- A.** 5
  - B.** 10
  - C.** The code does not compile due to the definition of Racecar.
  - D.** The code does not compile for some other reason.
- 12.** Fill in the blanks: It is possible to extend an \_\_\_\_\_ but not an \_\_\_\_\_.
- A.** interface, abstract class
  - B.** abstract class, enum
  - C.** enum, interface
  - D.** abstract class, interface

13. Which of the following results is not a possible output of this program?

```
package sea;
enum Direction { NORTH, SOUTH, EAST, WEST; };
public class Ship {
 public static void main(String[] compass) {
 System.out.print(Direction.valueOf(compass[0]));
 }
}
```

- A. WEST is printed.
  - B. south is printed.
  - C. An `ArrayIndexOutOfBoundsException` is thrown at runtime.
  - D. An `IllegalArgumentException` is thrown at runtime.
14. Which of the following is not an advantage of using enumerated types in Java?
- A. Ensure consistency of data across an application.
  - B. Offer ability to create new enumerated values at runtime.
  - C. Provide access to fixed constants whose value does not change during the course of the application.
  - D. Support cases where a value can only take one of a limited number of options.
15. Given the following enum declaration, how many lines contain compilation errors?
- ```
package rainbow;
enum Light {}
public enum Color extends Light {
    RED, BLUE, ORANGE, GREEN
    protected Color() {}
}
```
- A. None, the code compiles as is.
 - B. One
 - C. Two
 - D. Three
16. Which of the following cannot include a `static` method in its definition?
- A. Abstract class
 - B. Static nested class
 - C. Interface
 - D. Local inner class

17. What is the output of the following application?

```
package ai;

interface Pump {
    void pump(double psi);
}

interface Bend extends Pump {
    void bend(double tensileStrength);
}

public class Robot {
    public static final void apply(Bend instruction, double input) { // r1
        instruction.bend(input);
    }
    public static void main(String... future) {
        final Robot r = new Robot();
        r.apply(x -> System.out.print(x+" bent!"), 5);
    }
}
```

- A. 5.0 bent!
- B. The code does not compile because Bend is not a functional interface.
- C. The code does not compile because of line r1.
- D. None of the above.

18. What is the best reason for applying the @Override annotation to a method?

- A. It is required to implement an interface method.
- B. It is required to override a method.
- C. The method will fail to compile if it is not actually overriding another method.
- D. There are no good reasons other than as a form of documentation.

19. What is the output of the following application?

```
package space;

public class Bottle {
    public static class Ship {
        private enum Sail { // w1
            TALL {protected int getHeight() {return 100;}},
            SHORT {protected int getHeight() {return 2;}},
            protected abstract int getHeight();
        }
        public Sail getSail() {
```

```
        return Sail.TALL;
    }
}
public static void main(String[] stars) {
    Bottle bottle = new Bottle();
    Ship q = bottle.new Ship(); // w2
    System.out.print(q.getSail());
}
}
```

- A.** TALL
- B.** The code does not compile because of line w1.
- C.** The code does not compile because of line w2.
- D.** The code compiles but the application does not produce any output at runtime.
- 20.** Which of the following is not a valid lambda expression?
- A.** (Integer j, k) -> 5
- B.** (p,q) -> p+q
- C.** (Integer x, Integer y) -> x*y
- D.** (left,right) -> {return "null";}
- 21.** What is the output of the following application?
- ```
1: package fruit;
2:
3: interface Edible { void eat(); }
4: public class ApplePicking {
5: public static void main(String[] food) {
6: Edible apple = new Edible() {
7: @Override
8: void eat() {
9: System.out.print("Yummy!");
10: }
11: }
12: }
13: }
```
- A.** The application completes without printing anything.
- B.** Yummy!
- C.** One line of this application fails to compile.
- D.** Two lines of this application fail to compile.

**22.** What is the output of the following application?

```
package forest;

public class Woods {
 static class Tree {}
 public static void main(String[] leaves) {
 int water = 10+5;
 final class Oak extends Tree { // p1
 public int getWater() {
 return water; // p2
 }
 }
 System.out.print(new Oak().getWater());
 }
}
```

- A.** 15
- B.** The code does not compile because of line p1.
- C.** The code does not compile because of line p2.
- D.** None of the above

**23.** Fill in the blanks: \_\_\_\_\_ allow Java to support multiple inheritance, and anonymous inner classes can \_\_\_\_\_ of them.

- A.** Abstract classes, extend at most one
- B.** Abstract classes, extend any number
- C.** Interfaces, implement at most one
- D.** Interfaces, implement any number

**24.** What is the output of the following application?

```
package vessel;

class Problem extends Exception {}

abstract class Danger {
 protected abstract void isDanger() throws Problem;
}

public class SeriousDanger extends Danger {
 protected void isDanger() throws Exception { // m1
 throw new RuntimeException();
 }
 public static void main(String[] will) throws Throwable { // m2
 new SeriousDanger().isDanger(); // m3
 }
}
```

- A. The code does not compile because of line m1
  - B. The code does not compile because of line m2.
  - C. The code does not compile because of line m3.
  - D. The code compiles but throws an exception at runtime.
- 25.** Which of the following is not a true statement about interfaces and abstract classes?
- A. Interfaces can only extend other interfaces, while abstract classes can extend both abstract and concrete classes.
  - B. Unlike abstract classes, interfaces can be marked `final`.
  - C. Abstract classes offer support for single inheritance, while interfaces offer support for multiple inheritance.
  - D. All methods and variables in interfaces are `public`, while abstract classes can use various access modifiers for their methods and variables, including `private` in some cases.
- 26.** What is the output of the following application?
- ```
package weather;

public class Forecast {
    public enum Snow { BLIZZARD, SQUALL, FLURRY }
    public static void main(String[] modelData) {
        System.out.print(Snow.BLIZZARD.ordinal());
        System.out.print(" "+Snow.valueOf("flurry").toUpperCase().name());
    }
}
```
- A. 0 FLURRY
 - B. 1 FLURRY
 - C. The code does not compile.
 - D. The code compiles but throws an exception at runtime.
- 27.** Fill in the blank: The primary reason `default` interface methods were added to Java is to support _____.
- A. polymorphism
 - B. concrete methods in interfaces
 - C. multiple inheritance
 - D. backward compatibility
- 28.** What is the output of the following application?

```
package zoo;

public class Penguin {
```

```
private int volume = 1;
private class Chick {
    private static int volume = 3;
    void chick() {
        System.out.print("Honk(" + Penguin.this.volume + ")!");
    }
}
public static void main(String... eggs) {
    Penguin pen = new Penguin();
    final Penguin.Chick littleOne = pen.new Chick();
    littleOne.chick();
}
```

- A. Honk(1)!
 - B. Honk(3)!
 - C. The code does not compile.
 - D. The code compiles but the output cannot be determined until runtime.
29. Let's say `Dinosaur` is a class that contains a `public` member inner class called `Pterodactyl`. Given that `dino` is an instance of `Dinosaur`, how would you instantiate a new `Pterodactyl` from within a `static` method, such as `main()`?
- A. `new Pterodactyl();`
 - B. `dino.Pterodactyl();`
 - C. `Dinosaur.new Pterodactyl();`
 - D. `dino.new Pterodactyl();`
30. What is the result of compiling the following program?

```
package desert;

interface CanBurrow {
    public abstract void burrow();
}

@FunctionalInterface interface HasHardShell extends CanBurrow {} // k1
abstract class Tortoise implements HasHardShell {} // k2
    public abstract int toughness();
}

public class DesertTortoise extends Tortoise {} // k3
    public int toughness() { return 11; }
}
```

- A. The code does not compile because of line k1.
 - B. The code does not compile because of line k2.
 - C. The code does not compile because of line k3.
 - D. The code compiles without issue.
31. Which statement(s) about the following Twins class are true?

```
package clone;

interface Alex {
    default void write() {}
    static void publish() {}
    void think();
}

interface Michael {
    public default void write() {}
    public static void publish() {}
    public void think();
}

public class Twins implements Alex, Michael {
    @Override public void write() {}
    @Override public static void publish() {}
    @Override public void think() {
        System.out.print("Thinking...");
    }
}
```

- I. The class fails to compile because of the `write()` method.
 - II. The class fails to compile because of the `publish()` method.
 - III. The class fails to compile because of the `think()` method.
- A. I only
 - B. II only
 - C. I and II
 - D. II and III
32. Fill in the blanks: A(n) _____ and a(n) _____ can define static methods.
- A. abstract class, local inner class
 - B. anonymous inner class, interface
 - C. member inner class, enum
 - D. enum, static inner class

33. Which lambda expression can replace the instance of new BiologyMaterial() in the Scientist class and produce the same results under various inputted values?

```
package university;

@FunctionalInterface interface Study {
    abstract int learn(String subject, int duration);
}

class BiologyMaterial implements Study {
    @Override public int learn(String subject, int duration) {
        if(subject == null)
            return duration;
        else
            return duration+1;
    }
}

public class Scientist {
    public static void main(String[] courses) {
        final Study s = new BiologyMaterial() {};
        System.out.print(s.learn(courses[0], Integer.parseInt(courses[1])));
    }
}

A. (p,q) -> q==null ? p : p+1
B. (c,d) -> {int d=1; return c!=null ? d+1 : d;}
C. (x,y) -> {return x==null ? y : y+1;}
D. None of the above
```

34. Given the following enum declaration, how many lines contain compilation errors?

```
package myth;

public enum Proposition {
    TRUE(-10) { @Override String getNickName() { return "RIGHT"; } },
    FALSE(-10) { public String getNickName() { return "WRONG"; } },
    UNKNOWN(0) { @Override public String getNickName() { return "LOST"; } }

    private final int value;
    Proposition(int value) {
        this.value = value;
    }
    public int getValue() {
```

```
        return this.value;
    }
    protected abstract String getNickName();
}
```

- A. None. The code compiles as is.
B. One
C. Two
D. Three
35. What is the output of the following application?

```
package math;

interface AddNumbers {
    int add(int x, int y);
    static int subtract(int x, int y) { return x-y; }
    default int multiply(int x, int y) { return x*y; }
}
public class Calculator {
    protected void calculate(AddNumbers add, int a, int b) {
        System.out.print(add.add(a, b));
    }
    public static void main(String[] moreNumbers) {
        final Calculator ti = new Calculator();
        ti.calculate((k,p) -> p+k+1, 2, 5); // j1
    }
}
```

- A. 8
B. The code does not compile because `AddNumbers` is not a functional interface.
C. The code does not compile because of line `j1`.
D. None of the above
36. Given the class declaration below, what expression can be used to fill in the blank to return the `size` variable defined in the `Bottle` class, printing 14 at runtime?

```
package baby;

final public class Bottle {
    final private int size = 14;
    final protected class Insert {
```

```
private final int size = 25;
public final int getSize() {
    return _____;
}
final Insert insert = new Insert();
final public static void main(String[] feed) {
    System.out.print(new Bottle().insert.getSize());
}
}
```

- A. `Bottle.this.size`
 - B. `this.size`
 - C. `this.Bottle.size`
 - D. The code does not compile, regardless of what is placed in the blank.
37. What is the output of the following application?

```
package ocean;
abstract interface CanSwim {
    public void swim();
}
public class Turtle {
    public static void main(String[] seaweed) {
        int distance = 7;
        CanSwim seaTurtle = {
            @Override
            public void swim() {
                System.out.print(distance);
            }
        };
        seaTurtle.swim();
    }
}
```

- A. The application completes without printing anything.
- B. 7
- C. One line of this application fails to compile.
- D. Two lines of this application fail to compile.

38. What is the output of the following application?

```
package present;

interface Toy { String play(); }
public class Gift {
    public static void main(String[] matrix) {
        abstract class Robot {}
        class Transformer extends Robot implements Toy {
            public String name = "GiantRobot";
            public String play() {return "DinosaurRobot";}
        }
        Transformer prime = new Transformer () {
            public String play() {return name;} // y1
        };
        System.out.print(prime.play()+" "+name);
    }
}
```

- A.** GiantRobot GiantRobot
- B.** GiantRobot DinosaurRobot
- C.** The code does not compile because of line y1.
- D.** None of the above

39. What is the result of compiling the following program?

```
package ballroom;

class Leader {}
class Follower {}
abstract public class Dancer {
    public Leader getPartner() { return new Leader(); }
    abstract public Leader getPartner(int count); // u1
}

abstract class SwingDancer extends Dancer {
    public Follower getPartner() { // u2
        return new Follower(); // u3
    }
}
```

- A. The code does not compile because of line u1.
 - B. The code does not compile because of line u2.
 - C. The code does not compile because of line u3.
 - D. The code compiles without issue.
40. What is the output of the following application?
- ```
package prepare;
public class Ready {
 protected static int first = 2;
 private final short DEFAULT_VALUE = 10;
 private static class GetSet {
 int first = 5;
 static int second = DEFAULT_VALUE;
 }
 private GetSet go = new GetSet();
 public static void main(String[] begin) {
 Ready r = new Ready();
 System.out.print(r.go.first);
 System.out.print(", "+r.go.second);
 }
}
```
- A. 2, 5
  - B. 5, 10
  - C. The code does not compile because of the GetSet class body.
  - D. The code does not compile for another reason.



# Chapter 13



## Generics and Collections

---

**THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:**

✓ **Generics and Collections**

- Create and use a generic class
- Create and use ArrayList, TreeSet, TreeMap, and ArrayDeque objects
- Use java.util.Comparator and java.lang.Comparable interfaces
- Collections Streams and Filters
- Iterate using forEach methods of Streams and List
- Describe Stream interface and Stream pipeline
- Filter a collection by using lambda expressions
- Use method references with Streams

1. Which of the following can fill in the blank to make the code compile?

```
public class News<____> {}
```

- I. ?
  - II. News
  - III. Object
- A. None of them
- B. I
- C. II and III
- D. I, II, and III
2. Which method is available on both List and Stream implementations?
- A. filter()
  - B. forEach()
  - C. replace()
  - D. sort()
3. We are running a library. Patrons select books by name. They get at the back of the check-out line. When they get to the front, they scan the book's ISBN. The checkout system finds the book based on this number and marks the book as checked out. Of these choices, which data structures best represent the line to check out the book and the book lookup to mark it as checked out, respectively?
- A. ArrayDeque, TreeMap
  - B. ArrayDeque, TreeSet
  - C. ArrayList, TreeMap
  - D. ArrayList, TreeSet
4. Which cannot fill in the blank for this code to compile?
- ```
Collection<String> c = new _____<>();  
c.add("pen");  
c.remove("pen");  
System.out.println(c.isEmpty());
```
- A. ArrayDeque
 - B. TreeMap
 - C. TreeSet
 - D. All of these can fill in the blank.

5. Suppose we want to implement a `Comparator<String>` so that it sorts the longest strings first. You may assume there are no nulls. Which method could implement such a comparator?

A.

```
public int compare(String s1, String s2) {  
    return s1.length() - s2.length();  
}
```

B.

```
public int compare(String s1, String s2) {  
    return s2.length() - s1.length();  
}
```

C.

```
public int compare(Object obj1, object obj2) {  
    String s1 = (String) obj1;  
    String s2 = (String) obj2;  
    return s1.length() - s2.length();  
}
```

D.

```
public int compare(Object obj1, object obj2) {  
    String s1 = (String) obj1;  
    String s2 = (String) obj2;  
    return s2.length() - s1.length();  
}
```

6. Suppose we want to store `JellyBean` objects. Which of the following pairs require `JellyBean` to implement the `Comparable` interface or create a `Comparator` in order to add them to the Collection?

A. `ArrayList` and `ArrayDeque`

B. `HashMap` and `HashSet`

C. `HashMap` and `TreeMap`

D. `TreeMap` and `TreeSet`

7. What is a common reason for a stream pipeline not to run?

A. The source doesn't generate any items.

B. There are no intermediate operations.

C. The terminal operation is missing.

D. None of the above

8. We want this code to print the titles of each book twice. Why doesn't it?

```
LinkedList<String> list = new LinkedList<>();
list.add("Grapes of Wrath");
list.add("1984");

list.forEach(System.out::println);

Iterator it = list.iterator();
while (it.hasMore())
    System.out.println(it.next());
```

- A. The generic type of `Iterator` is missing.
- B. The `hasMore()` method should be changed to `hasNext()`.
- C. The iteration code needs to be moved before the `forEach()` since the stream is used up.
- D. None of the above. The code does print each book title twice.

9. What is the result of the following?

```
ArrayList<Integer> list = new ArrayList<>();
list.add(56);
list.add(56);
list.add(3);

TreeSet<Integer> set = new TreeSet<>(list);
System.out.print(set.size());
System.out.print(" ");
System.out.print(set.iterator().next());
```

- A. 2 3
- B. 2 56
- C. 3 3
- D. 3 56

10. What best describes a reduction?

- A. An intermediate operation where it filters the stream it receives
- B. An intermediate operation where it mathematically divides each element in the stream
- C. A terminal operation where a single value is generated by reading each element in the prior step in a stream pipeline

- D. A terminal operation where one element is returned from the prior step in a stream pipeline without reading all the elements

11. What is the output of the following?

```
5:  ArrayDeque<Integer> d = new ArrayDeque<>();
6:  d.offer(18);
7:  d.offer(5);
8:  d.push(13);
9:  System.out.println(d.poll() + " " + d.poll());
```

- A. 13 18
B. 18 5
C. 18 13
D. None of the above

12. What is the output of the following?

```
class Magazine {
    private String name;
    public Magazine(String name) {
        this.name = name;
    }
    public int compareTo(Magazine m) {
        return name.compareTo(m.name);
    }
    public String toString() {
        return name;
    }
}
public class Newstand {
    public static void main(String[] args) {
        Set<Magazine> set = new TreeSet<>();
        set.add(new Magazine("highlights"));
        set.add(new Magazine("Newsweek"));
        set.add(new Magazine("highlights"));
        System.out.println(set.iterator().next());
    }
}
A. highlights
B. Newsweek
```

- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

13. What is the result of the following?

```
6: List<String> list = new ArrayList<>();  
7: list.add("Monday");  
8: list.add(String::new);  
9: list.add("Tuesday");  
10: list.remove(0);  
11: System.out.println(list.get(0));
```

- A. An empty String
- B. Monday
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

14. How many lines does this code output?

```
List<String> list = new LinkedList<>();  
list.add("Archie");  
list.add("X-Men");  
  
list.stream().forEach(s -> System.out.println(s));  
list.stream().forEach(s -> System.out.println(s));
```

- A. Two
- B. Four
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

15. Which line in the `main()` method doesn't compile or points to a class that doesn't compile?

```
1: interface Comic<C> {  
2:     void draw(C c);  
3: }  
4: class ComicClass<C> implements Comic<C> {  
5:     public void draw(C c) {  
6:         System.out.println(c);  
7:     }  
8: }  
9: class SnoopyClass implements Comic<Snoopy> {  
10:    public void draw(Snoopy c) {  
11:        System.out.println(c);
```

```
12:     }
13: }
14: class SnoopyComic implements Comic<Snoopy> {
15:     public void draw(C c) {
16:         System.out.println(c);
17:     }
18: }
19: public class Snoopy {
20:     public static void main(String[] args) {
21:         Comic<Snoopy> c1 = c -> System.out.println(c);
22:         Comic<Snoopy> c2 = new ComicClass<>();
23:         Comic<Snoopy> c3 = new SnoopyClass();
24:         Comic<Snoopy> c4 = new SnoopyComic();
25:     }
26: }
```

- A.** Line 21
B. Line 22
C. Line 23
D. Line 24
- 16.** What is the output of the following?

```
Stream<String> s = Stream.of("Atlanta", "Chicago", "New York");
long count = s.filter(c -> c.startsWith("C")).count();
System.out.print(count);
```

- A.** 1
B. 2
C. The code does not compile.
D. The code compiles but throws an exception at runtime.
- 17.** Fill in the blank to make this code compile:

```
public class Truck implements Comparable<Truck> {
    private int id;
    public Truck(int id) {
        this.id = id;
    }
    @Override
    _____ {
```

```
        return id - t.id;
    }
}
```

- A. public int compare(Truck t)
 - B. public int compare(Truck t1, Truck t2)
 - C. public int compareTo(Truck t)
 - D. public int compareTo(Truck t1, Truck t2)
18. In a stream pipeline, which can return a value other than a Stream?
- A. Source
 - B. Intermediate operation
 - C. Terminal operation
 - D. None of the above
19. Rewrite this lambda using a constructor reference:
- ```
n -> new ArrayList<>(n)
```
- A. ArrayList::new;
  - B. ArrayList::new();
  - C. ArrayList::new(n);
  - D. ArrayList::new[n];
20. What is the result of the following?
- ```
Comparator<Integer> c = (x, y) -> y-x;
List<Integer> ints = Arrays.asList(3, 1, 4);
Collections.sort(ints, c);
System.out.println(Collections.binarySearch(ints, 1));
```
- A. 0
 - B. 1
 - C. The code does not compile.
 - D. The result is not defined.
21. How many lines does this code output?

```
List<String> list = new LinkedList<>();
list.add("Archie");
list.add("X-Men");

Stream<String> s = list.stream();
s.forEach(System.out::println);
s.forEach(System.out::println);
```

- A. Two
 - B. Four
 - C. The code does not compile.
 - D. The code compiles but throws an exception at runtime.
22. Which option cannot fill in the blank to print Clean socks?

```
class Wash<T> {  
    T item;  
    public void clean(T item) {  
        System.out.println("Clean " + item);  
    }  
}  
public class LaundryTime {  
    public static void main(String[] args) {  
        _____  
        wash.clean("socks");  
    }  
}
```

- A. Wash wash = new Wash();
 - B. Wash wash = new Wash<String>();
 - C. Wash<String> wash = new Wash<>();
 - D. All three can fill in the blank.
23. We want this code to print the titles of each book twice. Why doesn't it?

```
LinkedList<String> list = new LinkedList<>();  
list.add("Grapes of Wrath");  
list.add("1984");  
  
list.stream().forEach(System.out::println);  
  
Iterator it = list.iterator();  
while (it.hasNext())  
    System.out.println(it.next());
```

- A. The generic type of Iterator is missing.
- B. The hasNext() method should be changed to isNext().
- C. The iteration code needs to be moved before the forEach() since the stream is used up.
- D. None of the above. The code does print each book title twice.

24. Rewrite this lambda using a method reference:

`() -> Math.random()`

- A.** `Math.random`
- B.** `Math::random`
- C.** `Math::random()`
- D.** None of the above

25. Which operation can occur more than once in a stream pipeline?



- A.** Source
- B.** Intermediate operation
- C.** Terminal operation
- D.** None of the above

26. Which type allows inserting a `null` value?

- A.** `ArrayDeque`
- B.** `ArrayList`
- C.** `TreeSet`
- D.** All of these allow nulls.

27. Fill in the blank so this code outputs three lines:

```
List<String> list = new ArrayList<>();
list.add("Atlanta");
list.add("Chicago");
list.add("New York");

list.stream().filter(______).forEach(System.out::println);
```

- A.** `String::isEmpty`
- B.** `! String::isEmpty`
- C.** `String::! isEmpty`
- D.** None of the above

28. What is the output of the following?

```
TreeMap<String, Integer> map = new TreeMap<>();
map.put("3", 3);
map.put("three", 3);
```

```
map.put("THREE", 3);
System.out.println(map.firstKey() + " " + map.lastKey());
```

- A.** 3 three
- B.** 3 THREE
- C.** three 3
- D.** THREE 3

29. Which fills in the blank in the method signature to allow this code to compile?

```
import java.util.*;
public class ExtendingGenerics {
    private static <_____ , U> U add(T list, U element) {
        list.add(element);
        return element;
    }
    public static void main(String[] args) {
        List<String> values = new ArrayList<>();
        add(values, "duck");
        add(values, "duck");
        add(values, "goose");
        System.out.println(values);
    }
}
```

- A.** ? extends Collection<U>
- B.** ? implements Collection<U>
- C.** T extends Collection<U>
- D.** T implements Collection<U>

30. What is the result of the following?

```
List<String> list = new ArrayList<>();
list.add("Austin");
list.add("Boston");
list.add("San Francisco");

list.removeIf(a -> a.length() > 10);
System.out.println(list.size());

A. 1
B. 2
C. 3
D. None of the above
```

31. What does the following output?

```
ArrayDeque<Integer> dice = new ArrayDeque<>();
dice.offer(3);
dice.offer(2);
dice.offer(4);
System.out.print(dice.stream().filter(n -> n != 4));
```

- A.** 2
- B.** 3
- C.** The code does not compile.
- D.** None of the above

32. Which of the following cannot fill in the blank to make the code compile?

```
private void output(____ <?> x) {
    x.forEach(System.out::println);
}
```

- A.** ArrayDeque
- B.** Collection
- C.** TreeMap
- D.** None of the above

33. How many lines does this code output?

```
List<String> list = new LinkedList<>();
list.add("Archie");
list.add("X-Men");

list.stream().forEach(System.out.println);
list.stream().forEach(System.out.println);
```

- A.** Two
- B.** Four
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

34. What is the output of the following?

```
class Magazine implements Comparable<Magazine> {
    private String name;
    public Magazine(String name) {
        this.name = name;
    }
    @Override
    public int compareTo(Magazine m) {
```

```
        return name.compareTo(m.name);
    }
    @Override
    public String toString() {
        return name;
    }
}
public class Newstand {
    public static void main(String[] args) {
        Set<Magazine> set = new TreeSet<>();
        set.add(new Magazine("highlights"));
        set.add(new Magazine("Newsweek"));
        set.add(new Magazine("highlights"));
        System.out.println(set.iterator().next());
    }
}
```

- A.** highlights
- B.** Newsweek
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

- 35.** How many lines does the following code output?

```
import java.util.*;
class Blankie {
    String color;
    String getColor() {
        return color;
    }
}
public class PreSchool {
    public static void main(String[] args) {
        Blankie b1 = new Blankie();
        Blankie b2 = new Blankie();
        b1.color = "pink";
        List<Blankie> list = Arrays.asList(b1, b2);
        list.stream().filter(Blankie::getColor).forEach(System.out::println);
    }
}
```

- A. One
 - B. Two
 - C. The code does not compile.
 - D. The code compiles but throws an exception at runtime.
36. Which statement about a source in a Stream is true?
- A. The source is mandatory in a stream pipeline.
 - B. The source is only allowed to return primitives.
 - C. The source must be retrieved by calling the `stream()` method.
 - D. The source must return a finite number of elements.
37. What does the following output?
- ```
list<String> list = new ArrayList<>();
list.add("Austin");
list.add("Boston");
list.add("San Francisco");

long c = list.stream().filter(a -> a.length() > 10).count();
System.out.println(c + " " + list.size());
```
- A. 1 1
  - B. 1 3
  - C. 2 3
  - D. None of the above
38. Which options can fill in the blanks to print Cleaned 2 items?

```
import java.util.*;
class Wash<T _____ Collection> {
 T item;
 public void clean(T items) {
 System.out.println("Cleaned " + items.size() + " items");
 }
}
public class LaundryTime {
 public static void main(String[] args) {
 Wash<List> wash = new _____
 wash.clean(Arrays.asList("sock", "tie"));
 }
}
```

- A. extends, Wash<ArrayList>();
  - B. extends, Wash<List>();
  - C. super, Wash<ArrayList>();
  - D. super, Wash<List>();
39. Which of the following declares a Comparator where all objects are treated as equal?
- A. Comparator<Character> comp = (c1)-> 0;
  - B. Comparator<Character> comp = (c1)-> {0};
  - C. Comparator<Character> comp = (c1, c2)-> 0;
  - D. Comparator<Character> comp = (c1, c2)-> {0};
40. Why can't `String::charAt` be used as a method reference?
- A. Method references can only be used on static methods.
  - B. Method references can pass either the instance or the parameter from the lambda, but not both.
  - C. The `charAt()` method takes an `int` rather than `Integer` parameter.
  - D. There is no `charAt()` method in the `String` class.



# Chapter 14



## Lambda Built-in Functional Interfaces

---

**THE OCP EXAM TOPICS COVERED IN  
THIS PRACTICE TEST INCLUDE THE  
FOLLOWING:**

✓ **Lambda Built-in Functional Interfaces**

- Use the built-in interfaces included in the `java.util.function` package such as `Predicate`, `Consumer`, `Function`, and `Supplier`
- Develop code that uses primitive versions of functional interfaces
- Develop code that uses binary versions of functional interfaces
- Develop code that uses the `UnaryOperator` interface

1. Fill in the blanks: The \_\_\_\_\_ functional interface does not take any inputs, while the \_\_\_\_\_ functional interface does not return any data.
  - A. IntConsumer, LongSupplier
  - B. IntSupplier, Function
  - C. Supplier, DoubleConsumer
  - D. UnaryOperator, Consumer
2. Which functional interface takes a long value as an input argument and has an accept() method?
  - A. LongConsumer
  - B. LongFunction
  - C. LongPredicate
  - D. LongSupplier
3. What is the output of the following application?

```
package beach;
import java.util.function.*;

class Tourist {
 public Tourist(double distance) {
 this.distance = distance;
 }
 public double distance;
}
public class Lifeguard {
 private void saveLife(Predicate<Tourist> canSave, Tourist tourist) {
 System.out.print(canSave.test(tourist) ? "Saved" : "Too far"); // y1
 }
 public final static void main(String... sand) {
 new Lifeguard().saveLife(s -> s.distance<4, new Tourist(2)); // y2
 }
}
```

- A. Saved
- B. Too far
- C. The code does not compile because of line y1.
- D. The code does not compile because of line y2.

4. Which of the following statements about DoubleSupplier and Supplier<Double> is not true?

- A. Both are functional interfaces.
- B. Lambdas for both can return a double value.
- C. Lambdas for both cannot return a null value.
- D. One supports a generic type, the other does not.

5. Which functional interface, when filled into the blank, allows the class to compile?

```
package space;
import java.util.function.*;

public class Asteroid {
 public void mine(____ lambda) {
 // TODO: Apply functional interface
 }
 public static void main(String[] debris) {
 new Asteroid().mine((s,p) -> s+p);
 }
}
```

- A. BiConsumer<Integer,Double>
- B. BiFunction<Integer,Double,Double>
- C. BiFunction<Integer, Integer, Double>
- D. Function<Integer,Double>

6. Assuming the proper generic types are used, which lambda expression cannot be assigned to a ToDoubleBiFunction functional interface reference?

- A. (Integer a, Double b) -> {int c; return b;}
- B. (h,i) -> (long)h
- C. (String u, Object v) -> u.length()+v.length()
- D. (x,y) -> {int z=2; return y/z;}

7. Which of the following is not a functional interface in the java.util.function package?

- A. BiPredicate
- B. DoubleUnaryOperator
- C. ObjectDoubleConsumer
- D. ToLongFunction

8. What is the output of the following application?

```
package zoo;
import java.util.function.*;

public class TicketTaker {
 private static int AT_CAPACITY = 100;
 public int takeTicket(int currentCount, IntUnaryOperator<Integer> counter) {
 return counter.applyAsInt(currentCount);
 }
 public static void main(String...theater) {
 final TicketTaker bob = new TicketTaker();
 final int oldCount = 50;
 final int newCount = bob.takeTicket(oldCount,t -> {
 if(t>AT_CAPACITY) {
 throw new RuntimeException("Sorry, max has been reached");
 }
 return t+1;
 });
 System.out.print(newCount);
 }
}
```

- A.** 51
  - B.** The code does not compile because of lambda expression.
  - C.** The code does not compile for a different reason.
  - D.** The code compiles but prints an exception at runtime.
9. Which functional interface returns a primitive value?
- A.** BiPredicate
  - B.** CharSupplier
  - C.** LongFunction
  - D.** UnaryOperator
10. Which functional interface, when entered into the blank below, allows the class to compile?

```
package groceries;
import java.util.*;
import java.util.function.*;

public class Market {
```

```
private static void checkPrices(List<Double> prices,
 Scanner scanner) {
 prices.forEach(scanner);
}
public static void main(String[] args) {
 List<Double> prices = Arrays.asList(1.2, 6.5, 3.0);
 checkPrices(prices,
 p -> {
 String result = p < 5 ? "Correct" : "Too high";
 System.out.println(result);
 });
}
```

- A. Consumer  
B. DoubleConsumer  
C. Supplier<Double>  
D. None of the above
11. Which of the following three functional interfaces is not equivalent to the other two?
- A. BiFunction<Double, Double, Double>  
B. BinaryOperator<Double>  
C. DoubleFunction<Double>  
D. None of the above. All three are equivalent.
12. Which lambda expression can be passed to the `magic()` method?

```
package show;
import java.util.function.*;

public class Magician {
 public void magic(BinaryOperator<Long> lambda) {
 lambda.apply(3L, 7L);
 }
}
A. magic((a) -> a)
B. magic((b,w) -> (long)w.intValue())
C. magic((c,m) -> {long c=4; return c+m;})
D. magic((Integer d, Integer r) -> (Long)r+d)
```

13. What is the output of the following program?

```
package ai;
import java.util.function.*;

public class Android {
 public void wakeUp(Supplier supplier) { // d1
 supplier.get();
 }
 public static void main(String... electricSheep) {
 Android data = new Android();
 data.wakeUp(() -> System.out.print("Program started!")); // d2
 }
}
```

- A. Program started!
  - B. The code does not compile because of line d1 only.
  - C. The code does not compile because of line d2 only.
  - D. The code does not compile because of both lines d1 and d2.
14. Which statement about all UnaryOperator functional interfaces (generic and primitive) is correct?
- A. The input type must be compatible with the return type.
  - B. Some of them take multiple arguments.
  - C. They each take a generic argument.
  - D. They each return a primitive value.
15. Starting with DoubleConsumer and going downward, fill in the missing values for the table.

| Functional Interface | # Parameters |
|----------------------|--------------|
| DoubleConsumer       |              |
| IntFunction          |              |
| LongSupplier         |              |
| ObjDoubleConsumer    |              |

- A. 0, 1, 1, 1
- B. 0, 2, 1, 2
- C. 1, 1, 0, 2
- D. 1, 1, 0, 1

16. Starting with DoubleConsumer and going downward, fill in the values for the table. For the choices below, assume R is a generic type.

| Functional Interface | Return Type |
|----------------------|-------------|
| DoubleConsumer       |             |
| IntFunction          |             |
| LongSupplier         |             |
| ObjDoubleConsumer    |             |

- A. double, R, long, R
- B. R, int, long, R
- C. void, int, R, void
- D. void, R, long, void

17. Fill in the blanks: In the Collection interface, the method `removeIf()` takes a \_\_\_\_\_, while the method `forEach()` takes a \_\_\_\_\_.

- A. Function, Function
- B. Predicate, Consumer
- C. Predicate, Function
- D. Predicate, UnaryOperator

18. What is the output of the following application?

```
package nesting;
import java.util.function.*;

public class Doll {
 private int layer;
 public Doll(int layer) {
 super();
 this.layer = layer;
 }

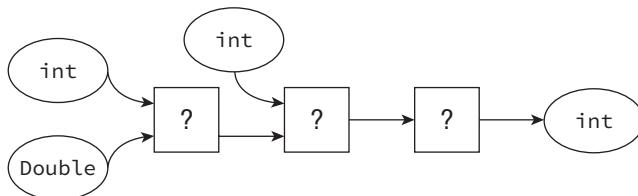
 public static void open(UnaryOperator<Doll> task, Doll doll) {
 while((doll = task.accept(doll)) != null) {
 System.out.print("X");
 }
 }
}
```

```

public static void main(String[] wood) {
 open(s -> {
 if(s.layer<=0) return null;
 else return new Doll(s.layer--);
 }, new Doll(5));
}
}

```

- A.** XXXXX
- B.** The code does not compile because of the lambda expression.
- C.** The code does not compile for a different reason.
- D.** The code compiles but produces an infinite loop at runtime.
- 19.** Which functional interface has a get() method?
- A.** Consumer
- B.** Function
- C.** Supplier
- D.** UnaryOperator
- 20.** The following diagram shows input arguments being applied to three functional interfaces of unknown type. Which three functional interfaces, inserted in order from left to right, could be used to complete the diagram?



- A.** DoubleBinaryOperator  
`ToDoubleBiFunction<Integer,Double>`  
`UnaryOperator<Integer>`
- B.** BinaryOperator<Double>  
`BiFunction<Integer,Integer,Double>`  
`UnaryOperator<Integer>`
- C.** Function<Double,Integer>  
`BiFunction<Integer,Integer,Double>`  
`DoubleToIntFunction`
- D.** BiFunction<Integer,Double,Integer>  
`BinaryOperator<Integer>`  
`IntUnaryOperator`

- 21.** Which statement about functional interfaces and lambda expressions is not true?
- A.** A lambda expression may be compatible with multiple functional interfaces.
  - B.** A lambda expression must be assigned to a functional interface when it is declared.
  - C.** A method can return a lambda expression in the form of a functional interface instance.
  - D.** The compiler uses deferred execution to skip determining whether a lambda expression compiles or not.

- 22.** Which expression is compatible with the IntSupplier functional interface?

- A.** `() -> 1<10 ? "3" : 4`
- B.** `() -> {return 1/0;}`
- C.** `() -> return 4`
- D.** `System.out::print`

- 23.** What is the output of the following application?

```
package tps;
import java.util.*;

class Boss {
 private String name;
 public Boss(String name) {
 this.name = name;
 }
 public String getName() {return name.toUpperCase();}
 public String toString() {return getName();}
}
public class Initech {
 public static void main(String[] reports) {
 final List<Boss> bosses = new ArrayList<Boss>(8);
 bosses.add(new Boss("Jenny"));
 bosses.add(new Boss("Ted"));
 bosses.add(new Boss("Grace"));
 bosses.removeIf(s -> s.equalsIgnoreCase("ted"));
 System.out.print(bosses);
 }
}
```

- A.** [JENNY, GRACE]
- B.** [tps.Boss@4218224c, tps.Boss@815f19a]
- C.** The code does not compile because of the lambda expression.
- D.** The code does not compile for a different reason.

- 24.** Which of the following method references can be passed to a method that takes `Consumer<Object>` as an argument?
- I.** `ArrayList::new`
  - II.** `String::new`
  - III.** `System.out::println`
- A.** I only
  - B.** I, II, and III
  - C.** I and III
  - D.** III only
- 25.** Which of the following is a valid functional interface in the `java.util.function` package?
- A.** `FloatPredicate`
  - B.** `ToDoubleBiFunction`
  - C.** `UnaryIntOperator`
  - D.** `TriPredicate`
- 26.** Which functional interface, when filled into the blank, prevents the class from compiling?
- ```
package morning;
import java.util.function.*;

public class Sun {
    public static void dawn(____ sunrise) {}
    public void main(String... rays) {
        dawn(s -> s+1);
    }
}
```
- A.** `DoubleUnaryOperator`
 - B.** `Function<String, String>`
 - C.** `IntToLongFunction`
 - D.** `UnaryOperator`
- 27.** Which functional interface does not have the correct number of generic arguments?
- A.** `BiFunction<T, U, R>`
 - B.** `DoubleFunction<T, R>`
 - C.** `ToDoubleFunction<T>`
 - D.** `ToIntBiFunction<T, U>`

28. Which lambda expression, when filled into the blank, allows the code to compile?

```
package ballroom;
import java.util.function.*;

public class Dance {

    public static Integer rest(BiFunction<Integer,Double,Integer> takeABreak) {
        return takeABreak.apply(3, 10.2);
    }

    public static void main(String[] participants) {
        rest(_____);
    }
}

A. (int n, double e) -> (int)(n+e)
B. (n,w,e) -> System.out::print
C. (s,w) -> 2*w
D. (s,e) -> s.intValue()+e.intValue()
```

29. Fill in the blank: _____ is the only functional interface that does not involve double, int, or long.

- A. BooleanSupplier
- B. CharPredicate
- C. FloatUnaryOperator
- D. ShortConsumer

30. What is the output of the following application?

```
package savings;
import java.util.function.*;

public class Bank {
    private int savingsInCents;
    private static class ConvertToCents {
        static DoubleToIntFunction f = p -> p*100;
    }
    public static void main(String... currency) {
        Bank creditUnion = new Bank();
        creditUnion.savingsInCents = 100;
        double deposit = 1.5;

        creditUnion.savingsInCents += ConvertToCents.f.applyAsInt(deposit); // j1
    }
}
```

```
        System.out.print(creditUnion.savingsInCents);
    }
}
```

- A.** 200
- B.** 250
- C.** The code does not compile because of line j1.
- D.** None of the above

31. Which functional interface takes a `double` value and has a `test()` method?

- A.** `DoubleConsumer`
- B.** `DoublePredicate`
- C.** `DoubleUnaryOperator`
- D.** `ToDoubleFunction`

32. Given the following class, how many lines contain compilation errors?

```
1: package showtimes;
2: import java.util.*;
3: import java.util.function.*;
4: public class FindMovie {
5:     private Function<String> printer;
6:     protected FindMovie() {
7:         printer = s -> {System.out.println(s); return s;}
8:     }
9:     void printMovies(List<String> movies) {
10:         movies.forEach(printer);
11:     }
12:     public static void main(String[] screen) {
13:         List<String> movies = new ArrayList<>();
14:         movies.add("Stream 3");
15:         movies.add("Lord of the Recursion");
16:         movies.add("Silence of the Lambdas");
17:         new FindMovie().printMovies(movies);
18:     }
19: }
```

- A.** None. The code compiles as is.
- B.** One
- C.** Two
- D.** Three

- 33.** Which lambda expression cannot be assigned to a DoubleToLongFunction functional interface?
- A.** `a -> null==null ? 1 : 2L`
 - B.** `e -> (int)(10.0*e)`
 - C.** `(double m) -> {long p = (long)m; return p;}`
 - D.** `(Double s) -> s.longValue()`
- 34.** Which of the following is not a functional interface in the `java.util.function` package?
- A.** `DoublePredicate`
 - B.** `LongUnaryOperator`
 - C.** `ShortSupplier`
 - D.** `ToIntBiFunction`
- 35.** Which functional interface, when filled into the blank, allows the class to compile?

```
package sleep;
import java.util.function.*;

class Sheep {}
public class Dream {
    int MAX_SHEEP = 10;
    int sheepCount;
    public void countSheep(____ backToSleep) {
        while(sheepCount<MAX_SHEEP) {
            // TODO: Apply lambda
            sheepCount++;
        }
    }
    public static void main(String[] dark) {
        new Dream().countSheep(System.out::println);
    }
}
```

- A.** `Consumer<Sheep>`
- B.** `Function<Sheep, void>`
- C.** `UnaryOperator<Sheep>`
- D.** `None of the above`

36. What is the output of the following application?

```
package pet;
import java.util.*;
import java.util.function.*;

public class DogSearch {
    void reduceList(List<String> names, Predicate<String> tester) {
        names.removeIf(tester);
    }
    public static void main(String[] treats) {
        int MAX_LENGTH = 2;
        DogSearch search = new DogSearch();
        List<String> names = new ArrayList<>();
        names.add("Lassie");
        names.add("Benji");
        names.add("Brian");
        MAX_LENGTH += names.size();
        search.reduceList(names, d -> d.length()>MAX_LENGTH);
        System.out.print(names.size());
    }
}
```

- A.** 2
- B.** 3
- C.** The code does not compile because of lambda expression.
- D.** The code does not compile for a different reason.

37. Which functional interface takes two values and has an apply() method?

- A.** BiConsumer
- B.** BiFunction
- C.** BiPredicate
- D.** DoubleBinaryOperator

38. Which of the following lambda expressions can be passed to a method that takes IntFunction<Integer> as an argument?

- I.** (Integer f) -> f
 - II.** (v) -> null
 - III.** s -> s
- A.** I, II, and III
 - B.** II and III only
 - C.** III only
 - D.** None of the above

- 39.** What is the output of the following application?

```
package lot;
import java.util.function.*;

public class Warehouse {
    private int quantity = 40;
    private final BooleanSupplier stock;
    {
        stock = () -> quantity>0;
    }
    public void checkInventory() {
        if(stock.get())
            System.out.print("Plenty!");
        else {
            System.out.print("On Backorder!");
        }
    }
    public static void main(String... widget) {
        final Warehouse w13 = new Warehouse();
        w13.checkInventory();
    }
}
```

- A.** Plenty!
 - B.** On Backorder!
 - C.** The code does not compile because of the `checkInventory()` method.
 - D.** The code does not compile for a different reason.
- 40.** Which of the following statements about functional interfaces is true?
- A.** It is possible to define a functional interface that returns two data types.
 - B.** It is possible to define a primitive functional interface that uses `float`, `char`, or `short`.
 - C.** It is not possible to define a functional interface that does not take any arguments nor return any value.
 - D.** None of the primitive functional interfaces include generic arguments.

Chapter 15



Java Stream API

**THE OCP EXAM TOPICS COVERED
IN THIS PRACTICE TEST INCLUDE
THE FOLLOWING:**

✓ **Java Stream API**

- Develop code to extract data from an object using peek() and map() methods including primitive versions of the map() method
- Search for data by using search methods of the Stream classes including findFirst, findAny, anyMatch, allMatch, noneMatch
- Develop code that uses the Optional class
- Develop code that uses Stream data methods and calculation methods
- Sort a collection using Stream API
- Save results to a collection using the collect method and group/partition data using the Collectors class
- Use flatMap() methods in the Stream API

1. Which of the following fills in the blank so that the code outputs one line but uses a poor practice?

```
import java.util.*;  
  
public class Cheater {  
    int count = 0;  
    public void sneak(Collection<String> coll) {  
        coll.stream()._____;  
    }  
  
    public static void main(String[] args) {  
        Cheater c = new Cheater();  
        c.sneak(Arrays.asList("weasel"));  
    }  
}
```

- A. peek(System.out::println)
- B. peek(System.out::println).findFirst()
- C. peek(r -> System.out.println(r)).findFirst()
- D. peek(r -> {count++; System.out.println(r); }).findFirst()

2. Which can fill in the blank to have the code print true?

```
Stream<Integer> stream = Stream.iterate(1, i -> i+1);  
boolean b = stream._____ (i -> i > 5);  
System.out.println(b);
```

- A. anyMatch
- B. allMatch
- C. noneMatch
- D. None of the above

3. On a DoubleStream, how many of the methods average(), count(), and sum() return an OptionalDouble?

- A. None
- B. One
- C. Two
- D. Three

4. How many of the following can fill in the blank to have the code print 44?

```
Stream<String> stream = Stream.of("base", "ball");
stream._____(s -> s.length()).forEach(System.out::print);
```

- I. map
- II. mapToInt
- III. mapToObject
- A. None
- B. One
- C. Two
- D. Three

5. What is the result of the following?

```
IntStream s = IntStream.empty();
System.out.print(s.average().getAsDouble());
```

- A. The code prints 0.
- B. The code prints 0.0.
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

6. Which of these stream pipeline operations takes a `Predicate` as a parameter and returns an `Optional`?

- A. anyMatch()
- B. filter()
- C. findAny()
- D. None of the above

7. What is the result of the following?

```
List<Double> list = new ArrayList<>();
list.add(5.4);
list.add(1.2);
Optional<Double> opt = list.stream().sorted().findFirst();
System.out.println(opt.get() + " " + list.get(0));
```

- A. 1.2 1.2
- B. 1.2 5.4
- C. 5.4 5.4
- D. None of the above

8. Fill in the blank so this code prints 8.0.

```
IntStream stream = IntStream.of(6, 10);
LongStream longs = stream.mapToLong(i -> i);
System.out.println(_____);
```

- A. longs.average().get()
- B. longs.average().getAsDouble()
- C. longs.getAverage().get()
- D. longs.getAverage().getAsDouble()

9. How many of these collectors can fill in the blank to make this code compile?

```
Stream<Character> chars = Stream.of(
    'o', 'b', 's', 't', 'a', 'c', 'l', 'e');
chars.map(c -> c).collect(Collectors. _____);
```

- I. `toArrayList()`
- II. `toList()`
- III. `toMap()`
- A. None
- B. One
- C. Two
- D. Three

10. What does the following output?

```
import java.util.*;

public class MapOfMaps {
    public static void main(String[] args) {
        Map<Integer, Integer> map = new HashMap<>();
        map.put(9, 3);
        Map<Integer, Integer> result = map.stream().map((k,v) -> (v,k));
        System.out.println(result.keySet().iterator().next());
    }
}

A. 3
B. 9
C. The code does not compile.
D. The code compiles but throws an exception at runtime.
```

11. Which of the following creates an `Optional` that returns `true` when calling `opt.isPresent()`?

- I. `Optional<String> opt = Optional.empty();`
 - II. `Optional<String> opt = Optional.of(null);`
 - III. `Optional<String> opt = Optional.ofNullable(null);`
- A. I
 - B. I and II
 - C. I and III
 - D. None of the above

12. What is the output of the following?

```
Stream<String> s = Stream.of("speak", "bark", "meow", "growl");
BinaryOperator<String> merge = (a, b) -> a;
Map<Integer, String> map = s.collect(toMap(String::length, k -> k, merge));
System.out.println(map.size() + " " + map.get(4));
```

- A. 2 bark
- B. 2 meow
- C. 4 bark
- D. None of the above

13. What is the output of the following?

```
1: package reader;
2: import java.util.stream.*;
3:
4: public class Books {
5:     public static void main(String[] args) {
6:         IntegerStream pages = IntegerStream.of(200, 300);
7:         IntegerSummaryStatistics stats = pages.summaryStatistics();
8:         long total = stats.getSum();
9:         long count = stats.getCount();
10:        System.out.println(total + "-" + count);
11:    }
12: }
```

- A. 500-0
- B. 500-2
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

14. If this method is called with `Stream.of("hi")`, how many lines are printed?

```
public static void print(Stream<String> stream) {  
    Consumer<String> print = System.out::println;  
    stream.peek(print)  
        .peek(print)  
        .map(s -> s)  
        .peek(print)  
        .forEach(print);  
}
```

- A. Three
 - B. Four
 - C. The code compiles but does not output anything.
 - D. The code does not compile.
15. What is true of the following code?

```
Stream<Character> stream = Stream.of('c', 'b', 'a');      // z1  
stream.sorted().findAny().ifPresent(System.out::println); // z2
```

- A. It is guaranteed to print the single character a.
 - B. It can print any single character of a, b, or c.
 - C. It does not compile because of line z1.
 - D. It does not compile because of line z2.
16. Suppose you have a stream pipeline where all the elements are of type `String`. Which of the following can be passed to the intermediate operation `sorted()`?
- A. `(s, t) -> s.length() - t.length()`
 - B. `String::isEmpty`
 - C. Both of these
 - D. Neither of these
17. Fill in the blanks so that both methods produce the same output for all inputs.

```
private static void longer(Optional<Boolean> opt) {  
    if (opt._____())  
        System.out.println("run: " + opt.get());  
}  
private static void shorter(Optional<Boolean> opt) {  
    opt.map(x -> "run: " + x)._____ (System.out::println);  
}
```

- A. `isNotNull, isPresent`
- B. `ifPresent, isPresent`
- C. `isPresent, forEach`
- D. `isPresent, ifPresent`

18. What is the output of this code?

```
Stream<Boolean> bools = Stream.iterate(true, b -> !b);
Map<Boolean, List<Boolean>> map = bools.limit(1)
    .collect(partitioningBy(b -> b));
System.out.println(map);
```

- A. `{true=[true]}`
- B. `{false=null, true=[true]}`
- C. `{false=[], true=[true]}`
- D. None of the above

19. What does the following output?

```
Set<String> set = new HashSet<>();
set.add("tire-");
List<String> list = new LinkedList<>();
Deque<String> queue = new ArrayDeque<>();
queue.push("wheel-");
Stream.of(set, list, queue)
    .flatMap(x -> x.stream())
    .forEach(System.out::print);
```

- A. `[tire-][wheel-]`
- B. `tire-wheel-`
- C. None of the above.
- D. The code does not compile.

20. What is the output of the following?

```
Stream<String> s = Stream.of("over the river",
    "through the woods",
    "to grandmother's house we go");
s.filter(n -> n.startsWith("t"))
    .sorted(Comparator::reverseOrder)
    .findFirst()
    .ifPresent(System.out::println);
```

- A. over the river
B. through the woods
C. to grandmother's house we go
D. None of the above
21. Which fills in the blank so the code is guaranteed to print 1?

```
Stream<Integer> stream = Stream.of(1, 2, 3);
System.out.println(stream._____);
```

- A. findAny()
B. first()
C. min()
D. None of the above
22. Which of the following can be the type for x?

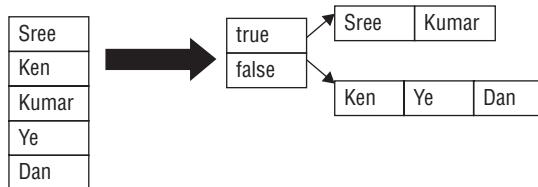
```
private static void spot(_____ x) {
    x.filter(y -> ! y.isEmpty())
    .map(y -> 8)
    .ifPresent(System.out::println);
}
```

- I. List<String>
II. Optional<Collection>
III. Optional<String>
IV. Stream<Collection>
A. I
B. IV
C. II and III
D. II and IV
23. Which can fill in the blank to have the code print true?

```
Stream<Integer> stream = Stream.iterate(1, i -> i);
boolean b = stream._____ (i -> i > 5);
System.out.println(b);
```

- A. anyMatch
B. allMatch
C. noneMatch
D. None of the above

24. What collector turns the stream at left to the Map at right?



- A. grouping()
- B. groupingBy()
- C. partitioning()
- D. partitioningBy()

25. Which fills in the blank for this code to print 667788?

```
IntStream ints = IntStream.empty();
IntStream moreInts = IntStream.of(66, 77, 88);
Stream.of(ints, moreInts)._____(x -> x).forEach(System.out::print);
```

- A. flatMap
- B. flatMapToInt
- C. map
- D. None of the above

26. Fill in the blank so this code prints 8.0. Note that it must not print `OptionalDouble[8.0]`.

```
LongStream stream = LongStream.of(6, 10);
LongSummaryStatistics stats = stream.summaryStatistics();
System.out.println(_____);
```

- A. stats.avg()
- B. stats.average()
- C. stats.average().get()
- D. stats.getAverage()

27. Which can independently fill in the blank to output `No dessert today?`

```
import java.util.*;
public class Dessert {
    public static void main(String[] yum) {
        eatDessert(Optional.of("Cupcake"));
    }
}
```

```
private static void eatDessert(Optional<String> opt) {
    System.out.println(opt._____);
}
}
```

- A. get("No dessert today")
- B. orElse("No dessert today")
- C. orElseGet(() -> "No dessert today")
- D. None of the above

28. What does the following output?

```
Stream<Character> chars = Stream.generate(() -> 'a');
chars.filter(c -> c < 'b')
    .sorted()
    .findFirst()
    .ifPresent(System.out::print);
```

- A. a
- B. The code runs successfully without any output.
- C. The code enters an infinite loop.
- D. The code compiles but throws an exception at runtime.

29. How many of the following can fill in the blank to have the code print the single digit 9?

```
LongStream stream = LongStream.of(9);
stream._____(< -> p).forEach(System.out::print);
```

- I. mapToDouble
- II. mapToInt
- III. mapToLong
- A. None
- B. One
- C. Two
- D. Three

30. Suppose you have a stream with one element and the code
`stream.xxxx.forEach(System.out::println)`. Filling in xxxx from top to bottom in the table, how many elements can be printed out?

xxxx	Number elements printed
filter()	
flatMap()	
map()	

- A. Zero or one, zero or more, exactly one
- B. Zero or one, exactly one, zero or more
- C. Zero or one, zero or more, zero or more
- D. Exactly one, zero or more, zero or more

31. What is the output of the following?

```
Stream<Character> stream = Stream.of('c', 'b', 'a');
System.out.println(stream.sorted().findFirst());
```

- A. It is guaranteed to print the single character a.
- B. It can print any single character of a, b, or c.
- C. The code does not compile.
- D. None of the above

32. What is the output of the following?

```
public class Compete {
    public static void main(String[] args) {
        Stream<Integer> is = Stream.of(8, 6, 9);
        Comparator<Integer> c = (a, b) -> a - b;
        is.sort(c).forEach(System.out::print);
    }
}
```

- A. 689
- B. 986
- C. The code does not compile
- D. The code compiles but throws an exception at runtime.

33. What is the result of the following?

```
class Ballot {
    private String name;
    private int judgeNumber;
    private int score;

    public Ballot(String name, int judgeNumber, int score) {
        this.name = name;
        this.judgeNumber = judgeNumber;
        this.score = score;
    }
    // all getters and setters
}
```

```
public class Speaking {
    public static void main(String[] args) {
        Stream<Ballot> ballots = Stream.of(
            new Ballot("Mario", 1, 10),
            new Ballot("Christina", 1, 8),
            new Ballot("Mario", 2, 9),
            new Ballot("Christina", 2, 8)
        );
        Map<String, Integer> scores = ballots.collect(
            groupingBy(Ballot::getName, summingInt(Ballot::getScore))); // w1
        System.out.println(scores.get("Mario"));
    }
}
```

- A.** The code prints 2.
 - B.** The code prints 19.
 - C.** The code does not compile due to line w1.
 - D.** The code does not compile due to a different line.
- 34.** Which can fill in the blank so this code outputs true?
- ```
import java.util.function.*;
import java.util.stream.*;

public class HideAndSeek {
 public static void main(String[] args) {
 Stream<Boolean> hide = Stream.of(true, false, true);
 boolean found = hide.filter(b -> b._____());
 System.out.println(found);
 }
}
```
- A.** Only anyMatch
  - B.** Only allMatch
  - C.** Both anyMatch and allMatch
  - D.** The code does not compile with any of these options.
- 35.** What does the following output?

```
Set<String> set = new HashSet<>();
set.add("tire-");
List<String> list = new LinkedList<>();
Deque<String> queue = new ArrayDeque<>();
queue.push("wheel-");
```

```
Stream.of(set, list, queue)
 .flatMap(x -> x)
 .forEach(System.out::println);
```

- A. [tire-][wheel-]  
B. tire-wheel-  
C. None of the above  
D. The code does not compile.
36. When working with a `Stream<String>`, which of these types can be returned from the `collect()` terminal operator by passing arguments to `Collectors.groupingBy()`?
- I. `Map<Integer, List<String>>`
  - II. `Map<Boolean, HashSet<String>>`
  - III. `List<String>`
- A. I  
B. II  
C. I and II  
D. I, II, and III
37. Which line can replace line 18 without changing the output of the program?

```
1: class Runner {
2: private int numberMinutes;
3: public Runner(int n) {
4: numberMinutes = n;
5: }
6: public int getNumberMinutes() {
7: return numberMinutes;
8: }
9: public boolean isFourMinuteMile() {
10: return numberMinutes < 4*60;
11: }
12: }
13: public class Marathon {
14: public static void main(String[] args) {
15: Stream<Runner> runners = Stream.of(new Runner(250),
16: new Runner(600), new Runner(201));
17: long count = runners
18: .filter(Runner::isFourMinuteMile)
19: .count();
20: System.out.println(count);
21: }
22: }
```

- A. `.map(Runner::isFourMinuteMile)`  
B. `.mapToBool(Runner::isFourMinuteMile)`  
    `.filter(b -> b == true)`  
C. `.mapToBoolean(Runner::isFourMinuteMile)`  
    `.filter(b -> b == true)`  
D. None of the above
38. Which method is not available on the `IntSummaryStatistics` class?  
A. `getCountAsLong()`  
B. `getMax()`  
C. `toString()`  
D. None of the above—all three methods are available.
39. Which can fill in the blank so this code outputs `Caught it`?
- ```
import java.util.*;  
public class Catch {  
    public static void main(String[] args) {  
        Optional opt = Optional.empty();  
        try {  
            apply(opt);  
        } catch (IllegalArgumentException e) {  
            System.out.println("Caught it");  
        }  
    }  
    private static void apply(Optional<Exception> opt) {  
        opt._____IllegalArgumentException::new);  
    }  
}
```
- A. `orElse`
B. `orElseGet`
C. `orElseThrow`
D. None of the above. The `main()` method does not compile.
40. A developer tries to rewrite a method that uses `flatMap()` without using that intermediate operator. Which pair of method calls shows the `withoutFlatMap()` method is not equivalent to the `withFlatMap()` method?

```
public static void main(String[] args) {  
    List<String> list = new LinkedList<>();  
    Deque<String> queue = new ArrayDeque<>();  
    queue.push("all queued up");  
    queue.push("last");  
}  
  
private static void withFlatMap(Collection<?> coll) {  
    Stream.of(coll)  
        .flatMap(x -> x.stream())  
        .forEach(System.out::print);  
    System.out.println();  
}  
  
private static void withoutFlatMap(Collection<?> coll) {  
    Stream.of(coll)  
        .filter(x -> !x.isEmpty())  
        .map(x -> x)  
        .forEach(System.out::print);  
    System.out.println();  
}  
  
A. withFlatMap(list); withoutFlatMap(list);  
B. withFlatMap(queue); withoutFlatMap(queue);  
C. Both pairs disprove the claim.  
D. Neither pair disproves this claim.
```


Chapter 16



Exceptions and Assertions

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

✓ **Exceptions and Assertions**

- Use try-catch and throw statements
- Use catch, multi-catch, and finally clauses
- Use Autoclose resources with a try-with-resources statement
- Create custom exceptions and Auto-closeable resources
- Test invariants by using assertions

1. If a `try`, a `catch`, and a `finally` statement are used together but no exception is generated, which blocks are executed and in which order?

- A. `try`
- B. `try, catch`
- C. `try, catch, finally`
- D. `try, finally`

2. Fill in the blanks: A `try` statement _____ a `catch` or a `finally` block, while a `try-with-resources` statement _____.

- A. is not required to contain, is not required to contain either
- B. is not required to contain, must contain one of them
- C. must contain, is not required to contain either
- D. must contain, must contain a `catch` block

3. What is the output of the following application?

```
package park;

class LostBallException extends Exception {}

public class Ball {
    public void toss() throws LostBallException {
        throw new ArrayStoreException();
    }

    public static void main(String[] bouncy) {
        try {
            new Ball().toss();
        } catch (Throwable e) {
            System.out.print("Caught!");
        }
    }
}
```

- A. Caught!
- B. The code does not compile because `LostBallException` is not handled or declared in the `main()` method.
- C. The code does not compile because `ArrayStoreException` is not handled or declared in the `toss()` method.
- D. The code does not compile for a different reason.

4. Which symbol(s) can be used to separate exception types in a multi-catch statement?

- I. &
- II. |
- III. ||

- A.** II only
 - B.** III only
 - C.** II and III
 - D.** I, II, and III
5. What is the result of executing the following application with assertions enabled?

```
1: package ice;
2: public class Igloo {
3:     public static void main(String[] bricks) {
4:         int flakes = 10;
5:         double assert = 7.0;
6:         assert (true : "");
7:         assert flakes++>5;
8:     }
9: }
```

- A.** It throws an `AssertionError` at runtime.
 - B.** It prints nothing at runtime.
 - C.** Exactly one line of code does not compile.
 - D.** Two lines of code do not compile.
6. Which of the following classes is a checked exception?
- A.** `java.lang.Error`
 - B.** `java.lang.IllegalStateException`
 - C.** `java.text.ParseException`
 - D.** `java.lang.RuntimeException`
7. How many constructors in `WhaleSharkException` compile in the following class?

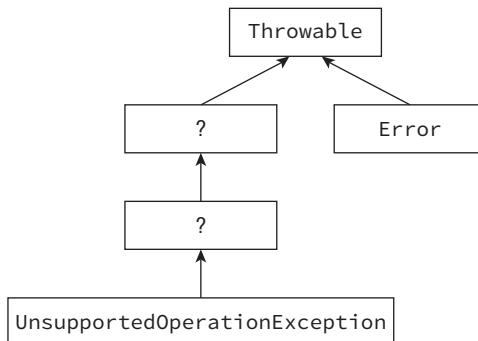
```
package friendly;
public class WhaleSharkException extends Exception {
    public WhaleSharkException() {
        super("Friendly shark!");
    }

    public WhaleSharkException(String message) {
        super(new Exception(new WhaleSharkException()));
    }

    public WhaleSharkException(Exception cause) {}
}
```

- A.** None
- B.** One
- C.** Two
- D.** Three

8. Given the following class diagram, which two classes are missing in the hierarchy starting with the bottom and going upward?



- A. IOException, Exception
 B. RuntimeException, Exception
 C. IllegalArgumentException, RuntimeException
 D. IllegalStateException, RuntimeException
9. How many lines of text does the following program print?

```

package lighting;
import java.io.IOException;
public class Light {
    public void turnOn() throws IOException {
        new IOException("Not ready");
    }
    public static void main(String[] b) throws Exception {
        try {
            new Light().turnOn();
        } catch (RuntimeException b) { // y1
            System.out.println(b);
            throw new IOException(); // y2
        } finally {
            System.out.println("complete");
        }
    }
}
  
```

- A. One
 B. Two
 C. The code does not compile because of line y1.
 D. The code does not compile because of line y2.

- 10.** Which statement, when inserted into the `main()` method of a program, guarantees an `AssertionError` will be thrown at runtime?

- A.** `assert(0,"Invalid");`
- B.** `assert 0==1;`
- C.** `assert 0==0;`
- D.** None of the above

- 11.** What is the output of the following application?

```
package paper;
import java.io.Closeable;
public class PrintCompany {
    class Printer implements Closeable { // r1
        public void print() {
            System.out.println("This just in!");
        }
        public void close() {}
    }
    public void printHeadlines() {
        try {Printer p = new Printer()} { // r2
            p.print();
        }
    }
    public static void main(String[] headlines) {
        new PrintCompany().printHeadlines(); // r3
    }
}
```

- A.** This just in!
- B.** The code does not compile because of line r1.
- C.** The code does not compile because of line r2.
- D.** The code does not compile because of line r3.

- 12.** Which statement about try-with-resources is not true?

- A.** If the `try` block and `close()` method both throw an exception, the one thrown by the `close()` method is suppressed.
- B.** A `catch` block is not required.
- C.** If more than one resource is used, the resources are closed in the order they were created.
- D.** Parentheses are used for the resource declaration section, even if more than one resource is used.

13. How many lines of text does the following program print?

```
package bee;
class SpellingException extends RuntimeException {}
public class SpellChecker {
    public final static void main(String... participants) {
        try {
            if(!"cat".equals("kat")) {
                new SpellingException();
            }
        } catch (SpellingException | NullPointerException e) {
            System.out.println("Spelling problem!");
        } catch (Exception e) {
            System.out.println("Unknown Problem!");
        } finally {
            System.out.println("Done!");
        }
    }
}
```

- A.** One
- B.** Two
- C.** Three
- D.** The code does not compile.

14. Which exception classes, when inserted into the blank in the `Problems` class, allow the code to compile?

```
package more;
class MissingMoneyException extends Exception {}
class MissingFoodException extends Exception {}
public class Problems {
    public void doIHaveAProblem() throws MissingMoneyException,
        MissingFoodException {
        System.out.println("No problems");
    }
    public static void main(String[] lots) throws _____ {
        try {
            final Problems p = new Problems();
            p.doIHaveAProblem();
        } catch (Exception e) {
```

- ```
 throw e;
 }
}
}

I. Exception
II. MissingMoneyException
III. MissingMoneyException, MissingFoodException
A. I only
B. III only
C. I and III
D. I, II, and II

15. Which statement about Closeable and AutoCloseable is true?
A. AutoCloseable extends Closeable.
B. The close() method in a class that implements AutoCloseable cannot throw an
IOException.
C. The close() method in a class that implements Closeable cannot throw an
Exception.
D. There is no difference; one was added for backward compatibility.

16. Which expression, when inserted into the blank in the following class, allows the code to
compile?
```

```
package sun;
import java.io.*;
public class Beach {
 class TideException extends Exception {}
 public void surf() throws RuntimeException {
 try {
 throw new TideException();
 } catch (_____) {}
 }
}

A. Exception a | RuntimeException f
B. IllegalStateException | TideException t
C. TideException | IOException i
D. TideException | Exception x
```

- 17.** Which statement about a multi-catch statement is true?
- A.** The exception types must be ordered from broadest to narrowest.
  - B.** The exception types must be ordered from narrowest to broadest.
  - C.** The variable of a multi-catch block with more than one exception cannot be reassigned within the block.
  - D.** The variable of a multi-catch block with one exception type cannot be reassigned within the block.

- 18.** Given the following class, how many lines contain compilation errors?

```
package move;
interface Closing {
 void close() throws Exception;
}
class Shelf implements Closing {
 public void close() throws Exception {}
}
public class Step {
 static {
 try (Shelf shelf = new Shelf()) {
 throws new IllegalArgumentException();
 } catch (Exception e) {
 } catch (IllegalArgumentException e) {
 } finally {
 shelf.close();
 }
 }
}
```

- A.** None
- B.** Two
- C.** Three
- D.** Four

- 19.** Which of the following is not true of using a try-with-resources statement?

- A.** Associated catch blocks are run before the declared resources have been closed.
- B.** It is compatible with all classes that implement the AutoCloseable interface.
- C.** It is compatible with all classes that implement the Closeable interface.
- D.** It shortens the amount of code a developer must write.

20. Assuming the following application is executed with assertions enabled, what is the result?

```
package input;
public class DataIntegrity {
 private int score;
 public DataIntegrity() {
 super();
 DataIntegrity.this.score = 5;
 }
 public static void main(String[] books) {
 final DataIntegrity johnny5 = new DataIntegrity();
 assert(johnny5.score>2) : johnny5.score++;
 assert johnny5.score>=5 : System.out.print("No input");
 System.out.print("Made it!");
 }
}
```

- A. An `AssertionError` is thrown with a message of 5.
  - B. An `AssertionError` is thrown with a message of `No input`.
  - C. `Made it!` is printed.
  - D. The code does not compile.
21. Which of the following classes is an unchecked exception?
- A. `java.io.IOException`
  - B. `java.io.NotSerializableException`
  - C. `java.sql.SQLException`
  - D. `java.util.MissingResourceException`
22. What is the result of compiling and executing the following class?

```
package wind;
public class Storm {
 public static void main(String... rain) throws Exception {
 try (final AutoCloseable weatherTracker = new AutoCloseable() {
 public void close() throws RuntimeException {}
 }) {
 System.out.println(weatherTracker.toString());
 } catch (Exception e) {
 if(weatherTracker != null) {
 weatherTracker.close();
 }
 }
 }
}
```

```
 } finally {
 System.out.println("Storm gone");
 }
 }
}
```

- A. It prints one line.
  - B. It prints two lines.
  - C. It does not compile due to an error in the declaration of the `weatherTracker` resource.
  - D. It does not compile for a different reason.
23. Which of the following is not a command that enables or disables assertions at runtime?
- A. `-di`
  - B. `-disableassertions`
  - C. `-ea`
  - D. `-enableassertions`
24. What is the output of the following application?

```
package signlanguage;
import java.io.Closeable;
class ReadSign implements Closeable {
 public void close() {}
 public String get() {return "Hello";}
}
class MakeSign implements AutoCloseable {
 public void close() {}
 public void send(String message) {
 System.out.print(message);
 }
}
public class Translate {
 public static void main(String... hands) {
 try (ReadSign r = new ReadSign();
 MakeSign w = new MakeSign()) {
 w.send(r.get());
 }
 }
}
```

- A. Hello
- B. The code does not compile because of the `ReadSign` class.
- C. The code does not compile because of the `try-with-resources` statement.
- D. None of the above

25. What is the output of the following application?

```
package what;
class FunEvent implements AutoCloseable {
 public void close() {
 System.out.print("1");
 }
}
public class Happening {
 public static void main(String... lots) {
 try (FunEvent f = new FunEvent()) {
 System.out.print("2");
 throw new ArithmeticException();
 } catch (Exception e) {
 System.out.print("3");
 } finally {
 System.out.print("4");
 }
 }
}
```

- A. 214
  - B. 2134
  - C. 2314
  - D. The code does not compile.
26. Which statement best describes how a class that implements the AutoCloseable interface should be written?
- A. The close() method is optional since the AutoCloseable interface defines a default implementation.
  - B. The close() method should avoid modifying data after it has been run once.
  - C. The close() method should not throw any exceptions.
  - D. The close() method should return a status code.
27. Which statement about the following program is correct?

```
package dogpark;
public class Fetch {
 public int play(String dogName) throws Exception {
 try {
 throw new RuntimeException(dogName);
 } catch (Exception e) {
 throw new RuntimeException(e);
 }
 }
}
```

```
 }
 public static final void main(String[] ball) throws RuntimeException {
 new Fetch().play("Webby");
 new Fetch().play("Georgette");
 }
}
```

- A.** The program prints one exception at runtime.
  - B.** The program prints two exceptions at runtime.
  - C.** The class does not compile because of the `play()` method.
  - D.** The class does not compile because of the `main()` method.
- 28.** Which of the following is not a good use of assertions?
- A.** Check method post conditions.
  - B.** Modify local variables.
  - C.** Test control flow invariants.
  - D.** Validate class invariants.
- 29.** Which statement about the following application is correct?

```
package highway;
import java.io.*;
class CarCrash extends RuntimeException {}
public class Car {
 public static void main(String[] seatbelts) throws Exception { // w1
 try {
 throw new IOException("Auto-pilot error");
 } catch (Exception | CarCrash e) { // w2
 throw e;
 } catch (Exception a) { // w3
 throw a;
 }
 }
}
```

- A.** The code does not compile because of line w1.
- B.** The code does not compile because of line w2.
- C.** The code does not compile because of line w3.
- D.** The code compiles and runs without issue.

**30.** Which statements about the following classes are true?

```
public class Dopey extends Grumpy {}
public class Grumpy extends Exception {}
public class Happy extends IOException {}
public class Sleepy extends IllegalStateException {}
public class Sneezy extends Throwable {}
```

- I.** Four of the classes are checked exceptions.
  - II.** Two of the classes are unchecked exceptions.
  - III.** None of the class declarations contain any compilation errors.
- A.** I only
  - B.** I and III
  - C.** II and III
  - D.** I, II, and III

**31.** What is the output of the following application?

```
package vortex;
class TimeException extends Exception {}
class TimeMachine implements AutoCloseable {
 int v;
 public TimeMachine(int v) {this.v = v;}
 public void close() throws Exception {
 System.out.print(v);
 }
}
public class TimeTraveler {
 public static void main(String[] twelve) {
 try (TimeMachine timeSled = new TimeMachine(1);
 TimeMachine delorean = new TimeMachine(2);
 TimeMachine tardis = new TimeMachine(3)) {
 } catch (TimeException e) {
 System.out.print(4);
 } finally {
 System.out.print(5);
 }
 }
}
```

- A.** 1235
- B.** 3215
- C.** 41235
- D.** The code does not compile.

- 32.** Which expression, when inserted into the blank in the following class, allows the code to compile?

```
package music;
public class Bells {
 class Player implements AutoCloseable {
 @Override public void close() throws RingException {}
 }
 class RingException extends Exception {
 public RingException(String message) {}
 }
 public static void main(String[] notes) throws Throwable {
 try (Player p = null) {
 throw new Exception();
 } catch (Exception e) {
 } catch (_____) {
 }
 }
}
```

- A.** Error r
- B.** IllegalStateException b
- C.** RingException p
- D.** The code does not compile regardless of the expression used.

- 33.** Given the following two variables, which assertion statement compiles successfully?

```
int age = 22;
final String name = "Josephine";
```

- A.** assert (age=2);
- B.** assert age!=age : (1<age ? "Error" : 10);
- C.** assert name.equals("") : () -> "Oops";
- D.** assert name.length()<(long)age : return "Mistake";

- 34.** Which statement about the following program is true?

```
package tag;
class MissedCallException extends Exception {}
public class Phone {
 static void makeCall() throws RuntimeException {
 throw new ArrayIndexOutOfBoundsException("Call");
 }
 public static void main(String[] messages) {
```

```
 try {
 makeCall();
 } catch (MissedCallException e) {
 throw new RuntimeException("Voicemail");
 } finally {
 throw new RuntimeException("Text");
 }
}
```

- A. An exception is printed at runtime with `Call` in the message.
  - B. An exception is printed at runtime with `Voicemail` in the message.
  - C. An exception is printed at runtime with `Text` in the message.
  - D. The code does not compile.
35. Which statement about the following program is correct?

```
package fairy;
public class Tale {
 class BearException extends RuntimeException {}
 class WolfException extends RuntimeException {}
 class DragonException extends RuntimeException {}
 public int tellStory() {
 try {} catch (BearException d) {
 d = new RuntimeException();
 throw d;
 } catch (WolfException | DragonException e) {
 e = new RuntimeException();
 throw e;
 }
 return 3;
 }
 public static void main(String... wand) throws RuntimeException{
 new Tale().tellStory();
 }
}
```

- A. The class compiles and does not print anything at runtime.
- B. The code does not compile solely due to the first `catch` block in `tellStory()`.
- C. The code does not compile solely due to the second `catch` block in `tellStory()`.
- D. The code does not compile due to errors in both `catch` blocks in `tellStory()`.

**36.** What is the output of the following application?

```
package classical;
import java.io.*;
class OutOfTuneException extends Exception {
 OutOfTuneException(String message) { super(message); }
}
public class Piano {
 public void play() throws OutOfTuneException, FileNotFoundException {
 throw new OutOfTuneException("Sour note!");
 }
 public static void main(String... keys) throws OutOfTuneException {
 final Piano piano = new Piano();
 try {
 piano.play();
 } catch (Exception e) {
 throw e;
 } finally {
 System.out.println("Song finished!");
 }
 }
}
```

- A.** Song finished!
- B.** An exception is printed with Sour note! in the stack trace.
- C.** Both of the above
- D.** None of the above

**37.** Given the following class, which command causes the class to throw an `AssertionError` at runtime?

```
public class Falcon extends Exception {
 private int parsec = 12;
 public Falcon(String name) {
 super(name);
 }
 public static void main(String[] aluminum) {
 assert new Falcon(null).parsec<12;
 }
}
```

- A.** java Falcon
- B.** java -ea -da:Falcon Falcon
- C.** java -da -ea:Falcon Falcon
- D.** The code does not compile.

38. What is the output of the following application?

```
package db;
import java.io.*;
import java.sql.*;
public class DatabaseHelper {
 static class MyDatabase implements Closeable {
 public void close() throws SQLException {
 System.out.print("2");
 }
 public void write(String data) {}
 public String read() {return null;}
 }
 public static void main(String... files) throws Exception {
 try (MyDatabase myDb = new MyDatabase()) {
 // TODO: Decide what to read/write
 } finally {
 System.out.print("1");
 }
 }
}
```

- A. 12
- B. 21
- C. The code does not compile because of the MyDatabase class.
- D. The code does not compile because of the try-with-resources statement.

39. How many lines of text does the following program print?

```
package tron;
class DiskPlayer implements AutoCloseable {
 public void close() throws Exception {}
}
public class LightCycle {
 public static void main(String... bits) {
 try (DiskPlayer john = new DiskPlayer()) {
 System.out.println("ping");
 } finally {
 System.out.println("pong");
 }
 }
}
```

```
 System.out.println("return");
 }
}
```

- A.** One
- B.** Two
- C.** Three
- D.** The code does not compile.

- 40.** Given the application below, what is the name of the class printed at line e1?

```
package canyon;
final class FallenException extends Exception {}
final class HikingGear implements AutoCloseable {
 @Override public void close() throws Exception {
 throw new FallenException();
 }
}
public class Cliff {
 public final void climb() throws Exception {
 try (HikingGear gear = new HikingGear()) {
 throw new RuntimeException();
 }
 }
 public static void main(String... rocks) {
 try {
 new Cliff().climb();
 } catch (Throwable t) {
 System.out.println(t); // e1
 }
 }
}
```

- A.** canyon.FallenException
- B.** java.lang.RuntimeException
- C.** The code does not compile.
- D.** The code compiles, but the answer cannot be determined until runtime.

# Chapter 17



## Use Java SE 8 Date/Time API

---

**THE OCP EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Use Java SE 8 Date/Time API**

- Create and manage date-based and time-based events including a combination of date and time into a single object using LocalDate, LocalTime, LocalDateTime, Instant, Period, and Duration
- Work with dates and times across timezones and manage changes resulting from daylight savings including Format date and times values
- Define and create and manage date-based and time-based events using Instant, Period, Duration, and TemporalUnit

1. What package is the LocalTime class in?
  - A. java.date
  - B. java.lang
  - C. java.time
  - D. java.util
2. How many of the classes Duration, LocalDateTime, and LocalTime have the concept of a time zone?
  - A. None
  - B. One
  - C. Two
  - D. Three
3. Which class has a getSeconds() method?
  - A. Only the Duration class
  - B. Only the Period class
  - C. Both the Duration and Period classes
  - D. Neither class
4. Which of these represents the earliest date/time?
  - A. 2017-02-15T03:00+01:00[Europe/Berlin]
  - B. 2017-02-15T04:00+02:00[Europe/Helsinki]
  - C. 2017-02-15T05:00+01:00[Europe/Warsaw]
  - D. None of the above. We have a tie.
5. Most of the United States observes daylight savings time on March 12, 2017, by moving the clocks forward an hour at 2 a.m. What does the following code output?

```
LocalDate localDate = LocalDate.of(2017, 3, 12);
LocalTime localTime = LocalTime.of(1, 0);
ZoneId zone = ZoneId.of("America/New_York");
ZonedDateTime z = ZonedDateTime.of(localDate, localTime, zone);
Duration duration = Duration.ofHours(3);
ZonedDateTime later = z.plus(duration);
System.out.println(later.getHour());
```

- A. 4
- B. 5
- C. 6
- D. None of the above

6. What does the following output?

```
int year = 1874;
int month = Month.MARCH;
int day = 24;
LocalDate date = LocalDate.of(year, month, day);
System.out.println(date.isBefore(LocalDate.now()));
```

- A. false
  - B. true
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
7. Which correctly fills in the blank to print 2017-01-15?

```
LocalDate hatDay = LocalDate.of(2017, Month.JANUARY, 15);
DateTimeFormatter f = DateTimeFormatter.ISO_DATE;
System.out.println(_____);
```

- I. f.format(hatDay)
  - II. f.formatDate(hatDay)
  - III. hatDay.format(f)
- A. I
  - B. III
  - C. I and III
  - D. II and III
8. Which of the answer choices is true given the following?

```
2017-01-07T10:00-07:00[America/Phoenix]
2017-01-07T08:00-08:00[America/Vancouver]
```

- A. The first date/time is one hour earlier than the second.
  - B. The first date/time is three hours earlier than the second.
  - C. The first date/time is one hour later than the second.
  - D. The first date/time is three hours later than the second.
9. Given that daylight savings time starts on March 12, 2017, at 2 a.m. and clocks jump from 1:59 a.m. to 03:00 a.m., which of the following can fill in the blank so the code doesn't throw an exception?

```
LocalDate localDate = LocalDate.of(2017, 3, 12);
LocalTime localTime = LocalTime.of(_____);
ZoneId zone = ZoneId.of("America/New_York");
ZonedDateTime z = ZonedDateTime.of(localDate, localTime, zone);
```

- A. 2, 0  
B. 3, 0  
C. Either of the above will run without throwing an exception.  
D. Both of these will cause an exception to be thrown.
10. What is the result of the following?
- ```
11: LocalDate waffleDay = LocalDate.of(2017, Month.MARCH, 25);
12: Period period = Period.of(1, 6, 3);
13: LocalDate later = waffleDay.plus(period);
14: later.plusDays(1);
15: LocalDate thisOne = LocalDate.of(2018, Month.SEPTEMBER, 28);
16: LocalDate thatOne = LocalDate.of(2018, Month.SEPTEMBER, 29);
17: System.out.println(later.isBefore(thisOne) + " "
18:     + later.isBefore(thatOne));
```
- A. false false
B. false true
C. true true
D. The code does not compile.
11. What is a possible result of the following?
- ```
LocalDate montyPythonDay = LocalDate.of(2017, Month.MAY, 10);
LocalDate aprilFools = LocalDate.of(2018, Month.APRIL, 1);
Duration duration = Duration.ofDays(1);
LocalDate result = montyPythonDay.minus(duration);
System.out.println(result + " " + aprilFools.isBefore(result));
```
- A. 2017-05-09 false  
B. 2017-05-09 true  
C. The code does not compile.  
D. None of the above
12. What is the result of running this code?
- ```
12: LocalDate pieDay = LocalDate.of(2017, Month.JANUARY, 23);
13: LocalTime midnight = LocalTime.of(0, 0);
14: LocalDateTime pieTime = LocalDateTime.of(pieDay, midnight);
15:
16: DateTimeFormatter f = DateTimeFormatter
17:     .ofLocalizedDate(FormatStyle.SHORT);
18: f.format(pieDay);
19: f.format(pieTime);
20: f.format(midnight);
```

- A.** The code runs successfully.
 - B.** The code throws an exception on line 19.
 - C.** The code throws an exception on line 20.
 - D.** The code does not compile.
- 13.** In the United States, daylight savings time ends on November 5th, 2017 at 02:00 a.m. and we repeat the previous hour. What is the output of the following?

```
import java.time.*;  
  
public class FallBack {  
  
    public static void main(String[] args) {  
        LocalDate localDate = LocalDate.of(2017, Month.NOVEMBER, 5);  
        LocalTime localTime = LocalTime.of(1, 0);  
        ZoneId zone = ZoneId.of("America/New_York");  
        ZonedDateTime z = ZonedDateTime.of(localDate, localTime, zone);  
  
        for (int i = 0; i < 6; i++)  
            z.plusHours(1);  
  
        System.out.println(z.getHour());  
    }  
}
```

- A.** 5
 - B.** 6
 - C.** 7
 - D.** None of the above
- 14.** What format pattern would you pass to a `DateTimeFormatter` so it creates hour and minute output such as 02:33?
- A.** HH:MM
 - B.** HH:mm
 - C.** hh:MM
 - D.** hh:mm
- 15.** `LocalTime.of()` has a number of overloads. Which of the following is not one of them?
- A.** `LocalTime.of(int hour, int minute)`
 - B.** `LocalTime.of(int hour, int minute, int second)`
 - C.** `LocalTime.of(int hour, int minute, int second, int nanoOfSecond)`
 - D.** `LocalTime.of(int hour, int minute, int second, int nanoOfSecond, int picoSeconds)`

16. How many of the classes LocalDate, Period, and ZonedDateTime have a method to get the year?

- A.** None
- B.** One
- C.** Two
- D.** Three

17. Which statement is not true about these two variables?

```
Duration duration = Duration.ofDays(1);  
Period period = Period.ofDays(1);
```

- A.** Both output the same value when calling `toString()`.
- B.** The Duration object compiles because durations are for smaller units of time.
- C.** The Period object compiles because periods are for larger units of time.
- D.** None of the above

18. What is a possible output of this code?

```
LocalTime time = LocalTime.of(1,2,3,4);  
System.out.println(time);
```

- A.** 01:02:03.4
- B.** 01:02:03.000000004
- C.** 01/01/1970 01:02:03.4
- D.** 01/01/1970 01:02:03.000000004

19. What does the following print?

```
import java.time.*;  
import java.time.format.*;  
  
public class PolarBear {  
    public static void main(String[] args) {  
        LocalDate polarBearDay = LocalDate.of(2017, 2, 27);  
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy dd MMM");  
        System.out.println(polarBearDay.format(formatter));  
    }  
}
```

- A.** 2017 27 Jan
- B.** 2017 27 Feb
- C.** 2017 Jan 27
- D.** 2017 Feb 27

- 20.** Which contains a constant named HOURS?
- A.** ChronoUnit
 - B.** Duration
 - C.** Instant
 - D.** Period
- 21.** The United States observes daylight savings time on March 12, 2017, by moving the clocks forward an hour at 2 a.m. What does the following code output?

```
LocalDate localDate = LocalDate.of(2017, 3, 12);
LocalTime localTime = LocalTime.of(13, 0);
ZoneId zone = ZoneId.of("America/New_York");
ZonedDateTime z = ZonedDateTime.of(localDate, localTime, zone);
Duration duration = Duration.ofHours(3);
ZonedDateTime later = z.plus(duration);
System.out.println(later.getHour());
```

- A.** 13
 - B.** 16
 - C.** 17
 - D.** None of the above
- 22.** What is a possible result of the following?

```
LocalDate montyPythonDay = LocalDate.of(2017, Month.MAY, 10);
LocalTime time = LocalTime.of(5, 40);
LocalDateTime dateTime = LocalDateTime.of(montyPythonDay, time);
Duration duration = Duration.ofDays(1);
LocalDateTime result = dateTime.minus(duration);
System.out.println(result);
```

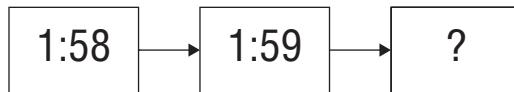
- A.** 2017-05-09
 - B.** 2017-05-09T05:40
 - C.** 2017-05-10T05:40
 - D.** None of the above
- 23.** Which correctly fills in the blank to print 2017-01-15?

```
LocalDate hatDay = LocalDate.of(2017, Month.JANUARY, 15);
DateFormatter f = DateFormatter.ISO_DATE;
System.out.println(_____);
```

- A.** f.format(hatDay)
- B.** hatDay.format(f)
- C.** Both of the above
- D.** Neither of the above

24. `LocalDateTime.of()` has a number of overloads. Which of the following is not one of them?
- A. `LocalDateTime.of(LocalDate date, LocalTime time)`
 - B. `LocalDateTime.of(LocalDate date, int hour, int minute)`
 - C. `LocalDateTime.of(int year, int month, int day, int hour, int minute)`
 - D. `LocalDateTime.of(int year, Month month, int day, int hour, int minute)`

25. In the United States, daylight savings time for 2017 starts at 2 a.m. on March 12th and ends at 2 a.m. on November 5th. Given the sequence in the following image, what time comes next on March 12th, July 4th, and November 5th, respectively?



- A. 01:00, 02:00, 01:00
 - B. 01:00, 02:00, 03:00
 - C. 03:00, 02:00, 01:00
 - D. 03:00, 02:00, 03:00
26. What is the output of the following?
- ```
LocalDate date1 = LocalDate.of(2017, Month.MARCH, 3);
LocalDate date2 = LocalDate.of(2017, Month.FEBRUARY, 31);
System.out.println(date1.equals(date2));
```
- A. false
  - B. true
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
27. Given this date/time and time zone offset, what time is it in GMT?

2017-03-09T16:00-10:00[US/Hawaii]

- A. 02:00
  - B. 04:00
  - C. 06:00
  - D. 10:00
28. What is a possible output of the following?
- ```
LocalDate trainDay = LocalDate.of(2017, 5, 13);  
LocalTime time = LocalTime.of(10, 0);  
ZoneId zone = ZoneId.of("America/Los_Angeles");
```

```
ZonedDateTime zdt = ZonedDateTime.of(trainDay, time, zone);
Instant instant = zdt.toInstant();
instant = instant.plus(1, ChronoUnit.DAYS);
System.out.println(instant);
```

- A. 2017-05-13T10:00-07:00[America/Los_Angeles]
- B. 2017-05-13T17:00:00Z
- C. 2017-05-14T10:00-07:00[America/Los_Angeles]
- D. 2017-05-14T17:00:00Z

29. What is the output of the following?

```
LocalDate date = LocalDate.of(2017, Month.JULY, 17);
LocalTime time = LocalTime.of(10, 0);
ZoneId zone = ZoneId.of("America/New_York");
ZonedDateTime iceCreamDay = ZonedDateTime.of(date, time, zone);
time = time.plusMonths(1);
System.out.println(iceCreamDay.getMonthValue());
```

- A. 6
- B. 7
- C. 8
- D. The code does not compile.

30. What does the following print?

```
import java.time.*;
import java.time.format.*;

public class PolarBear {
    public static void main(String[] args) {
        LocalDate polarBearDay = LocalDate.of(2017, 2, 27);
        DateTimeFormatter formatter = DateTimeFormatter
            .ofPattern("Holiday: yyyy dd MMM");
        System.out.println(polarBearDay.format(formatter));
    }
}
```

- A. Holiday: 2017 27 Jan
- B. Holiday: 2017 27 Feb
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

31. Which of these represents the earliest date/time?

- A.** 2017-02-15T16:00+07:00[Asia/Bangkok]
- B.** 2017-02-15T18:00+04:00[Asia/Dubai]
- C.** 2017-02-15T20:00+08:00[Asia/Kuala_Lumpur]
- D.** None of the above. We have a tie.

32. What is the result of the following?

```
11: LocalDate waffleDay = LocalDate.of(2017, Month.MARCH, 25);
12: Period period = Period.ofYears(1).ofMonths(6).ofDays(3);
13: LocalDate later = waffleDay.plus(period);
14: later.plusDays(1);
15: LocalDate thisOne = LocalDate.of(2018, Month.SEPTEMBER, 28);
16: LocalDate thatOne = LocalDate.of(2018, Month.SEPTEMBER, 29);
17: System.out.println(later.isBefore(thisOne) + " "
18:     + later.isBefore(thatOne));
```

- A.** false false
- B.** false true
- C.** true true
- D.** The code does not compile.

33. How many of the following can fill in the blank so this code compiles and prints 31?

```
LocalDate xmas = LocalDate.of(2017, 12, 25);
LocalDate blackFriday = LocalDate.of(2017, 11, 24);
long shoppingDaysLeft = _____;
System.out.println(shoppingDaysLeft);
```

- I.** blackFriday.until(xmas, ChronoUnit.DAYS)
- II.** blackFriday.until(xmas, TemporalUnit.DAYS)
- III.** ChronoUnit.DAYS.between(blackFriday, xmas)
- IV.** TemporalUnit.DAYS.between(blackFriday, xmas)
- A.** One
- B.** Two
- C.** Three
- D.** Four

34. How many of these classes cause a compiler error when filling in the blank: LocalDate, LocalDateTime, LocalTime, ZonedDateTime?

```
private static String formatMe(_____ obj) {
    DateTimeFormatter f = DateTimeFormatter.ofLocalizedTime(FormatStyle.MEDIUM);
    return f.format(obj);
}
```

- A. None
- B. One
- C. Two
- D. Three

35. What is the output of the following?

```
LocalDate date = LocalDate.of(2017, Month.JULY, 17);
LocalTime time = LocalTime.of(10, 0);
ZoneId zone = ZoneId.of("America/New_York");
ZonedDateTime iceCreamDay = ZonedDateTime.of(date, time, zone);
date = date.plusMonths(1);
System.out.println(iceCreamDay.getMonthValue());
```

- A. 6
- B. 7
- C. 8
- D. The code does not compile.

36. Which of the following can fill in the blank to make this code compile?

```
public boolean isItMyBirthday(LocalDateTime dateTime) {
    _____
    return now.getMonth() == dateTime.getMonth()
        && now.getDayOfMonth() == dateTime.getDayOfMonth();
}
```

- A. LocalDate now = LocalDate.now();
- B. LocalDate now = new LocalDate();
- C. ZonedDateTime now = ZonedDateTime.now();
- D. ZonedDateTime now = new ZonedDateTime();

37. What is the output of the following?

```
LocalDate date1 = LocalDate.of(2017, Month.MARCH, 3);
LocalDate date2 = date1.plusDays(2).minusDays(1).minusDays(1);
System.out.println(date1.equals(date2));
```

- A. false
- B. true
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

38. What is a possible output of the following?

```
LocalDate date = LocalDate.of(2017, 5, 13);
LocalTime time = LocalTime.of(10, 0);
LocalDateTime trainDay = LocalDateTime.of(date, time);
Instant instant = trainDay.toInstant();
instant = instant.plus(1, ChronoUnit.DAYS);
System.out.println(instant);
```

- A.** 2017-05-14T10:00-07:00[America/Los_Angeles]
- B.** 2017-05-14T17:00:00Z
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

39. What is the result of the following?

```
public class PiDay {
    public static void main(String[] args) {
        LocalDateTime pi = LocalDateTime.of(2017, 3, 14, 1, 59);
        DateTimeFormatter formatter = DateTimeFormatter
            .ofPattern("M.ddhhmm");
        System.out.println(formatter.format(pi));
    }
}
```

- A.** 3.140159
- B.** 59.140103
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

40. Daylight savings time ends on November 5, 2017 at 2 a.m. when we repeat the hour. Suppose we have a ZonedDateTime that outputs 2017-11-05T01:00-04:00[America/New_York] when calling `toString()`. What is a possible value of the ZonedDateTime obtained by adding an hour to this value?

- A.** 2017-11-05T01:00-04:00[America/New_York]
- B.** 2017-11-05T02:00-04:00[America/New_York]
- C.** 2017-11-05T01:00-05:00[America/New_York]
- D.** 2017-11-05T02:00-05:00[America/New_York]

Chapter 18

Java I/O Fundamentals

**THE OCP EXAM TOPICS COVERED IN
THIS PRACTICE TEST INCLUDE THE
FOLLOWING:**

✓ **Java I/O Fundamentals**

- Read and write data from the console
- Use BufferedReader, BufferedWriter, File, FileReader, FileWriter, FileInputStream, FileOutputStream, ObjectOutputStream, ObjectInputStream, and PrintWriter in the java.io package.

1. Fill in the blanks: `Writer` is _____ that related stream classes _____.
 - A. a concrete class, extend
 - B. an abstract class, extend
 - C. an interface, extend
 - D. an interface, implement
2. Which of the following methods is defined in `java.io.File`?
 - A. `createDirectory()`
 - B. `getLength()`
 - C. `listFile()`
 - D. `renameTo()`
3. Which method in `InputStream` can be used in place of calling `skip(1)`?
 - A. `jump()`
 - B. `mark()`
 - C. `read()`
 - D. `reset()`
4. Which methods are classes that implement `java.io.Serializable` required to implement?
 - A. `deserialize()`
 - B. `serial()`
 - C. `serialize()`
 - D. None of the above
5. Fill in the blanks: Given a valid `Console` instance, `reader()` returns a _____, while `writer()` returns a _____.
 - A. `PrintReader, PrintWriter`
 - B. `PrintReader, Writer`
 - C. `Reader, Writer`
 - D. `StringReader, Writer`
6. Assuming the file path referenced in the following class is accessible and able to be written, what is the output of the following program?

```
package alarm;
import java.io.*;
public class Smoke {
    public void sendAlert(File fn) {
        try(BufferedWriter w = new BufferedWriter(new FileOutputStream(fn))) {
            w.write("ALERT!");
            w.flush();
            w.write('!');
        }
    }
}
```

```
        System.out.print("1");
    } catch (IOException e) {
        System.out.print("2");
    } finally {
        System.out.print("3");
    }
}
public static void main(String[] testSignal) {
    new Smoke().sendAlert(new File("alarm.txt"));
}
}
```

- A.** 3
B. 13
C. 23
D. The code does not compile.
7. Which class is used to read information about a directory within the file system?
- A.** `java.io.File`
B. `java.io.Directories`
C. `java.io.Directory`
D. `java.io.Path`
8. Which of the following is a high-level stream class that can only be used to wrap a low-level stream?
- A.** `FileOutputStream`
B. `FileReader`
C. `ObjectInputStream`
D. `PrintWriter`
9. Assume the file `prime6.txt` exists and contains the first six prime numbers as bytes: 2, 3, 5, 7, 11, 13. What is the output of the following application?

```
package numbers;
import java.io.*;
public class PrimeReader {
    public static void main(String[] real) throws Exception {
        try (InputStream is = new FileInputStream("prime6.txt")) {
            is.skip(1);
            is.read();
            is.skip(1);
            is.read();
        }
    }
}
```

```
        is.mark(4);
        is.skip(1);
        is.reset();
        System.out.print(is.read());
    }
}
```

- A. 11
B. 13
C. The code does not compile.
D. The code compiles but throws an exception at runtime.
10. Fill in the blanks: For a given file, the absolute is the path from the _____ to the file, while the relative path is the path from the _____ to the file.
A. current directory, current working directory
B. parent directory, temporary directory
C. root directory, current working directory
D. root directory, parent directory
11. Which statement best describes the following two methods?

```
public void writeSecret1() throws IOException {
    final Writer w = new BufferedWriter(
        new FileWriter("dont.open"));
    w.write("Secret passcode");
    w.close();
}
public void writeSecret2() throws IOException {
    try(final Writer w = new BufferedWriter(
        new FileWriter("dont.open"))) {
        w.write("Secret passcode");
    }
}
```

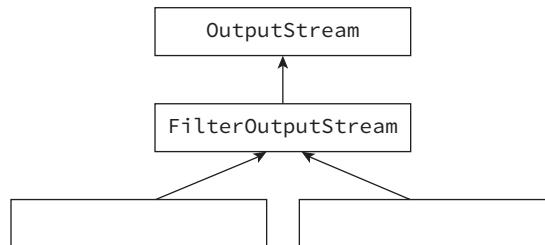
- A. Both methods compile and are equivalent to each other.
B. Neither method compiles.
C. Only one of the methods compiles.
D. The methods compile, but one method may lead to a resource leak.

12. What is the result of compiling and executing the following program?

```
package vacation;
import java.io.*;
import java.util.*;
public class Itinerary {
    private List<String> activities = new ArrayList<>();
    private static Itinerary getItinerary(String name) {
        return null;
    }
    public static void printItinerary() throws Exception {
        Console c = new Console();
        final String name = c.readLine("What is your name?");
        final Itinerary stuff = getItinerary(name);
        stuff.activities.forEach(s -> c.printf(s));
    }
    public static void main(String[] holidays) throws Exception {
        printItinerary();
    }
}
```

- A. The code does not compile.
- B. The code compiles and prints a `NullPointerException` at runtime.
- C. The code compiles but does not print anything at runtime.
- D. None of the above

13. Given the following diagram, which two classes can be placed in the blank boxes?



- A. `BufferedOutputStream` and `PrintStream`
- B. `BufferedOutputStream` and `PrintOutputStream`
- C. `ByteArrayOutputStream` and `Stream`
- D. `FileOutputStream` and `OutputStream`

14. Let's say we want to write an instance of `Cereal` to disk, having a name value of `CornLoops`. What is the value of name after this object has been read using the `ObjectInputStream`'s `readObject()` method?

```
package breakfast;
public class Cereal {
    private String name = "CocoaCookies";
    private transient int sugar;
    public Cereal() {
        super();
        this.name = "CaptainPebbles";
    }
    {
        name = "SugarPops";
    }
    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
    public int getSugar() { return sugar; }
    public void setSugar(int sugar) {
        this.sugar = sugar;
    }
}
```

- A. CaptainPebbles
 - B. CornLoops
 - C. SugarPops
 - D. None of the above
15. Which statement best describes the difference between a `Writer` and an `OutputStream` class?
- A. Only one of them can write text or character data.
 - B. Only one of them has built-in methods for writing character data.
 - C. Only one of them has a `flush()` method to force the data to be written out.
 - D. One uses a byte array to process character data more efficiently.
16. What is the output of the following application? It is safe to assume the directories referenced in the class do not exist prior to the execution of the program and that the file system is available and able to be written.

```
package job;
import java.io.*;
public class Resume {
```

```
public void resetWorkingDirectory() throws Exception {
    File f1 = new File("/templates/proofs");
    f1.mkdirs();
    File f2 = new File("/templates");
    f2.mkdir(); // k1
    new File(f2,"draft.doc").createNewFile();
    f1.delete();
    f2.delete(); // k2
}
public static void main(String... leads) {
    try {
        new Resume().resetWorkingDirectory();
    } catch (Exception e) {
        new RuntimeException(e);
    }
}
```

- A. Line k1 does not compile or triggers an exception at runtime.
- B. Line k2 does not compile or triggers an exception at runtime.
- C. The code compiles and runs without printing an exception.
- D. None of the above
17. Given the following class, three of the values ensure it runs properly on various different systems. Which value does not?

```
package magic;
import java.io.*;
public class Store {
    private final String directory;
    public Store(String directory) {
        this.directory = directory;
    }
    public File getDatabaseFolder(String file) {
        return new File(directory + _____ + file);
    }
}
```

- A. `java.io.File.separator`
- B. `new File(new String()).separatorChar`
- C. `System.getProperty("file.separator")`
- D. `System.getProperty("path.separator")`

18. How many compilation errors does the following class contain?

```
package hero;
import java.io.*;
public class Guitar {
    public void readMusic(File f) {
        try (BufferedReader r = new BufferedReader(FileReader(f))) {
            final String music = null;
            try {
                while((music = r.readLine()) != null)
                    System.out.println(music);
            } catch (IOException e) {}
            } catch (FileNotFoundException e) {
                throw new RuntimeException(e);
            } finally {}
    }
}
```

- A. None
 - B. One
 - C. Two
 - D. Three
19. What is the difference between the two `Console` methods, `format()` and `printf()`?
- A. One of them takes an optional list of arguments; the other does not.
 - B. One of them takes `String` as input; the other takes an `Object`.
 - C. There is no difference between the two methods.
 - D. Trick question! `printf()` is not defined in `Console`.
20. Let's say you want to write a lot of text data to a file in an efficient manner. Which two `java.io` stream classes are best to use?
- A. `FileOutputStream` and `BufferedOutputStream`
 - B. `FileOutputStream` and `FileBufferedWriter`
 - C. `FileWriter` and `BufferedWriter`
 - D. `ObjectOutputStream` and `BufferedWriter`
21. Assume the file referenced in the `StudentManager` class exists and contains data. Which statement about the following class is correct?

```
package school;
import java.io.*;
class Student implements Serializable {}
public class StudentManager {
```

```
public static void main(String[] grades) {  
    try(ObjectInputStream ios = new ObjectInputStream(  
        new FileInputStream(new File("C://students.data")))) {  
        Student record;  
        while((record = (Student)ios.readObject()) != null) {  
            System.out.print(record);  
        }  
    } catch (EOFException e) {  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
}
```

- A.** The code does not compile.
 - B.** The code compiles but prints an exception at runtime.
 - C.** The program runs and prints all students in the file.
 - D.** The program runs but may only print some students in the files.
- 22.** Which java.io class does not have a complementary input stream?
- A.** BufferedOutputStream
 - B.** BufferedWriter
 - C.** FileWriter
 - D.** PrintWriter
- 23.** Assuming the path /Earth does not exist within the file system, what is the output of the following program?

```
package center;  
import java.io.*;  
public class Journey {  
    public static void main(String[] dig) {  
        File file = new File("/Earth");  
        System.out.print(file.getParent()  
            +" - "  
            +file.getParent().getParent()); } }
```

- A.** / - /
- B.** / - null
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

24. Which statements about executing the following program are true?

```
package test;
import java.io.*;
public class Turing {
    public static void main(String... robots) {
        Console c = System.console();
        final String response = c.readLine("Are you human?");
        System.err.print(response);
    }
}
```

- I.** The program may ask the user a question and print the response to the error stream.
 - II.** The program may throw a `NullPointerException` at runtime.
 - III.** The program may wait indefinitely.
- A.** I
- B.** I and III
- C.** II and III
- D.** I, II, and III

25. Which of the following statements about the `deleteTree()` method is correct?

```
public void deleteTree(File f) {
    if(!f.isDirectory())
        f.delete();
    else {
        Stream.of(f.list())
            .forEach(s -> deleteTree(s));
        f.deleteDirectory();
    }
}
```

- A.** It compiles and is capable of deleting a directory tree.
- B.** If one line were modified, it would be capable of deleting a directory tree.
- C.** If two lines were modified, it would be capable of deleting a directory tree.
- D.** None of the above
- 26.** Which of the following is not a built-in stream in Java?
- A.** `System.err`
- B.** `System.in`

- C. `System.info`
 - D. `System.out`
27. Assuming the file path referenced in the following class is accessible and able to be written, what is the output of the following program?
- ```
package store;
import java.io.*;
public class Furniture {
 public final static void main(String... inventory) throws Exception {
 Writer w = new FileWriter("couch.txt");
 try (BufferedWriter bw = new BufferedWriter(w)) {
 bw.write("Blue coach on Sale!");
 } finally {
 w.flush();
 w.close();
 }
 System.out.print("Done!");
 }
}
```
- A. Done!
  - B. The code does not compile for one reason.
  - C. The code does not compile for two reasons.
  - D. The code compiles but throws an exception at runtime.
28. Given an instance of `Console c`, which of the following method calls is not a way to read input from the user?
- A. `c.reader().read()`
  - B. `c.reader().readLine()`
  - C. `c.readLine()`
  - D. `c.readPassword()`
29. The `copyPidgin()` method is used to copy the contents of one file to another. Which statement about the implementation is correct?

```
package birds;
import java.io.*;
public class Pidgin {
 public void copyPidgin(File s, File t) throws Exception {
 try(InputStream is = new FileInputStream(s);
 OutputStream os = new FileOutputStream(t)) {
 byte[] data = new byte[123];

```

```
 int chirps;
 while((chirps = is.read(data))>0) {
 os.write(data);
 }
 }}
```

- A. The class does not compile because `read(byte[])` and `write(byte[])` can only be called on `BufferedInputStream` and `BufferOutputStream`, respectively.
- B. The method correctly copies the contents of all files.
- C. The method correctly copies the contents of some files.
- D. The method will always throw an exception at runtime because the `data` array size is not a power of 2.
30. Using what you know about `java.io` stream class names, what would a nonexistent class named `BufferedReader` most likely be used for?
- A. Reading a small text file from a remote network
- B. Reading an image from disk
- C. Reading large text files from a file system
- D. Reading serialized data from disk
31. What is the output of the following application?

```
package factory;
import java.io.*;
public class WidgetProcessor {
 public int getWidgetNumber(byte[] data) throws Exception {
 try (InputStream is = new ByteArrayInputStream(data)) {
 is.read(new byte[2]);
 if(!is.markSupported()) return -1;
 is.mark(5);
 is.read();is.read();
 is.skip(3);
 is.reset();
 return is.read();
 }
 }
 public static void main(String... sprockets) throws Exception {
 final WidgetProcessor p = new WidgetProcessor();
 System.out.print(p.getWidgetNumber(new byte[] {1,2,3,4,5,6,7}));
 }
}
```

- A. 3
  - B. 5
  - C. 7
  - D. An exception is thrown at runtime.
32. Assuming the working directory is accessible, empty, and able to be written, how many file system objects does the following class create?
- ```
1: package kitchen;
2: import java.io.*;
3: public class Bakers {
4:     public static void main(String... tooMany) throws IOException {
5:         File cake = new File("cake.txt");
6:         Writer pie = new FileWriter("pie.txt");
7:         pie.flush();
8:         new File("fudge.txt").mkdirs();
9:     }
}
```
- A. None
 - B. One
 - C. Two
 - D. Three
33. Let's say you wanted to read data from a file stored on disk that consists of `String`, `long`, and `Object` values? Given that the file is quite large, you intend to use three classes to achieve this result. Which of the following is not one of the three classes you should use?
- A. `BufferedInputStream`
 - B. `BufferedReader`
 - C. `FileInputStream`
 - D. `ObjectInputStream`
34. Which statement best describes the following two methods?

```
public String getNameQuick() throws IOException {
    final BufferedReader r = new BufferedReader(
        new FileReader("saved.name"));
    final String name = r.readLine();
    r.flush();
    return name;
}
public String getNameSafely() throws IOException {
    try(final BufferedReader r = new BufferedReader(

```

```
        new FileReader("saved.name"))){  
    final String name = r.readLine();  
    r.flush();  
    return name;  
}}
```

- A.** Both methods compile and are equivalent to each other.
 - B.** Neither method compiles.
 - C.** Only one of the methods compiles.
 - D.** The methods compile, but one method may lead to a resource leak.
- 35.** What is the output of the following application? Assume the `System.console()` is available and the user enters `badxbad` and presses Enter.

```
package hardway;  
import java.io.*;  
public class InconvenientImplementation {  
    public static void main(String... dontDoThis) throws Exception {  
        Console c = System.console();  
        if(c != null) {  
            c.writer().write('P');  
            c.writer().write('a');  
            c.writer().write('s');  
            c.writer().write('s');  
            c.writer().flush(); // t1  
            int i;  
            StringBuilder sb = new StringBuilder();  
            while((i = c.reader().read()) != 'x') { // t2  
                sb.append((char)i);  
            }  
            c.writer().format("Result: %s",sb.toString());  
        }  
    }  
}
```

- A.** Result: bad
- B.** Line t1 does not compile or triggers an exception at runtime.
- C.** Line t2 does not compile or triggers an exception at runtime.
- D.** None of the above

36. Why does `Console readPassword()` return a char array rather than a `String`?

- A.** It improves performance.
- B.** It improves security.
- C.** Passwords must be stored as a char array.
- D.** `String` cannot hold the individual password characters.

37. Which statement about the following program is true?

```
package mystical;
import java.io.*;
public class Unicorn {
    public void findUnicorns() {
        try(InputStream o = new ObjectInputStream(readBook())) {
            while(o.read() != -1) {
                System.out.println(o.read());
            }
        } catch (Throwable t) {
            throw new RuntimeException(t);
        }
    }
    private InputStream readBook() throws IOException {
        return new BufferedInputStream(new FileReader("magic.book"));
    }
    public static void main(String... horn) {
        new Unicorn().findUnicorns();
    }
}
```

- A.** The code does not compile.
- B.** The program prints every byte in the file without throwing an exception.
- C.** The program prints every other byte in the file without throwing an exception.
- D.** The program throws an `EOFException` when the end of the file is reached.

38. Choose the class that is least likely to be marked `Serializable`.

- A.** A class that holds data about the amount of rain that has fallen in a given year
- B.** A class that manages the memory of running processes in an application
- C.** A class that stores information about apples in an orchard
- D.** A class that tracks the amount of candy in a gumball machine

39. What is the output of the following application?

```
package cell;
import java.io.*;
public class TextMessage {
    public String receiveText() throws Exception {
        try (Reader r = new FileReader("messages.txt")) {
            StringBuilder s = new StringBuilder();
            int c;
            while((c = r.read()) != -1) {
                s.append((char)c);
                if(r.markSupported()) {
                    r.mark(100);
                    r.skip(10);
                    r.reset();
                }
            }
            return s.toString();
        }
    }
    public void sendText(String message) throws Exception {
        try (Writer w = new FileWriter("messages.txt")) {
            for(int i=0; i<message.length(); i++) {
                w.write(message.charAt(i));
                w.skip(1);
            }
        }
    }
    public static void main(String[] minutes) throws Exception {
        final TextMessage m = new TextMessage();
        m.sendText("You up?");
        System.out.println(m.receiveText());
    }
}
```

- A. You up?
- B. Y o u u p ?
- C. The code does not compile because of the `receiveText()` method.
- D. The code does not compile because of the `sendText()` method.

40. What is the output of the following program? Assume the file paths referenced in the class exist and are able to be written to and read from.

```
package heart;
import java.io.*;
public class Valve implements Serializable {
    private int chambers = -1;
    private transient Double size = null;
    private static String color;
    public Valve() {
        this.chambers = 3;
        color = "BLUE";
    }

    public static void main(String[] args) throws Throwable {
        try (ObjectOutputStream o = new ObjectOutputStream(
                new FileOutputStream("scan.txt"))) {
            final Valve v = new Valve();
            v.chambers = 2;
            v.size = 10.0;
            v.color = "RED";
            o.writeObject(v);
        }
        new Valve();
        try (ObjectInputStream o = new ObjectInputStream(
                new FileInputStream("scan.txt"))) {
            Valve v = (Valve)o.readObject();
            System.out.print(v.chambers+","+v.size+","+v.color);
        }
    }
    { chambers = 4; }
}
```

- A. 2,null,RED
- B. 2,null,BLUE
- C. 3,10.0,RED
- D. The code does not compile.

Chapter 19



Java File I/O (NIO.2)

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

✓ **Java File I/O (NIO.2)**

- Use Path interface to operate on file and directory paths
- Use Files class to check, read, delete, copy, move, manage metadata of a file or directory
- Use Stream API with NIO.2

1. Fill in the blanks: A(n) _____ is a file that contains a reference to another file or directory, while a(n) _____ is a file that contains content.
 - A. irregular file, regular file
 - B. regular file, opaque file
 - C. symbolic link, regular file
 - D. symbolic link, symbolic directory
2. Which methods listed below are found in the NIO.2 Path interface?
 - I. getRoot()
 - II. isDirectory()
 - III. listFiles()
 - IV. toRealPath()
 - A. I only
 - B. I, II, and III
 - C. I and IV
 - D. II and III
3. Assuming the file /secret/hidden.txt exists and is marked hidden, what is result of executing the following program?

```
package hidden;
import java.nio.file.*;
public class Finder {
    public void findHiddenFile(Path p) throws Exception {
        if(File.isHidden(p)) {
            System.out.print("Found!");
        }
    }
    public static void main(String[] folders) throws Exception {
        final Finder f = new Finder();
        f.findHiddenFile(Paths.get("/secret/hidden.txt"));
    }
}
```

- A. The class does not compile.
- B. An exception is printed at runtime.
- C. Found! is printed at runtime.
- D. Nothing is printed at runtime.

4. Fill in the blanks: `Files.walk()` performs a _____ traversal, while `Files.find()` performs a _____ traversal.
- A. breadth-first, breadth-first
 - B. breadth-first, depth-first
 - C. depth-first, breadth-first
 - D. depth-first, depth-first
5. When reading file information, what is an advantage of using an NIO.2 attribute interface rather than reading the values individually from `Files` methods?
- A. Costs fewer round-trips to the file system
 - B. Guarantees performance improvement
 - C. Has support for symbolic links
 - D. Reduces memory leaks
6. What is the result of compiling and executing the following program? Assume the current directory is `/stock` and the path `/stock/sneakers` does not exist prior to execution.

```
package shoe;
import java.io.*;
import java.nio.file.*;
public class Sneaker {
    public void setupInventory(Path desiredPath) throws Exception {
        Path suggestedPath = Paths.get("sneakers");
        if(Files.isSameFile(suggestedPath, desiredPath) // j1
            && !Files.exists(suggestedPath))
            Files.createDirectories(desiredPath); // j2
    }
    public static void main(String[] socks) throws Exception {
        Path w = new File("/stock/sneakers").toPath(); // j3
        new Sneaker().setupInventory(w);
    }
}
```

- A. The directory `/stock/sneakers` is created.
- B. Line j1 does not compile or produces an exception at runtime.
- C. Line j2 does not compile or produces an exception at runtime.
- D. Line j3 does not compile or produces an exception at runtime.

7. Assuming the path referenced below exists and contains a symbolic link that references /again, what is the expected result of executing the following code snippet?

```
System.out.print(Files.walk(Paths.get("/again/and/again")).count());
```

- A. An exception is thrown at runtime.
 - B. A number is printed at runtime.
 - C. The process hangs indefinitely.
 - D. The result cannot be determined with the information given.
8. Which method in the NIO.2 Files class is equivalent to the `java.io.File` method `length()`?
- A. `length()`
 - B. `size()`
 - C. `getLength()`
 - D. None of the above
9. Assuming the current working directory is `/home`, then what is the output of the following program?

```
1: package magic;
2: import java.nio.file.*;
3: public class Magician {
4:     public String doTrick(Path path) {
5:         return path.subpath(2,3)
6:             .getName(1)
7:             .toAbsolutePath()
8:             .toString();
9:     }
10:    public static void main(String... cards) {
11:        final Magician m = new Magician();
12:        System.out.print(m.doTrick(
13:            Paths.get("/bag/of/tricks/.../disappear.txt")));
14:    } }
```

- A. `/home/tricks`
- B. `/home`
- C. The code does not compile.
- D. The code compiles but prints an exception at runtime.

10. Which methods listed below are found in the NIO.2 Files class?

- I. `isSameFile()`
 - II. `length()`
 - III. `relativize()`
 - IV. `mkdir()`
- A. I only
 - B. I, II, and IV
 - C. II and III
 - D. IV only

11. The following code snippet, which attempts to move a file system record from `oldHardDrivePath` to `newHardDrivePath`, results in an exception at runtime. Which of the following is the most likely type of exception to be thrown?

```
Files.move(oldHardDrivePath,newHardDrivePath,REPLACE_EXISTING);
```

- A. `AtomicMoveNotSupportedException`
- B. `DirectoryNotEmptyException`
- C. `FileAlreadyExistsException`
- D. None of the above since the line of code does not compile

12. Which of the following can be filled into the blank that would allow the method to compile?

```
public String getPathName(String fileName) {  
    final Path p = _____;  
    return p.getFileName();  
}
```

- I. `new File(fileName).toPath()`
 - II. `new Path(fileName)`
 - III. `FileSystems.getDefault().getPath(fileName)`
- A. I and II
 - B. I and III
 - C. II
 - D. None of the above

13. Which statement about the following class is correct?

```
package clone;
import java.io.*;
import java.nio.file.*;
public class Rewriter {
    public static void copy(Path source, Path target) throws Exception {
        try (BufferedReader r = Files.newBufferedReader(source);
             Writer w = Files.newBufferedWriter(target)) {
            String temp = null;
            while((temp = r.readLine()) != null) {
                w.write(temp);
            }
        }
    }
    public static void main(String[] tooMany) throws Throwable {
        Rewriter.copy(Paths.get("/original.txt"),
                      FileSystems.getDefault().getPath("/", "unoriginal.txt"));
    }
}
```

- A. The class compiles without issue.
 - B. The class never throws an exception at runtime.
 - C. The implementation correctly copies a regular file.
 - D. All of the above
14. Fill in the blanks: The `Files.`_____ method returns a `List`, while the `Files.`_____ method returns a `Stream`.

- A. `lines()`, `readAllLines()`
- B. `lines()`, `readLines()`
- C. `readAllLines()`, `lines()`
- D. `readLines()`, `lines()`

15. What is the output of the following application?

```
1: package yellow;
2: import java.nio.file.*;
3: public class Road {
4:     public boolean findHome() {
5:         Path oftenTraveled = Paths.get("/highway/street/spot.txt");
6:         Path lessTraveled = Paths.get("/highway/street/house/../.+");
7:         lessTraveled.resolve("spot.txt");
8:         return oftenTraveled.equals(lessTraveled.normalize());
```

```
9:      }
10:     public static void main(String... emerald) {
11:       System.out.print("AM I HOME? "
12:           +(new Road().findHome() ? "yes" : " no"));
13:     }
14: }
```

- A.** AM I HOME? no
 - B.** AM I HOME? yes
 - C.** The class does not compile.
 - D.** The class compiles but throws an exception at runtime.
- 16.** Which of the following is not an advantage of using an NIO.2 Path instead of a `java.io.File` to work with files?
- A.** Contains built-in support for symbolic links
 - B.** Has ability to read operating-system-specific attributes
 - C.** Provides a single method for deleting a directory tree
 - D.** Provides efficient access of file metadata
- 17.** What is the result of executing the following program? Assume the path `/driveway` exists and is non-empty, and the directory tree is fully accessible within the file system.

```
package weather;
import java.io.*;
import java.nio.file.*;
public class Snow {
    public static boolean removeSnow(Path flake) throws IOException {
        if(!Files.isDirectory(flake) && !Files.isSymbolicLink(flake))
            return Files.delete(flake);
        else return true;
    }
    public static void main(String[] cones) throws IOException {
        File driveway = new File("/driveway");
        for(File f : driveway.listFiles()) {
            System.out.println(removeSnow(f.toPath()));
        }
    }
}
```

- A.** The program prints a list of only true values.
- B.** The program prints a mix of true and false values.
- C.** The code does not compile.
- D.** The code compiles but prints an exception at runtime.

18. Which interface name inserted into the blank below allows the code snippet to compile?

```
Path file = Paths.get("/data/movie.txt");
BasicFileAttributes b = Files.readAttributes(file, _____);
```

- A. BasicFileAttributes.class
- B. DosFileAttributes.class
- C. PosixFileAttributes.class
- D. All of the above

19. What is the output of the following code snippet? Assume that the current directory is the root path.

```
Path p1 = Paths.get("./locks");
Path p2 = Paths.get("/found/red.zip");
System.out.println(p1.relativize(p2));
System.out.println(p2.relativize(p1));
```

- A. ../found/red.zip
..../locks
- B. .../.../locks
../found/red.zip
- C. locks/.../found/red.zip
../found/locks
- D. None of the above

20. What is the output of the following code snippet? Assume that the current directory is the root path.

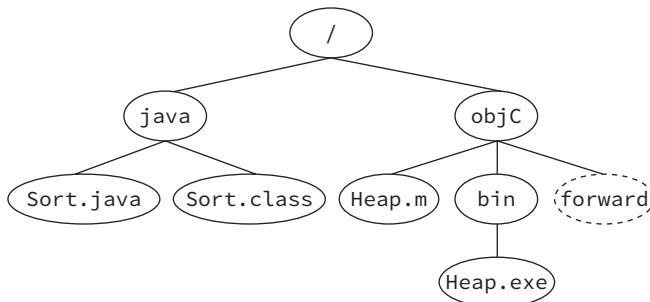
```
Path p1 = Paths.get("./found/.../keys");
Path p2 = Paths.get("/lost/blue.txt");
System.out.println(p1.resolve(p2));
System.out.println(p2.resolve(p1));
```

- A. /lost/blue.txt
./found/.../keys
- B. /found/.../keys./lost/blue.txt
/lost/blue.txt/keys
- C. /lost/blue.txt
/lost/blue.txt./found/.../keys
- D. None of the above

21. What is the output of the following application? Assume the application is called with a valid path that exists and is accessible within the file system.

```
package charity;
import java.nio.file.*;
public class Roster {
    protected void printRoster(Path p) {
        for(Path f : Files.list(p)) { // n1
            if(f.toString().endsWith(".per")) // n2
                System.out.print(f);
        }
    }
    public static void main(String... volunteers) {
        new Roster().printRoster(Paths.get(volunteers[0]));
    }
}
```

- A. A list of file names is printed at runtime.
 - B. The class does not compile due to line n1.
 - C. The class does not compile due to line n2.
 - D. None of the above
22. Given the following file system diagram, in which `forward` is a symbolic link to the `java` directory, which value does not print `/java/Sort.java` at runtime?



```
Path p = Paths.get("/", "objC", "bin");
System.out.println(p.resolve("_____").toRealPath());
```

- A. `../backwards/../forward/Sort.java`
- B. `../forward../Sort.java`
- C. `../java../forward/Sort.java`
- D. `....../java/Sort.java`

23. Using the file system diagram from the previous question, including the symbolic link from forward to java, how many calls to `Files.delete()` would need to be made before the following line could be executed without throwing an exception?

```
Files.delete(Paths.get("/objC"));
```

- A. One
- B. Four
- C. Seven
- D. None of the above. The symbolic link needs to be removed with `Files.deleteSymbolicLink()` first.

24. Assuming the `course.txt` file exists and is readable, what is the result of executing the following application?

```
package schoolwork;
import java.io.*;
import java.nio.file.*;
public class Notes {
    public void printNotes() {
        try (OutputStream out = System.out) { // y1
            Files.copy(out, Paths.get("course.txt"));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    public static void main(String[] coursework) {
        new Notes().printNotes();
    }
}
```

- A. The code compiles but prints an exception at runtime.
 - B. The class does not compile due to line y1.
 - C. The code does not compile for some other reason.
 - D. The program prints the contents of the `course.txt` file.
25. When reading file information, what is an advantage of loading a `BasicFileAttributeView` over a `BasicFileAttributes`?
- A. Allows the hidden attribute to be set
 - B. Allows the last modified date to be changed
 - C. All of the file information is read in a single round-trip.
 - D. There is no advantage.

- 26.** The Rose application is run with an input argument of /flower. The /flower directory contains five subdirectories, each of which contains five files. How many Path values does the following application print?

```
import java.nio.file.*;
public class Rose {
    public void tendGarden(Path p) throws Exception {
        Files.walk(p,1)
            .map(p -> p.toRealPath())
            .forEach(System.out::println);
    }
    public static void main(String... thorns) throws Exception {
        new Rose().tendGarden(Paths.get(thorns[0]));
    }
}
```

- A.** None
 - B.** One
 - C.** Six
 - D.** Thirty-one
- 27.** Which of the following statements, when run independently, produces a `NullPointerException` at runtime?
- I.** `Paths.get("../sang").getParent().getParent()`
 - II.** `Paths.get("/sing").getParent().getRoot()`
 - III.** `Paths.get("/song").getRoot().getRoot()`
 - IV.** `Paths.get("../sung").getRoot().getParent()`
- A.** I and III
 - B.** I and IV
 - C.** II and III
 - D.** IV only
- 28.** Which statement about the following `Finalize` class is correct?

```
1: package end;
2: import java.nio.file.*;
3: public class Finalize {
4:     public Path makeAbsolute(Path p) {
5:         if(p!=null && !p.isAbsolute())
6:             return p.toAbsolutePath();
```

```
7:         return p;
8:     }
9: }
```

- A. It does not compile because IOException is neither handled nor declared.
 - B. It does not compile for some other reason.
 - C. The method compiles and returns a Path value that is always equivalent to the input argument.
 - D. The method compiles and returns a Path value that may not be equivalent to the input argument.
29. Which of the following is a difference between the `createDirectory()` and `createDirectories()` methods found in the NIO.2 `Files` class?
- A. One takes multiple Path arguments; the other does not.
 - B. One throws an exception if a file already exists at the directory path; the other does not.
 - C. One declares a checked exception; the other does not.
 - D. One creates a single directory while the other may create many directories.
30. Assuming the current working directory is `/hail`, what is the expected output of executing the following code snippet?

```
Path w1 = Paths.get("../jungle/../../rain..")
    .toAbsolutePath().normalize();
System.out.print(w1.resolve("snow.txt"));
```

- A. `/jungle/snow.txt`
 - B. `/hail/rain../snow.txt`
 - C. `/rain../snow.txt`
 - D. An exception is printed at runtime.
31. What is the output of the following application?

```
package med;
import java.nio.file.*;
public class Surgeon {
    public Path rebuild(Path p) {
        Path v = null;
        for(int i=0; i<p.getNameCount(); i++)
            if(v==null) v = p.getName(i);
            else v = v.resolve(p.getName(i));
        return v;
    }
    public static void main(String... tools) {
        final Surgeon al = new Surgeon();
```

```
        Path original = Paths.get("/tissue/heart/chambers.txt");
        Path repaired = al.rebuild(original);
        System.out.print(original.equals(repaired));
    }
}

A. false
B. true
C. The code does not compile.
D. The code compiles but prints an exception at runtime.
```

32. Under which circumstances does `Files.deleteIfExists()` not throw an exception?
- A. The file system suddenly becomes unavailable.
 - B. The path does not exist.
 - C. The path represents a non-empty directory.
 - D. The process does not have write access to a path.
33. What is the output of the following code snippet? Assume all referenced paths exist within the file system.

```
Path v1 = Path.get("./desert/./").resolve(Paths.get("sand.doc"));
Path v2 = new File("/desert./cactus/../sand.doc").toPath();
System.out.print(Files.isSameFile(v1,v2));
System.out.print(" "+v1.equals(v2));
System.out.print(" "+v1.normalize().equals(v2.normalize()));
```

- A. false false false
- B. true false true
- C. true true true
- D. None of the above

34. How many lines of the following program contain compilation errors?

```
public class Song {
    public static void organize(Path folder, Path file) throws IOException {
        Path p = folder.resolve(file);
        BasicFileAttributeView vw = Files.getFileAttributeView(p,
            BasicFileAttributes.class);
        if(vw.creationTime().toMillis()<System.currentTimeMillis()) {
            vw.setTimes(ToFileTime.fromMillis(System.currentTimeMillis()),
                null,null);
        }
    }
    public static void main(String[] audio) throws Exception {
```

```

        Song.organize(Paths.get("/", "pub"), new File("/songs").toPath());
    }
}

```

- A.** None
B. One
C. Two
D. Three
- 35.** What is the output of the following application?

```

package stars;
import java.nio.file.*;
public class Sun {
    public void printInfo() {
        Path halleyComet = Paths.get("stars./rocks/..m1.meteor")
            .normalize();
        Path lexellsComet = Paths.get("./stars/..solar/");
        lexellsComet = lexellsComet.subpath(0, 2)
            .resolve("m1.meteor")
            .normalize();
        System.out.print(halleyComet.equals(lexellsComet)
            ? "Same!" : "Different!");
    }
    public static void main(String... emerald) {
        Sun s = new Sun();
        s.printInfo();
    }
}

```

- A.** Different!
B. Same!
C. The class does not compile.
D. The class compiles but throws an exception at runtime.
- 36.** Assuming the directory `/eclipse/projects` exists and its contents are accessible, which statement about the following code snippet is correct?

```

Path p = Paths.get("/eclipse/projects");
Files.walk(p)
    .map(z -> z.toAbsolutePath().toString())
    .filter(s -> s.endsWith(".java"))

```

```
.collect(Collectors.toList()).forEach(System.out::println);

Files.find(p, Integer.MAX_VALUE,
    (w,a) -> w.getAbsoluteFile().toString().endsWith(".java"))
.collect(Collectors.toList()).forEach(System.out::println);
```

- A. The first stream statement does not compile.
- B. The second stream statement does not compile.
- C. Both statements compile but are unlikely to print the same results at runtime.
- D. None of the above
37. Assuming the file referenced below exists and is significantly large, which statement about the following program is correct?

```
public class SpeedRead {
    public void jenniferReads(Path p) {
        Files.lines(p);
    }
    public void jonReads(Path p) {
        Files.readAllLines(p);
    }
    public static void main(String[] pages) {
        Path p = Paths.get("/bookshelf/mobydick.txt");
        final SpeedRead r = new SpeedRead();
        r.jenniferReads(p);
        r.jonReads(p);
    }
}
```

- A. The code does not compile.
- B. The method `jenniferReads()` is likely to take longer to run.
- C. The method `jonReads()` is likely to take longer to run.
- D. It is not possible to know which method will take longer to run.
38. What is the result of executing the following program? Assume the files referenced in the application both exist and are fully accessible within the file system.

```
package duplicate;
import static java.nio.file.StandardCopyOption.*;
import static java.nio.file.Files.*;
import java.io.*;
import java.nio.*;
public class CopyOfACopy {
```

```
public void main(String[] items) throws Exception {  
    final Path s = new File("apples.zip").toPath();  
    final Path t = FileSystems.getDefault().getPath("oranges.zip");  
    copy(s,t,REPLACE_EXISTING); // q1  
    copy(Files.newBufferedReader(t),t,ATOMIC_MOVE); // q2  
}  
}
```

- A. Line q1 does not compile.
 - B. Line q1 produces an exception at runtime.
 - C. Line q2 does not compile.
 - D. Line q2 produces an exception at runtime.
39. Which of the following `Files` methods requires the enclosing method to handle or declare a checked exception?
- A. `exists()`
 - B. `isDirectory()`
 - C. `isSameFile()`
 - D. `isSymbolicLink()`
40. What is the output of the following application? Assume `/all-data` exists and is accessible within the file system.

```
package numbers;  
import java.nio.file.*;  
import java.util.stream.Stream;  
public class TheCount {  
    public static Stream<String> readLines(Path p) {  
        try {  
            return Files.lines(p);  
        } catch (Exception e) {  
            throw new RuntimeException(e);  
        }  
    }  
    public static long count(Path p) throws Exception {  
        return Files.list(p)  
            .filter(w -> Files.isRegularFile(w))  
            .flatMap(s -> readLines(s))  
            .count();  
    }  
}
```

```
public final static void main(String[] day) throws Exception {  
    System.out.print(count(Paths.get("/all-data")));  
}  
}
```

- A.** The number of lines in all files in a directory tree
- B.** The number of lines in all files in a single directory
- C.** The code does not compile.
- D.** The code compiles but prints an exception at runtime.

Chapter 20



Java Concurrency

**THE OCP EXAM TOPICS COVERED
IN THIS PRACTICE TEST INCLUDE
THE FOLLOWING:**

✓ **Java Concurrency**

- Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
- Identify potential threading problems among deadlock, starvation, livelock, and race conditions
- Use synchronized keyword and `java.util.concurrent.atomic` package to control the order of thread execution
- Use `java.util.concurrent` collections and classes including CyclicBarrier and CopyOnWriteArrayList
- Use parallel Fork/Join Framework
- Use parallel Streams including reduction, decomposition, merging processes, pipelines and performance.

1. Which of the following methods is not available on an ExecutorService instance?
 - A. execute(Callable)
 - B. execute(Runnable)
 - C. submit(Callable)
 - D. submit(Runnable)
2. Which statements about executing the following TicketTaker application multiple times are true?

```
package performance;
import java.util.concurrent.atomic.*;
import java.util.stream.*;
public class TicketTaker {
    long ticketsSold;
    final AtomicInteger ticketsTaken;
    public TicketTaker() {
        ticketsSold = 0;
        ticketsTaken = new AtomicInteger(0);
    }
    public void performJob() {
        IntStream.iterate(1, p -> p+1)
            .parallel()
            .limit(10)
            .forEach(i -> ticketsTaken.getAndIncrement());
        IntStream.iterate(1, q -> q+1)
            .limit(5)
            .parallel()
            .forEach(i -> ++ticketsSold);
        System.out.print(ticketsTaken+ " "+ticketsSold);
    }
    public static void main(String[] matinee) {
        new TicketTaker().performJob();
    }
}
```

- I. The class compiles and runs without throwing an exception.
 - II. The first number printed is consistently 10.
 - III. The second number printed is consistently 5.
- A. I only
 - B. I and II
 - C. I, II, and III
 - D. None of the above

3. Which of the following is a recommended way to define an asynchronous task?
- A. Create a `Callable` expression and pass it to an instance of `Executors`.
 - B. Create a class that extends `Thread` and overrides the `start()` method.
 - C. Create a `Runnable` expression and pass it to a `Thread` constructor.
 - D. All of the above
4. Let's say you needed a thread executor to create tasks for a `CyclicBarrier` that has a barrier limit of five threads. Which `static` method in `ExecutorService` should you use to obtain it?
- A. `newSingleThreadExecutor()`
 - B. `newSingleThreadScheduledExecutor()`
 - C. `newCachedThreadPool()`
 - D. None of these would work.
5. Given the `original` array, how many of the following `for` statements result in an exception at runtime, assuming each is executed independently?

```
List<Integer> original = new ArrayList<>(Arrays.asList(1,2,3,4,5));
```

```
List<Integer> copy1 = new CopyOnWriteArrayList<>(original);
```

```
for(Integer w : copy1)
```

```
    copy1.remove(w);
```

```
List<Integer> copy2 = Collections.synchronizedList(original);
```

```
for(Integer w : copy2)
```

```
    copy2.remove(w);
```

```
List<Integer> copy3 = new ArrayList<>(original);
```

```
for(Integer w : copy3)
```

```
    copy3.remove(w);
```

```
Queue<Integer> copy4 = new ConcurrentLinkedQueue<>(original);
```

```
for(Integer w : copy4)
```

```
    copy4.remove(w);
```

A. Zero

B. One

C. Two

D. Three

6. Fill in the blanks: _____ is a special case of _____, in which two or more active threads try to acquire the same set of locks and are repeatedly unsuccessful.
- A. Deadlock, livelock
 - B. Deadlock, resource starvation
 - C. Livelock, resource starvation
 - D. Resource starvation, race conditions

7. What is the output of the following application?

```
1: package office;
2: import java.util.concurrent.*;
3: public class TpsReport {
4:     public void submitReports() {
5:         ExecutorService service = Executors.newCachedThreadPool();
6:         Future bosses = service.submit(() -> System.out.print(""));
7:         service.shutdown();
8:         System.out.print(bosses.get());
9:     }
10:    public static void main(String[] memo) {
11:        new TpsReport().submitReports();
12:    }
13: }
```

- A. null
- B. The code does not compile.
- C. Line 7 throws an exception at runtime.
- D. Line 8 throws an exception at runtime.

8. Which of the following `static` methods does not exist in the `Executors` class?

- A. `newFixedScheduledThreadPool()`
- B. `newFixedThreadPool()`
- C. `newSingleThreadExecutor()`
- D. `newSingleThreadScheduledExecutor()`

9. How many times does the following application print Ready at runtime?

```
package parade;
import java.util.concurrent.*;
public class CartoonCat {
    private void await(CyclicBarrier c) {
        try {
            c.await();
        } catch (Exception e) {}
    }
    public void march(CyclicBarrier c) {
        ExecutorService s = Executors.newSingleThreadExecutor();
        for(int i=0; i<12; i++)
            s.execute(() -> await(c));
        s.shutdown();
    }
}
```

```
public static void main(String... strings) {  
    new CartoonCat().march(new CyclicBarrier(4,  
        () -> System.out.println("Ready")));  
}
```

- A.** Zero
 - B.** One
 - C.** Three
 - D.** The code does not compile.
10. Which thread-safe class would you use to add elements to the front and back of an ordered data structure and includes methods for waiting a specified amount of time to do so?
- A.** BlockingDeque
 - B.** ConcurrentLinkedDeque
 - C.** ConcurrentSkipListSet
 - D.** LinkedBlockingDeque
11. Three of the four methods below always produce the same result whether they are executed on a serial or parallel ordered stream. Which one does not?
- A.** `findAny()`
 - B.** `findFirst()`
 - C.** `limit()`
 - D.** `skip()`
12. What is the result of executing the following application multiple times?

```
package bears;  
import java.util.*;  
public class Bounce {  
    public static void main(String... legend) {  
        Arrays.asList(1,2,3,4).stream()  
            .forEach(System.out::println);  
        Arrays.asList(1,2,3,4).parallel()  
            .forEachOrdered(System.out::println);  
    }  
}
```

- A.** Only the first array is printed in the same order every time.
- B.** Only the second array is printed in the same order every time.
- C.** Both arrays are printed in the same order every time.
- D.** None of the above

13. Fill in the blanks: In the fork/join framework, using the _____ class requires overriding an abstract `compute()` method containing a generic return type, while using the _____ class requires overriding an abstract `compute()` method containing a `void` return type.

- A. `ForkJoinTask`, `RecursiveAction`
- B. `RecursiveAction`, `RecursiveTask`
- C. `RecursiveTask`, `ForkJoinTask`
- D. `RecursiveTask`, `RecursiveAction`

14. Given the following code snippet, which lambda expression is the best choice for the accumulator, based on the rules for applying a parallel reduction?

```
public class GoodAccumulator {  
    int i;  
    public void test() {  
        BiFunction<Integer, Integer, Integer> accumulator = _____;  
        System.out.print(Arrays.asList(1,2,3,4,5)  
            .parallelStream()  
            .reduce(0, accumulator, (s1, s2) -> s1 + s2));  
    }  
}
```

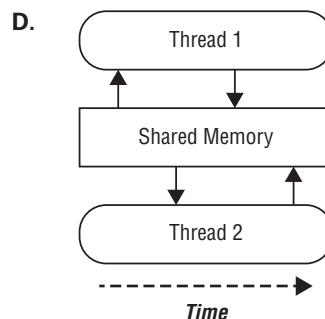
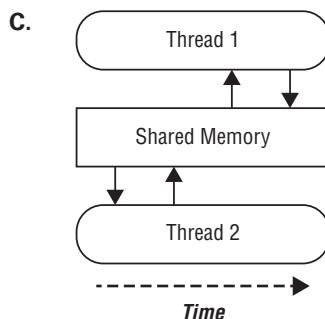
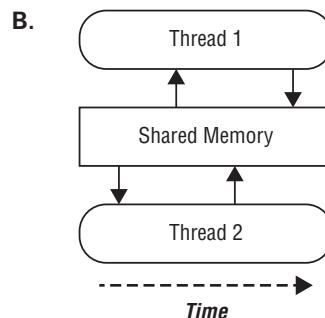
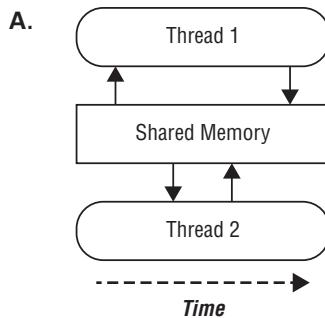
- A. $(a,b) \rightarrow (a-b)$
- B. $(a,b) \rightarrow 5$
- C. $(a,b) \rightarrow i++$
- D. None of the above are appropriate.

15. What is the output of the following code snippet?

```
Callable c = new Callable() {  
    public Object run() {return 10;}  
};  
ExecutorService s = Executors.newScheduledThreadPool(1);  
for(int i=0; i<10; i++) {  
    Future f = s.submit(c);  
    f.get();  
}  
s.shutdown();  
System.out.print("Done!");
```

- A. Done!
- B. The code does not compile.
- C. The code hangs indefinitely at runtime.
- D. The code throws an exception at runtime.

16. The following diagrams represent the order of read/write operations of two threads sharing a common variable. Each thread first reads the value of the variable from memory and then writes a new value of the variable back to memory. Which diagram demonstrates proper synchronization?



17. What is the output of the following application?

```
package story;
import java.util.*;
import java.util.concurrent.*;
public class Race {
    static ExecutorService service = Executors.newFixedThreadPool(8);
    public static int sleep() {
        try {
            Thread.sleep(1000);
        } catch (Exception e) {}
        return 1;
    }
}
```

```
public static void hare() {  
    try {  
        Callable c = () -> sleep();  
        final Collection<Callable<Integer>> r = Arrays.asList(c,c,c);  
        List<Future<Integer>> results = service.invokeAll(r);  
        System.out.println("Hare won the race!");  
    } catch (Exception e) {e.printStackTrace();}  
}  
public static void tortoise() {  
    try {  
        Callable c = () -> sleep();  
        final Collection<Callable<Integer>> r = Arrays.asList(c,c,c);  
        Integer result = service.invokeAny(r);  
        System.out.println("Tortoise won the race!");  
    } catch (Exception e) {e.printStackTrace();}  
}  
public static void main(String[] p) throws Exception {  
    service.execute(() -> hare());  
    service.execute(() -> tortoise());  
}
```

- A.** Hare won the race! is printed first.
 - B.** Tortoise won the race! is printed first.
 - C.** The code does not compile.
 - D.** The result is unknown until runtime.
- 18.** Which of the following concurrent collections is sorted?
- A.** ConcurrentLinkedQueue
 - B.** ConcurrentSkipListMap
 - C.** CopyOnWriteArrayList
 - D.** LinkedBlockingQueue
- 19.** What is the most likely result of executing the following application?

```
package unknown;  
import java.util.concurrent.*;  
public class Riddle {  
    public void sleep() {  
        try {  
            Thread.sleep(5000);  
        } catch (Exception e) {}  
    }  
}
```

```
public String getQuestion(Riddle r) {  
    synchronized {  
        sleep();  
        if(r != null) r.getAnswer(null);  
        return "How many programmers does it take "  
            + "to change a light bulb?";  
    }  
}  
public synchronized String getAnswer(Riddle r) {  
    sleep();  
    if(r != null) r.getAnswer(null);  
    return "None, that's a hardware problem";  
}  
public static void main(String... ununused) {  
    final Riddle r1 = new Riddle();  
    final Riddle r2 = new Riddle();  
    ExecutorService s = Executors.newFixedThreadPool(2);  
    s.submit(() -> r1.getQuestion(r2));  
    s.execute(() -> r2.getAnswer(r1));  
    s.shutdown();  
}
```

- A. A deadlock is produced at runtime.
B. A livelock is produced at runtime.
C. The application completes successfully.
D. The code does not compile.
20. Which ScheduledExecutorService method can result in the same action being executed by two threads at the same time?
- A. scheduleAtFixedDelay()
B. scheduleAtFixedRate()
C. scheduleWithFixedDelay()
D. There is no such method in ScheduledExecutorService.
21. What is the output of the following application?

```
package olympics;  
import java.util.concurrent.*;  
public class Athlete {  
    int stroke = 0;  
    public synchronized void swimming() {  
        stroke++;  
    }  
}
```

```
public static void main(String... laps) {  
    ExecutorService s = Executors.newFixedThreadPool(10);  
    Athlete a = new Athlete();  
    for(int i=0; i<1000; i++) {  
        s.execute(() -> a.swimming());  
    }  
    s.shutdown();  
    System.out.print(a.stroke);  
}
```

- A. 1000
- B. The code does not compile.
- C. The result is unknown until runtime because `stroke` is not accessed in a thread-safe manner and a write may be lost.
- D. The result is unknown until runtime for some other reason.
22. Which of the following is most likely to be caused by a race condition?
- A. A thread perpetually denied access to a resource
- B. An `int` variable incorrectly reporting the number of times an operation was performed
- C. Two threads actively trying to restart a blocked process that is guaranteed to always end the same way
- D. Two threads endlessly waiting on each other to release shared locks
23. What is the output of the following application?

```
package farm;  
import java.util.concurrent.*;  
public class CountSheep extends RecursiveAction {  
    static int[] sheep = new int[] {1,2,3,4};  
    final int start;  
    final int end;  
    int count = 0;  
    public CountSheep(int start, int end) {  
        this.start = start;  
        this.end = end;  
    }  
    public void compute() {  
        if(end-start<2) {  
            count+=sheep[start];  
            return;  
        } else {
```

```
        int middle = start + (end-start)/2;
        invokeAll(new CountSheep(start,middle),
                  new CountSheep(middle,end));
    }
}

public static void main(String[] night) {
    ForkJoinPool pool = new ForkJoinPool();
    CountSheep action = new CountSheep(0,sheep.length);
    pool.invoke(action);
    pool.shutdown();
    System.out.print(action.count);
}
}

A. 0
B. 10
C. The code does not compile.
D. None of the above.
```

24. Which statement about parallel streams is correct?

- A.** A parallel stream always executes all stream operations faster than a serial stream.
- B.** A parallel stream always executes certain stream operations faster than a serial stream.
- C.** A parallel stream synchronizes its operations so that they are atomic.
- D.** All streams can be converted to a parallel stream.

25. What is a possible output of the following application?

```
package salvage;
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.*;
public class Car {
    private String model;
    private int year;
    public Car(String name, int year) {
        this.model = name; this.year = year;
    }
    public int getYear() {return year;}
    @Override public String toString() {return model;}
}

public static void main(String... make) {
    List<Car> cars = new ArrayList<>();
    cars.add(new Car("Mustang",1967));
```

```
        cars.add(new Car("Thunderbird",1967));
        cars.add(new Car("Escort",1975));
        ConcurrentMap<Integer, List<Car>> map = cars
            .stream()
            .collect(Collectors.groupingByConcurrent(Car::getYear));
        System.out.print(map);
    }
}
```

- A. {1975=[Escort], 1967=[Thunderbird, Mustang]}
- B. {Escort=[1975], Thunderbird=[1967], Mustang=[1967]}
- C. The code does not compile.
- D. The application throws an exception at runtime because the stream is not parallel.

26. What is the output of the following application?

```
package exercise;
import java.util.*;
public class Concat {
    public String concat1(List<String> values) {
        return values.parallelStream()
            .reduce("a",
                (x,y)->x+y,
                String::concat);
    }
    public String concat2(List<String> values) {
        return values.parallelStream()
            .reduce((w,z)->z+w).get();
    }
    public static void main(String... questions) {
        Concat c = new Concat();
        List<String> list = Arrays.asList("Cat","Hat");
        String x = c.concat1(list);
        String y = c.concat2(list);
        System.out.print(x+" "+y);
    }
}
```

- A. aCataHat HatCat
- B. CatHat CatHat
- C. The code does not compile because concat1() returns an Optional.
- D. The code does not compile for a different reason.

27. What is the output of the following application?

```
package taxes;
import java.util.concurrent.*;
public class Accountant {
    public static void completePaperwork() {
        System.out.print("[Filing]");
    }
    public static double getPi() {
        return 3.14159;
    }
    public static void main(String[] args) throws Exception {
        ExecutorService x = Executors.newSingleThreadExecutor();
        Future<?> f1 = x.submit(() -> completePaperwork());
        Future<Object> f2 = x.submit(() -> getPi());
        System.out.print(f1.get()+" "+f2.get());
        x.shutdown();
    }
}
```

- A.** [Filing]null 3.14159
- B.** The declaration of f1 does not compile.
- C.** The declaration of f2 does not compile.
- D.** An exception is thrown at runtime.

28. Which statement about the following class is correct?

```
package my;
import java.util.*;
public class ThreadSafeList {
    private List<Integer> data = new ArrayList<>();
    public synchronized void addValue(int value) {
        data.add(value);
    }
    public int getValue(int index) {
        return data.get(index);
    }
    public int size() {
        synchronized(ThreadSafeList.class) {
            return data.size();
        }
    }
}
```

- A. The code does not compile because of the `size()` method.
 - B. The code compiles and is thread-safe.
 - C. The code compiles and is not thread-safe.
 - D. The code does not compile for another reason.
29. Which two method names, when filled into the `print2()` method, produce the same output as the `print1()` method? Assume the input arguments for each represent the same non-null numeric value, only accessible by a single thread at a time.

```
public static void print1(int value) {  
    System.out.println(value--);  
    System.out.println(++value);  
}  
  
public static void print2(AtomicInteger value) {  
    System.out.println(value._____);  
    System.out.println(value._____);  
}
```

- A. `decrementAndGet()` and `getAndIncrement()`
 - B. `decrementAndGet()` and `incrementAndGet()`
 - C. `getAndDecrement()` and `getAndIncrement()`
 - D. `getAndDecrement()` and `incrementAndGet()`
30. How many times does the following application print 1 at runtime?

```
package crew;  
import java.util.concurrent.*;  
import java.util.stream.*;  
public class Boat {  
    private void waitTillFinished(CyclicBarrier c) {  
        try {  
            c.await();  
            System.out.print("1");  
        } catch (Exception e) {}  
    }  
    public void row(ExecutorService service) {  
        final CyclicBarrier cb = new CyclicBarrier(5);  
        IntStream.iterate(1, i -> i+1)  
            .limit(12)  
            .forEach(i -> service.submit(() -> waitTillFinished(cb)));  
    }  
    public static void main(String[] oars) {  
        ExecutorService service = null;
```

```
    try {
        service = Executors.newCachedThreadPool();
        new Boat().row(service);
    } finally {
        service.shutdown();
    }
}
```

- A. 0
 - B. 10
 - C. 12
 - D. None of the above

31. Using the Boat class from the previous question, what is the final state of the application?

 - A. The application produces an exception at runtime.
 - B. The application terminates successfully.
 - C. The application hangs indefinitely because the ExecutorService is never shut down.
 - D. The application produces a deadlock at runtime.

32. What is the expected output of the following application?

```
package store;
import java.util.concurrent.*;
import java.util.stream.*;
public class Line {
    static BlockingDeque<Integer> queue = new LinkedBlockingDeque<>();
    public static void main(String[] participants) throws Exception {
        IntStream.iterate(1, i -> i+1).limit(5)
            .parallel()
            .forEach(s -> queue.offerLast(s,10000,TimeUnit.MILLISECONDS));
        IntStream.iterate(1, i -> 5).limit(10)
            .parallel()
            .forEach(s -> queue.pollFirst(10,TimeUnit.SECONDS));
        System.out.print(queue.size());
    }
}
```

- A. 0
 - B. A number from 0 to 5
 - C. The code does not compile.
 - D. The code compiles but throws an exception at runtime.

- 33.** Given the original array, how many of the following for statements result in an infinite loop at runtime, assuming each is executed independently?

```
List<Integer> original = new ArrayList<>(Arrays.asList(1,2,3));  
  
List<Integer> copy1 = new ArrayList<>(original);  
for(Integer q : copy1)  
    copy1.add(1);  
  
List<Integer> copy2 = new CopyOnWriteArrayList<>(original);  
for(Integer q : copy2)  
    copy2.add(2);  
  
Deque<Integer> copy3 = new ConcurrentLinkedDeque<>(original);  
for(Integer q : copy3)  
    copy3.push(3);  
  
List<Integer> copy4 = Collections.synchronizedList(original);  
for(Integer q : copy4)  
    copy4.add(4);
```

- A.** Zero
- B.** One
- C.** Two
- D.** Three

- 34.** Three of the four following options make up the requirements for performing a parallel reduction with the `collect()` method, which takes a `Collector` argument. Choose the one that is not a requirement.

- A.** The `Collector` argument is marked concurrent.
- B.** The elements of the stream implement the `Comparable` interface.
- C.** The stream is parallel.
- D.** The stream or `Collector` is marked unordered.

- 35.** Which statement about the following application is true?

```
package math;  
import java.util.concurrent.*;  
public class Fun extends RecursiveTask<Integer> {  
    final int value;  
    public Fun(int value) {  
        this.value = value;  
    }  
    @Override protected Integer compute() { // w1  
        if(value<1) {  
            return 1;
```

```
        }
        final Fun f1 = new Fun(value-1);
        final Fun f2 = new Fun(value-2);
        return f1.compute() * f2.compute();
    }
    public static void main(String... data) {
        ForkJoinPool pool = new ForkJoinPool();
        try {
            System.out.print(pool.invoke(new Fun(10)));
        } finally {
            pool.shutdown();
        }
    }
}
```

- A. The class does not compile due to line w1.
- B. The class does not compile for another reason.
- C. The application compiles and uses the fork/join framework correctly.
- D. The application compiles but does not use the fork/join framework correctly.
36. Which ExecutorService method guarantees all running tasks are stopped in an orderly fashion?
- A. shutdown()
- B. shutdownNow()
- C. halt()
- D. None of the above
37. Given the following code snippet, what statement about the values printed on lines p1 and p2 is correct?
- ```
List<Integer> db = Collections.synchronizedList(new ArrayList<>());
IntStream.iterate(1, i -> i+1).limit(5)
 .parallel()
 .map(i -> {db.add(i); return i;})
 .forEachOrdered(System.out::print); // p1
System.out.println();
db.forEach(System.out::print); // p2
```
- A. They are always the same.
- B. They are sometimes the same.
- C. They are never the same.
- D. The code will produce a ConcurrentModificationException at runtime.

- 38.** Assuming 10 seconds is enough time for all of the tasks to finish, what is the output of the following application?

```
package finance;
import java.util.concurrent.*;
public class Bank {
 static int cookies = 0;
 public synchronized void deposit(int amount) {
 cookies += amount;
 }
 public static synchronized void withdrawal(int amount) {
 cookies -= amount;
 }
 public static void main(String[] args) throws Exception {
 ExecutorService teller = Executors.newScheduledThreadPool(50);
 Bank bank = new Bank();
 for(int i=0; i<25; i++) {
 teller.submit(() -> bank.deposit(5));
 teller.submit(() -> bank.withdrawal(5));
 }
 teller.shutdown();
 teller.awaitTermination(10, TimeUnit.SECONDS);
 System.out.print(bank.cookies);
 }
}
```

- A.** 0
- B.** The code does not compile.
- C.** The result is unknown until runtime.
- D.** An exception is thrown at runtime.

- 39.** What is the output of the following application?

```
package util;
import java.util.*;
public class SearchList<T> {
 private List<T> data;
 private boolean foundMatch = false;
 public SearchList(List<T> list) {
 this.data = list;
 }
 public void exists(T value, int start, int end) {
 if(end-start<=1) {
```

```
 foundMatch = foundMatch || value.equals(data.get(start));
 } else {
 final int middle = start + (end-start)/2;
 new Thread(() -> exists(value,start,middle)).run();
 new Thread(() -> exists(value,middle,end)).run();
 }
}
public static void main(String[] a) throws Exception {
 List<Integer> data = Arrays.asList(1,2,3,4,5,6);
 SearchList<Integer> t = new SearchList<Integer>(data);
 t.exists(5, 0, data.size());
 System.out.print(t.foundMatch);
}
}
A. true
B. false
C. The code does not compile.
D. The result is unknown until runtime.
```

- 40.** How many lines of the following code snippet contain compilation errors?

```
11: ScheduledExecutorService t = Executors
12: .newSingleThreadScheduledExecutor();
13: Future result = t.execute(System.out::println);
14: t.invokeAll(null);
15: t.scheduleAtFixedRate(() -> {return;}, 5, TimeUnit.MINUTES);
A. None. The code compiles as is.
B. One
C. Two
D. Three
```



# Chapter **21**



# **Building Database Applications with JDBC**

---

**THE OCP EXAM TOPICS COVERED  
IN THIS PRACTICE TEST INCLUDE  
THE FOLLOWING:**

✓ **Building Database Applications with JDBC**

- Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations
- Identify the components required to connect to a database using the DriverManager class including the JDBC URL
- Submit queries and read results from the database including creating statements, returning result sets, iterating through the results, and properly closing result sets, statements, and connections

1. How many of `Connection`, `Driver`, and `DriverManager` are JDBC interfaces included with the JDK?
  - A. None
  - B. One
  - C. Two
  - D. Three
2. Which is found in the `java.sql` package?
  - A. `DerbyDriver`
  - B. `MySqlDriver`
  - C. `OracleDriver`
  - D. None of the above
3. What must be the first characters of a database URL?
  - A. `db,`
  - B. `db:`
  - C. `jdbc,`
  - D. `jdbc:`
4. Which is responsible for getting a connection to the database?
  - A. `Driver`
  - B. `Connection`
  - C. `Statement`
  - D. `ResultSet`
5. Which most accurately fills in the blanks in this sentence? With JDBC 3.0, the driver is \_\_\_\_\_ to contain a `java.sql.Driver` file, and the code getting a `Connection` is \_\_\_\_\_ to call `Class.forName()`.
  - A. allowed, allowed
  - B. allowed, required
  - C. required, allowed
  - D. required, required
6. Which of these obtains a `Connection`?
  - A. `Connection.getConnection(url)`
  - B. `Driver.getConnection(url)`
  - C. `DriverManager.getConnection(url)`
  - D. `new Connection(url)`
7. Which method is overloaded to allow passing a username and password?
  - A. `forName()`
  - B. `getConnection()`

- C. `getStatement()`
  - D. None of the above
8. What is the name of a concrete class that implements `Statement` and is included in the core JDK?
- A. `CallableStatement`
  - B. `PreparedStatement`
  - C. `StatementImpl`
  - D. None of the above
9. How many of the following could be valid JDBC URL formats for an imaginary driver named `magic` and a database named `box`?
- I. `jdbc:magic:127.0.0.1:1234/box`
  - II. `jdbc:magic:box`
  - III. `jdbc:magic:@127.0.0.1:1234`
- A. None
  - B. One
  - C. Two
  - D. Three
10. Which most accurately fills in the blanks in this sentence? With JDBC 4.0, the driver is \_\_\_\_\_ to contain a `java.sql.Driver` file, and the code getting a `Connection` is \_\_\_\_\_ to call `Class.forName()`.
- A. allowed, allowed
  - B. allowed, required
  - C. required, allowed
  - D. required, required
11. Which pair of statements is true when requesting a `ResultSet` to be both scroll sensitive and updatable?
- I. The scroll sensitive option is passed as a parameter before the updatable parameter.
  - II. The updatable option is passed as a parameter before the scroll sensitive parameter.
  - III. If these options are not available, the database driver returns a `ResultSet` with different options.
  - IV. If these options are not available, the database driver throws a `SQLException`.
- A. I, III
  - B. I, IV
  - C. II, III
  - D. II, IV

12. Suppose the blue database exists and we are using a JDBC 4.0 driver. Which is the outcome of this code?

```
String url = "jdbc:derby:blue";
Class.forName(url);
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select count(*) from sky")) {
 System.out.println(rs.getInt(1));
}
```

- A. It runs successfully and prints the number of rows in the sky table.
  - B. It throws a `ClassNotFoundException`.
  - C. It throws a `SQLException`.
  - D. It does not compile.
13. Fill in the blanks: There are \_\_\_\_\_ `ResultSet` concurrency modes, and drivers are required to support \_\_\_\_\_.
- A. two, both
  - B. two, one of them
  - C. three, two of them
  - D. three, all of them
14. What is the output when run with a JDBC 4.0 driver if the clowns database exists and contains an empty clowns table?

```
String url = "jdbc:derby:clowns";
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select count(*) from clowns")) {
 System.out.println(rs.getInt(1));
}
```

- A. 0
  - B. 1
  - C. The code does not compile.
  - D. The code compiles but throws an exception at runtime.
15. Consider the three methods `execute()`, `executeQuery()`, and `executeUpdate()`. Fill in the blanks: \_\_\_\_\_ of these methods is/are allowed to run a `DELETE` SQL statement while \_\_\_\_\_ of these methods is/are allowed to run an `UPDATE` SQL statement.
- A. One, one
  - B. One, two
  - C. Two, one
  - D. Two, two

16. Assuming the `clowns` database exists and contains one empty table named `clowns`, what is the output of the following when run using a JDBC 4.0 driver?

```
import java.sql.*;

public class EmptyTable {
 public static void main(String[] args) throws SQLException { // s1
 String url = "jdbc:derby:clowns";
 try (Connection conn = new Connection(url); // s2
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select * from clowns")) {
 if (rs.next())
 System.out.println(rs.getString(1));
 }
 }
}
```

- A. The code terminates successfully without any output.
  - B. The code does not compile due to line s1.
  - C. The code does not compile due to line s2.
  - D. None of the above
17. Which are valid `ResultSet` types?
- I. `TYPE_BACKWARD_ONLY`
  - II. `TYPE_FORWARD_ONLY`
  - III. `TYPE_REVERSE_ONLY`
- A. II
  - B. I and II
  - C. II and III
  - D. None of the above
18. Given the table `books` in the figure and a `ResultSet` created by running the following SQL statement, which option prints the value 379?

| title<br>character varying(255) | num_pages<br>integer |
|---------------------------------|----------------------|
| OCA                             | 379                  |
| OCP                             | 669                  |

```
select * from cert where title = 'OCA'

A. System.out.println(rs.getInt(1));
B. System.out.println(rs.getInt(2));
C. System.out.println(rs.getInteger(1));
D. System.out.println(rs.getInteger(2));
```

19. Given the table books in the previous question and a `ResultSet` created by running this SQL statement, which option prints OCP?

```
select title from cert where num_pages > 500
```

- A. `System.out.println(rs.getString());`
- B. `System.out.println(rs.getString("0"));`
- C. `System.out.println(rs.getString("1"));`
- D. `System.out.println(rs.getString("title"));`

20. Assume the database exists with all referenced table and column names. Which is a true statement when a JDBC 4.0 driver is used?

```
String url = "jdbc:derby:precipitation";
try (Connection conn = DriverManager.getConnection(url)) {
 Statement stmt = conn.createStatement(
 ResultSet.CONCUR_READ_ONLY,
 ResultSet.TYPE_SCROLL_INSENSITIVE);
 ResultSet rs = stmt.execute(
 "select total from precip where type = 'rain'") {
 System.out.println(rs.getString("total"));
 }
}
```

- A. There is a compiler error on the line of code that creates the `Statement`.
- B. There is a compiler error on the line of code that creates the `ResultSet`.
- C. This code compiles and runs without error.
- D. This code throws a `SQLException` at runtime.

21. Which resources have their `close()` method called when this code runs?

```
public static void runQuery(Connection conn) throws SQLException{
 try (Statement stmt = conn.createStatement()) {
 ResultSet rs = stmt.executeQuery("select * from clowns");
 rs.next();
 }
}
```

- A. No `close()` methods are called.
- B. Only `Statement`
- C. Only `Statement` and `Connection`
- D. Only `Statement` and `ResultSet`

22. Which statement is true about the JDBC core classes?

- A. `Driver` is an implementation of `DriverManager`.
- B. A general `Connection` implementation is included in the JDK.

- C. A Statement automatically starts in auto-commit mode.
  - D. A ResultSet automatically starts pointing to the first row of data.
23. Which of the following is required in a database driver implementation?
- A. A file named `jdbc.driver`
  - B. A file named `java.sql.Driver`
  - C. At least one implementation of the `Connection` interface
  - D. None of the above
24. Given that the `people` table has 10 rows, what is the result of the following when using a driver that supports a scroll sensitive `ResultSet`?
- ```
try (Connection conn = DriverManager.getConnection(url);
      Statement stmt = conn.createStatement(
          ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
      ResultSet rs = stmt.executeQuery("select count(*) from people")) {
    rs.next();
    rs.absolute(0); // q1
    System.out.print(rs.getInt(1)); // q2
}
```
- A. 10
 - B. The code does not compile.
 - C. Line q1 throws a `SQLException`.
 - D. Line q2 throws a `SQLException`.
25. Given a scrollable updatable `ResultSet` that contains the following, what does the code snippet output?

color character varying(255)	count integer
black	20
blue	5
red	0

```
rs = stmt.executeQuery("select * from pens");
rs.afterLast();
rs.previous();
rs.updateInt(2, 10);
rs.updateRow();

rs = stmt.executeQuery("select * from pens where color = 'red'");
while (rs.next())
    System.out.println(rs.getInt(2));
```

A. 0

B. 10

C. The code does not compile.

D. The code compiles but throws an exception at runtime.

26. Given a scrollable updatable ResultSet represented by the image in the previous question, what does the code snippet output?

```
rs = stmt.executeQuery("select * from pens");
rs.afterLast();
while(rs.prev())
    rs.updateInt(2,0);
rs = stmt.executeQuery("select * from pens where color = 'black'");
while (rs.next())
    System.out.println(rs.getInt(2));
```

A. 0

B. 20

C. The code does not compile.

D. The code compiles but throws an exception at runtime.

27. How many rows are added to the colors table from running the following?

```
try (Connection conn = DriverManager.getConnection(url);
     Statement stmt = conn.createStatement()) {
    conn.setAutoCommit(false);
    stmt.executeUpdate("insert into colors values ('red')");
    stmt.executeUpdate("insert into colors values ('blue')");
    conn.commit();
    conn.setAutoCommit(true);
    stmt.executeUpdate("insert into colors values ('green')");
}
```

A. None

B. One

C. Two

D. Three

28. Which is true if the clowns database exists and contains an empty clowns table?

```
String url = "jdbc:derby:clowns";
try (Connection conn = DriverManager.getConnection(url);
     Statement stmt = conn.createStatement());
     ResultSet rs = stmt.executeQuery("select count(*) from clowns")) {
```

```
    rs.next();    // r1
    System.out.println(rs.getInt(1));    // r2
}
```

- A.** The code compiles and runs without error.
 - B.** The code does not compile.
 - C.** The code compiles but throws an exception at runtime on line r1.
 - D.** The code compiles but throws an exception at runtime on line r2.
- 29.** Suppose the `ResultSet` is scrollable and contains 10 rows with the values 1–10 respectively. What is the output of the following?
- ```
5: rs.absolute(0);
6: rs.relative(5);
7: rs.relative(-10);
8: rs.relative(5);
9: System.out.print(rs.getInt(1));
```

- A.** 4
- B.** 5
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

- 30.** Suppose the blue database does not exist and we are using a JDBC 4.0 driver. Which is the outcome of this code?

```
String url = "jdbc:derby:blue";
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select count(*) from sky")) {
 System.out.println(rs.getInt(1));
}
```

- A.** It runs successfully and prints the number of rows in the sky table.
  - B.** It throws a `ClassNotFoundException`.
  - C.** It throws a `SQLException`.
  - D.** It does not compile.
- 31.** What is the most likely outcome of this code if the people table is empty?

```
6: Statement stmt = conn.createStatement();
7: ResultSet rs1 = stmt.executeQuery("select * from people");
8: ResultSet rs2 = stmt.executeQuery("select * from people");
9: System.out.println(rs1.next() + " " + rs2.next());
```

- A.** It prints `false false`.
- B.** It prints `true false`.
- C.** It does not terminate.
- D.** It throws a `SQLException`.

- 32.** How many rows are added to the colors table from running the following?

```
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement()) {
 conn.setAutoCommit(false);
 stmt.executeUpdate("insert into colors values ('red')");
 stmt.executeUpdate("insert into colors values ('blue')");
 conn.rollback();
 conn.setAutoCommit(true);
 stmt.executeUpdate("insert into colors values ('green')");
}
```

- A.** None
- B.** One
- C.** Two
- D.** Three

- 33.** Assuming the clowns database exists and contains one empty table named clowns, what is the output of the following when run using a JDBC 4.0 driver?

```
import java.sql.*;
public class EmptyTable {
 public static void main (String[] args) throws SQLException { // s1
 String url = "jdbc:derby:clowns";
 try (Connection conn = DriverManager.getConnection(url); // s2
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select * from clowns")) {
 if (rs.next())
 System.out.println(rs.getString(1));
 }
 }
}
```

- A.** The code terminates successfully without any output.
- B.** The code does not compile due to line s1.
- C.** The code does not compile due to line s2.
- D.** None of the above

- 34.** Given the following code snippet and the table below, what is the output of the following when using a driver that supports a scroll sensitive ResultSet?

```
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement(
 ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

```
ResultSet rs = stmt.executeQuery(
 "select * from people order by last_name asc")) {
 rs.afterLast();
 rs.next();
 rs.next();
 rs.previous();
 rs.previous();
 System.out.println(rs.getString(1));
}
```

| first_name<br>character varying(255) | last_name<br>character varying(255) |
|--------------------------------------|-------------------------------------|
| Jeanne                               | Boyarsky                            |
| Janeice                              | DelVecchio                          |
| Elena                                | Felder                              |
| Scott                                | Selikoff                            |

- A.** Janeice
  - B.** Elena
  - C.** Scott
  - D.** The code throws a SQLException at runtime.
- 35.** Given the following code snippet and the table in question 34, what is the output of the following when using a driver that supports a scroll sensitive ResultSet?

```
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement(
 ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
 ResultSet rs = stmt.executeQuery(
 "select * from people order by last_name asc")) {
 rs.absolute(-1);
 System.out.print(rs.getString(1));
 System.out.print(" ");
 rs.absolute(1);
 System.out.print(rs.getString(1));
}
```

- A.** Jeanne Scott
- B.** Scott Jeanne
- C.** The code compiles, but the output is neither of these.
- D.** The code throws a SQLException at runtime.

- 36.** Given the following code snippet and the table in question 34, what is the output of the following?

```
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery(
 "select * from people order by last_name asc")) {
 rs.next();
 rs.next();
 System.out.print(rs.getString(1));
 System.out.print(" ");
 rs.absolute(1);
 System.out.print(rs.getString(1));
}
```

- A.** Jeanne Scott
  - B.** Scott Jeanne
  - C.** The code compiles, but the output is neither of these.
  - D.** The code throws a SQLException at runtime.
- 37.** Given that the people table has 10 rows, what is the result of the following when using a driver that supports a scroll sensitive Resultset?

```
String sql = "select count(*) from people";
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery(sql,
 ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
 rs.next();
 rs.absolute(0); // q1
 System.out.print(rs.getInt(1)); // q2
}
```

- A.** 10
  - B.** The code does not compile.
  - C.** Line q1 throws a SQLException.
  - D.** Line q2 throws a SQLException.
- 38.** How many rows are added to the colors table from running the following?

```
try (Connection conn = DriverManager.getConnection(url);
 Statement stmt = conn.createStatement()) {
 conn.setAutoCommit(false);
 stmt.executeUpdate("insert into colors values ('red')");
 stmt.executeUpdate("insert into colors values ('blue')");
```

```
conn.rollback();
conn.setAutoCommit(true);
stmt.executeUpdate("insert into colors values ('green')");
conn.rollback();
}
```

- A.** None  
**B.** One  
**C.** Two  
**D.** Three
- 39.** Suppose the `ResultSet` is scrollable and contains 10 rows. How many times does the following print true?
- ```
16: System.out.println(rs.absolute(-2));
17: System.out.println(rs.relative(-1));
18: System.out.println(rs.beforeFirst());
19: System.out.println(rs.relative(5));
```
- A.** Two
B. Three
C. Four
D. None of the above
- 40.** What is the correct order to close database resources?
- A.** Connection then Statement then ResultSet
B. ResultSet then Statement then Connection
C. Statement then Connection then ResultSet
D. Statement then ResultSet then Connection

Chapter 22



Localization

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

✓ **Localization**

- Read and set the locale by using the Locale object
- Create and read a Properties file
- Build a resource bundle for each locale and load a resource bundle in an application

1. Which is not a locale?
 - A. Cultural region
 - B. Geographical region
 - C. Political region
 - D. Time zone region
2. When localizing an application, which type of data varies in presentation depending on locale?
 - A. Currencies
 - B. Dates
 - C. Both
 - D. Neither
3. How do you find out the locale of the running program?
 - A. `Locale.get("default")`
 - B. `Locale.get(Locale.DEFAULT)`
 - C. `Locale.getDefault()`
 - D. None of the above
4. If your application has all town names in a single file named `strings.properties`, what is the most specific process that has been implemented?
 - A. Internationalization
 - B. Localization
 - C. Specialization
 - D. Translation
5. Which interfaces does `Properties` implement?
 - I. `Hashtable`
 - II. `HashMap`
 - III. `Map`
 - A. III
 - B. I and II
 - C. I and III
 - D. II and III
6. Which filename extension can hold a `String` property value in a resource bundle?
 - A. `.java`
 - B. `.properties`
 - C. Both of the above
 - D. Neither of the above

7. How long will the effects of calling `Locale.setDefault()` be active assuming no other calls to that method are made?
 - A. Until the end of the method
 - B. Until the program exits
 - C. Until the next reboot of the computer
 - D. None of the above. It persists even past a reboot.
8. Given this properties file used to load the `Properties` object `props` and this code snippet, what is the output?

```
mystery=bag
type=paper

18: System.out.print(props.getProperty("mystery"));
19: System.out.print(" ");
20: System.out.print(props.getProperty("more"));
```

- A. bag
 - B. bag null
 - C. bag ?
 - D. This code throws a runtime exception on line 20.
9. Fill in the blanks to complete the following code for a Java resource bundle.

```
public class Flights_en extends _____ {
    protected Object[][] _____() {
        return new Object[0][0];
    }
}
```

- A. `JavaResourceBundle`, `getContents`
 - B. `JavaResourceBundle`, `getProperties`
 - C. `ListResourceBundle`, `getContents`
 - D. `ListResourceBundle`, `getProperties`
10. Which of the following shows a valid `Locale` format for the language Hindi and the country India?
 - A. `hi_IN`
 - B. `HI_in`
 - C. `in_HI`
 - D. `IN_hi`

11. If the key `purple` is in all four of these files, which file will the following code use for the resource bundle?

```
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle("Colors");
rb.getString("purple");
```

- A. `Colors.class`
- B. `Colors.properties`
- C. `Colors_en_US.class`
- D. `Colors_en_US.properties`

12. What is the output of the following?

```
package counter;
import java.util.*;

public class CountResource extends ListResourceBundle {
    private int count = 0;

    @Override
    protected Object[][] getContents() {
        return new Object[][] { { "count", count++ } };
    }

    public static void main(String[] args) {
        ResourceBundle rb = ResourceBundle.getBundle("counter.CountResource");
        System.out.println(rb.getObject("count") + " " + rb.getObject("count"));
    }
}
```

- A. 0 0
- B. 0 1
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

13. Which of the following shows a valid `Locale` format?

- A. `en`
- B. `US`
- C. `US_en`
- D. None of the above

14. Which filename extension can define a `LocalDateTime` property value in a resource bundle?

- A. `.java`
- B. `.properties`
- C. Both of the above
- D. Neither of the above

15. What happens if you run this code with no resource bundles available?

```
 ResourceBundle rb = ResourceBundle.getBundle("ghost.None");
```

- A. It creates a resource bundle with no key/value pairs.
- B. It runs without issue because the resource bundle is only searched for when getting the first key.
- C. It throws a `MissingResourceException`.
- D. None of the above

16. What is the result of the following?

```
 Map<String, String> map = new TreeMap<>();  
 map.put("tool", "hammer");  
 map.put("problem", "nail");  
  
 Properties props = new Properties();           // p1  
 map.forEach((k,v) -> props.put(k, v));      // p2  
  
 String t = props.get("tool");                  // p3  
 String n = props.get("nail");  
 System.out.println(t + " " + n);
```

- A. hammer nail
- B. The code does not compile due to line p1.
- C. The code does not compile due to line p2.
- D. The code does not compile due to line p3.

17. Which filename extension can hold a `LocalDateTime` property key in a resource bundle?

- A. `.java`
- B. `.properties`
- C. Both of the above
- D. Neither of the above

18. If the key `purple` is in all four of these files, which file will the following code use for the resource bundle?

```
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle("Colors");
rb.getString("purple");
```

- A. `Colors_en.class`
- B. `Colors_en.properties`
- C. `Colors_US.class`
- D. `Colors_US.properties`

19. Fill in the blank so the code correctly compiles and creates a `Locale`.

```
Locale loc = Locale._____;
```

- A. `get("Italian")`
- B. `get(Locale.ITALIAN)`
- C. `getLocale("Italian")`
- D. None of the above

20. What is the result of the following?

```
Map<Object, Object> map = new TreeMap<>();
map.put("tool", "hammer");
map.put("problem", "nail");

Properties props = new Properties();      // p1
map.forEach((k,v) -> props.put(k, v));  // p2

String t = props.getProperty("tool");      // p3
String n = props.getProperty("problem");
System.out.println(t + " " + n);
```

- A. hammer nail
- B. The code does not compile due to line p1.
- C. The code does not compile due to line p2.
- D. The code does not compile due to line p3.

21. What is the output of the following?

```
1: package keyboard;
2: import java.util.*;
3:
4: public class Type {
5:     protected Object[][] getContents() {
6:         return new Object[][] { { "keys", new ArrayList<String>() } };
7:     }
}
```

```
8:     public static void main(String[] args) {  
9:         ResourceBundle rb = ResourceBundle.getBundle("keyboard.Type");  
10:        List<String> keys = (List) rb.getObject("keys");  
11:        keys.add("q");  
12:        keys.add("w");  
13:        keys.add("e");  
14:        keys.add("r");  
15:        keys.add("t");  
16:        keys.add("y");  
17:        System.out.println(((List) rb.getObject("keys")).size());  
18:    }  
19: }
```

- A.** 0
- B.** 6
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

22. How many lines does the following print out?

```
3:     Locale.setDefault(Locale.KOREAN);  
4:     System.out.println(Locale.getDefault());  
5:     Locale.setDefault(new Locale("en", "AU"));  
6:     System.out.println(Locale.getDefault());  
7:     Locale.setDefault(new Locale("EN"));  
8:     System.out.println(Locale.getDefault());
```

- A.** None; it does not compile.
- B.** One followed by an exception
- C.** Two followed by an exception
- D.** All three

23. Given this properties file used to load the Properties object `props` and this code snippet, what is the output?

```
mystery=bag  
type=paper  
  
18: System.out.print(props.getProperty("mystery", "?"));  
19: System.out.print(" ");  
20: System.out.print(props.getProperty("more", "?"));
```

- A.** bag
- B.** bag null
- C.** bag ?
- D.** This code throws a runtime exception on line 20.

24. Given the following four properties files, what does this code print?

Cars_en.properties	Cars_fr.properties
engine=engine horses=241	engine=moteur
Cars_en_US.properties	Cars_fr_FR.properties
country=US	country=France

```
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle(
    "Cars", new Locale("fr", "FR"));
System.out.println(rb.getString("country") + " "
    + rb.getString("engine"));
```

- A. France engine
- B. France moteur
- C. France null
- D. The code throws an exception at runtime.

25. Given the four properties files in question 24, what does this code print?

```
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle(
    "Cars", new Locale("fr", "CA"));
System.out.println(rb.getString("engine") + " "
    + rb.getString("horses"));
```

- A. engine 241
- B. moteur 241
- C. moteur null
- D. The code throws an exception at runtime.

26. Given the four properties files in question 24, what does this code print?

```
Locale.setDefault(new Locale("fr", "CA"));
ResourceBundle rb = ResourceBundle.getBundle(
    "Cars", new Locale("en", "CA"));
```

```
System.out.println(rb.getString("engine") + " "
    + rb.getString("horses"));
```

- A. engine 241
 - B. moteur 241
 - C. moteur null
 - D. The code throws an exception at runtime.
27. Which statement about `ListResourceBundle` is true?
- A. A disadvantage over properties files is that you cannot create the value at runtime.
 - B. `Names-fr.java` is a bad implementation class name.
 - C. Values of type `Integer` are not allowed.
 - D. None of the above
28. If the key `red` is in all four of these files, which file will the following code use for the resource bundle?
- ```
Locale loc = new Locale("zh", "CN");
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle("Colors", loc);
rb.getString("red");
```
- A. Colors\_CN.properties
  - B. Colors\_en.properties
  - C. Colors\_US.properties
  - D. Colors\_zh.properties
29. What is the output of the following?

```
package counter;
import java.util.*;

public class PropertyCounter extends ListResourceBundle {
 private int count = 0;
 @Override
 protected Object[][] getContents() {
 return new Object[][] {{ "count", new PropertyCounter() }};
 }
 public int getCount() {
 return count++;
}
```

```
public static void main(String[] args) {
 ResourceBundle rb = ResourceBundle.getBundle("counter.PropertyCounter");
 PropertyCounter obj = (PropertyCounter) rb.getObject("count");
 System.out.println(obj.getCount() + " " + obj.getCount());
}
}
```

- A.** 0 0
- B.** 0 1
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

**30.** What is the output of the following?

```
1: package keyboard;
2: import java.util.*;
3:
4: public class Type extends ListResourceBundle {
5: protected Object[][] getContents() {
6: return new Object[][] { { "keys", new ArrayList<String>() } };
7: }
8: public static void main(String[] args) {
9: ResourceBundle rb = ResourceBundle.getBundle("Type");
10: List<String> keys = (List) rb.getObject("keys");
11: keys.add("q");
12: keys.add("w");
13: keys.add("e");
14: keys.add("r");
15: keys.add("t");
16: keys.add("y");
17: System.out.println((List) rb.getObject("keys")).size());
18: }
19: }
```

- A.** 0
- B.** 6
- C.** The code does not compile.
- D.** The code compiles but throws an exception at runtime.

31. Given the following snippets of resource bundles from when we compiled the application, what is the result of the following?

|                                                             |                         |
|-------------------------------------------------------------|-------------------------|
| Buggy.java                                                  | Buggy.properties        |
| getContents:<br>return new Object[][]<br>{{"wheels", "4"}}; | color=white             |
| Buggy_en.java                                               | Buggy_en.properties     |
| getContents:<br>return<br>new Object[0][0];                 | wheels=6<br>color=black |

```
Locale.setDefault(new Locale("en", "US"));
 ResourceBundle rb = ResourceBundle.getBundle("Buggy");
 System.out.println(rb.getString("wheels"));
```

- A. null
  - B. 4
  - C. 6
  - D. The code throws an exception at runtime.
32. Given the snippets of resource bundles in question 31 from when we compiled the application, what is the result of the following?
- ```
Locale.setDefault(new Locale("en"));
 ResourceBundle rb = ResourceBundle.getBundle("Buggy");
 System.out.println(rb.getString("color"));
```
- A. null
 - B. black
 - C. white
 - D. The code throws an exception at runtime.
33. Given the snippets of resource bundles in question 31 from when we compiled the application, what is the result of the following?

```
Locale.setDefault(new Locale("zh"));
 ResourceBundle rb = ResourceBundle.getBundle("Buggy");
 System.out.println(rb.getString("wheels"));
```

- A. null
- B. 4
- C. 6
- D. The code throws an exception at runtime.

- 34.** If the key `red` is in all three of these files, which file will the following code use for the resource bundle?

```
Locale loc = new Locale("zh", "CN");
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle("Colors", loc);
rb.getString("red");
```

- A.** `Colors_en.properties`
- B.** `Colors.properties`
- C.** `Red_en.properties`
- D.** None of the above. It will throw a `MissingResourceException`.

- 35.** Which can fill in the blank in this code?

```
Properties props = new Properties();
props._____("x");
```

- A.** `get`
- B.** `getProperty`
- C.** Both of the above
- D.** Neither of the above

- 36.** What is the output of the following?

```
1:  package keyboard;
2:  import java.util.*;
3:
4:  public class Type extends ListResourceBundle {
5:      protected Object[][] getContents() {
6:          return new Object[][] { { "keys", new ArrayList<String>() } };
7:      }
8:      public static void main(String[] args) {
9:          ResourceBundle rb = ResourceBundle.getBundle("keyboard.Type");
10:         List<String> keys = (List) rb.getObject("keys");
11:         keys.add("q");
12:         keys.add("w");
13:         keys.add("e");
14:         keys.add("r");
15:         keys.add("t");
16:         keys.add("y");
17:         System.out.println(((List) rb.getObject("keys")).size());
18:     }
19: }
```

- A. 0
- B. 6
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

37. What is the result of the following?

```
Map<String, String> map = new TreeMap<>();
map.put("tool", "hammer");
map.put("problem", "nail");

Property props = new Property();           // p1
map.forEach((k,v) -> props.put(k, v));    // p2

String t = props.getProperty("tool");       // p3
String n = props.getProperty("nail");
System.out.println(t + " " + n);
```

- A. hammer nail
- B. The code does not compile due to line p1.
- C. The code does not compile due to line p2.
- D. The code does not compile due to line p3.

38. What is the output of the following?

```
package counter;
import java.util.*;

public class CountResource extends ListResourceBundle {
    private int count = 0;

    @Override
    protected Object[][][] getContents() {
        return new Object[][][] { { "count", count++ } };
    }
    public static void main(String[] args) {
        ResourceBundle rb = new ResourceBundle("counter.CountResource");
        System.out.println(rb.getObject("count") + " " + rb.getObject("count"));
    }
}
```

- A. 0 0
- B. 0 1
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.

39. Given the following two properties files in the pod package, what does the following class output?

```
pod.container.properties
name=generic
number=2

pod.container_en.properties
name=Docker
type=container

package pod;
import java.util.*;
public class WhatKind {
    public static void main(String[] args) {
        Locale.setDefault(new Locale("en"));
        ResourceBundle rb = ResourceBundle.getBundle("pod.container");
        String name = rb.getString("name");
        String type = rb.getString("type");
        System.out.println(name + " " + type);    }
}
```

- A. Docker container
 - B. generic container
 - C. generic null
 - D. None of the above
40. If the key red is in all three of these files, which file will the following code use for the resource bundle?

```
Locale loc = new Locale("zh", "CN");
Locale.setDefault(new Locale("en", "US"));
ResourceBundle rb = ResourceBundle.getBundle("Colors", loc);
rb.getString("red");



- A. Colors_EN.properties
- B. Colors_ZH.properties
- C. Red_EN.properties
- D. None of the above. It will throw a MissingResourceException.

```

Chapter 23



OCP Practice Exam

This chapter contains 85 questions and is designed to simulate a real OCP exam. While previous chapters were focused on a specific set of objectives, this chapter covers all of the objectives on the exam. We recommend you take this exam only after you score well on the questions in the individual chapters.

For this chapter, you should try to simulate the real exam experience as much as possible. This means setting aside 150 minutes of uninterrupted time to complete the test, as well as not looking at any reference material while taking the exam. If you don't know an answer to a question, complete it as best you can and move on to the next question, just as you would on a real exam.

Remember, the exam permits writing material, such as a whiteboard. If you do not have a whiteboard handy, you can just use blank sheets of paper and a pencil. If you do well on this test, then you are hopefully ready to take the real exam. With that said, good luck!

1. Suppose the `ResultSet` is scrollable and contains 10 rows with values numbered from 1 to 10 in ascending order. What is true about the following? (Choose two.)

```
12: rs.beforeFirst();
13: System.out.println(rs.relative(5));
14: System.out.println(rs.getInt(1));
15: System.out.println(rs.relative(-10));
16: System.out.println(rs.getInt(1));
17: System.out.println(rs.relative(5));
18: System.out.print(rs.getInt(1));
```

- A. It outputs true once.
 - B. It outputs true twice.
 - C. It outputs true three times.
 - D. It completes without throwing an exception.
 - E. It throws an exception at runtime.
2. What is the result of the following?

```
public class PiDay {
    public static void main(String[] args) {
        LocalDateTime pi = LocalDateTime.of(2017, 3, 14, 1, 59);
        DateTimeFormatter formatter = DateTimeFormatter
            .ofPattern("m.ddhhMM");
        System.out.println(formatter.format(pi));
    }
}
```

- A. 3.011459
 - B. 3.140159
 - C. 59.011459
 - D. 59.140103
 - E. The code does not compile.
 - F. The code compiles but throws an exception at runtime.
3. Which of the following are valid lambda expressions? (Choose three.)
- A. () -> {}
 - B. (Double adder) -> {int y; System.out.print(add); return add;}
 - C. (Long w) -> {Long w=5; return 5;}
 - D. (int count, vote) -> count*vote
 - E. dog -> dog
 - F. name -> {name.toUpperCase()}

4. What is the result of compiling and running the following application?

```
package names;
import java.util.*;
import java.util.function.*;
interface ApplyFilter {
    void filter(List<String> input);
}
public class FilterBobs {
    static Function<String, String> first = s ->
        {System.out.println(s); return s;};
    static Predicate second = t -> "bob".equalsIgnoreCase(t);
    public void process(ApplyFilter a, List<String> list) {
        a.filter(list);
    }
    public static void main(String[] contestants) {
        final List<String> people = new ArrayList<>();
        people.add("Bob");
        people.add("bob");
        people.add("Jennifer");
        people.add("Samantha");
        final FilterBobs f = new FilterBobs();
        f.process(q -> {
            q.removeIf(second);
            q.forEach(first);
        }, people);
    }
}
```

- A. It prints two lines.
- B. It prints three lines.
- C. One line of code does not compile.
- D. Two lines of code do not compile.
- E. Three lines of code do not compile.
- F. The code compiles but prints an exception at runtime.

5. What is the output of the following?

```
public class Compete {
    public static void main(String[] args) {
        Stream<Integer> is = Stream.of(8, 6, 9);
        Comparator<Integer> c = (a, b) -> b - a; // r1
        is.sorted(c).forEach(System.out::print); // r2
    }
}
```

- A. 689
 - B. 986
 - C. The code does not compile because of line r1.
 - D. The code does not compile because of line r2.
 - E. The code does not compile due to another line.
 - F. The code compiles but throws an exception at runtime.
6. What is the output of the following application?

```
package tax;

public class Accountant {
    class AddingException extends Exception {};
    class DividingException extends Exception {};
    class UnexpectedException extends RuntimeException {};
    public void doTaxes() throws Throwable {
        try {
            throw new IllegalStateException();
        } catch (AddingException | DividingException e) { // p1
            System.out.println("Math Problem");
        } catch (UnexpectedException | Exception f) { // p2
            System.out.println("Unknown Error");
            throw f;
        }
    }
    public static void main(String[] numbers) throws Throwable {
        try {
            new Accountant().doTaxes();
        } finally {
            System.out.println("All done!");
        }
    }
}
```

- A. Math Problem
- B. Unknown Problem
- C. Unknown Problem followed by All done!
- D. The code does not compile solely due to line p1.
- E. The code does not compile solely due to line p2.
- F. The code does not compile due to lines p1 and p2.

7. What is the output of the following application?

```
1: package drawing;
2: interface HasHue {String getHue();}
3: enum COLORS implements HasHue {
4:     red {
5:         public String getHue() {return "FF0000";}
6:     }, green {
7:         public String getHue() {return "00FF00";}
8:     }, blue {
9:         public String getHue() {return "0000FF";}
10:    }
11: }
12: class Book {
13:     static void main(String[] pencils) {}
14: }
15: final public class ColoringBook extends Book {
16:     final void paint(COLORS c) {
17:         System.out.print("Painting: "+c.getHue());
18:     }
19:     final public static void main(String[] crayons) {
20:         new ColoringBook().paint(COLORS.green);
21:     }
22: }
```

- A. Painting: 00FF00
- B. One line of code does not compile.
- C. Two lines of code do not compile.
- D. Three lines of code do not compile.
- E. The code compiles but prints an exception at runtime.
- F. None of the above

8. How many of the following can fill in the blank to make this code compile?

```
public boolean isItMyBirthday(LocalDateTime dateTime) {
    _____
    return now.getMonth() == dateTime.getMonth()
        && now.getDayOfMonth() == dateTime.getDayOfMonth();
}
```

- I. `LocalDate now = LocalDate.now();`
 - II. `LocalDate now = new LocalDate();`
 - III. `LocalDateTime now = LocalDateTime.now();`
 - IV. `LocalDateTime now = new LocalDateTime();`
 - V. `ZonedDateTime now = ZonedDateTime.now();`
 - VI. `ZonedDateTime now = new ZonedDateTime();`
- A. None
 - B. One
 - C. Two
 - D. Three
 - E. Four
 - F. Five
9. Which two can independently fill in the blank to output No dessert today? (Choose two.)
- ```
import java.util.*;
public class Dessert {
 public static void main(String[] yum) {
 eatDessert(Optional.empty());
 }
 private static void eatDessert(Optional<String> opt) {
 System.out.println(opt._____);
 }
}
```
- A. `get("No dessert today")`
  - B. `get(() -> "No dessert today")`
  - C. `orElse("No dessert today")`
  - D. `orElse(() -> "No dessert today")`
  - E. `orElseGet("No dessert today")`
  - F. `orElseGet(() -> "No dessert today")`
10. What is the output of the following?

```
public class InitOrder {
 { System.out.print("1"); }
 static { System.out.print("2"); }

 public InitOrder() {
 System.out.print("3");
 }
}
```

```
public static void callMe() {
 System.out.print("4");
}
public static void main(String[] args) {
 callMe();
 callMe();
 System.out.print("5");
}
```

- A.** 1223445
- B.** 2445
- C.** 22445
- D.** 223445
- E.** 2233445
- F.** None of the above

**11.** Which of the following cannot be instantiated directly by the caller using the constructor?

- I.** Locale
  - II.** Properties
  - III.** ResourceBundle
- A.** I
  - B.** II
  - C.** III
  - D.** I, II
  - E.** I, III
  - F.** II, III

**12.** What design pattern or principle is used in this class?

```
public class Daffodil {
 int height;
 public Daffodil(int height) {
 this.height = height;
 }
 public int getHeight() {
 return height;
 }
}
```

- A.** Encapsulation
  - B.** Immutability
  - C.** Singleton
  - D.** Both A and B
  - E.** None of the above
- 13.** What is the output of the following application? Assume the file system is available and able to be written to and read from.

```
package boat;
import java.io.*;
public class Cruise {
 private int numPassengers = 1;
 private transient String schedule = "NONE";
 {numPassengers = 2;}
 public Cruise() {
 this.numPassengers = 3;
 this.schedule = "Tropical Island";
 }
 public static void main(String... passengers) throws Exception {
 try (ObjectOutputStream o = new ObjectOutputStream(
 new FileOutputStream("ship.txt"))) {
 Cruise c = new Cruise();
 c.numPassengers = 4;
 c.schedule = "Casino";
 o.writeObject(c);
 }
 try (ObjectInputStream i = new ObjectInputStream(
 new FileInputStream("ship.txt"))) {
 Cruise c = i.readObject();
 System.out.print(c.numPassengers+", "+c.schedule);
 }
 }
}
```

- A.** 2,NONE
- B.** 3,null
- C.** 4,Casino
- D.** 4,null
- E.** The class does not compile.
- F.** The class compiles but throws an exception at runtime.

- 14.** Which of the following are JDBC interfaces in the `java.sql` package?
- I.** Driver
  - II.** DriverManager
  - III.** Query
  - IV.** ResultSet
- A.** I, III
  - B.** I, IV
  - C.** II, III
  - D.** II, IV
  - E.** I, II, III
  - F.** I, II, IV
- 15.** Which of the following lambda expressions can be passed to a method that takes `IntUnaryOperator` as an argument? (Choose three.)
- A.** `v -> {System.out.print("Hello!"); return 2%1;}`
  - B.** `(Integer w) -> w.intValue()`
  - C.** `(int j) -> (int)30L`
  - D.** `(int q) -> q/3.1`
  - E.** `(long x) -> (int)x`
  - F.** `z -> z`
- 16.** Which of the following statements about `InputStream` and `Reader` are correct? (Choose two.)
- A.** One contains a `read()` method that returns a byte value, while the other contains a `read()` method that returns an `int` value.
  - B.** Only one of them contains a `flush()` method.
  - C.** Only one of them contains a `skip()` method.
  - D.** They are both abstract classes.
  - E.** They are both interfaces.
  - F.** They can both be used to read character data.
- 17.** Fill in the blank so this code prints -1.
- ```
LocalDate xmas = LocalDate.of(2017, 12, 25);
LocalDate blackFriday = LocalDate.of(2017, 11, 24);
long monthsLeft = ChronoUnit.MONTHS._____;
System.out.println(monthsLeft);
```
- A.** `between(blackFriday, xmas)`
 - B.** `between(xmas, blackFriday)`
 - C.** `minus(blackFriday, xmas)`
 - D.** `minus(xmas, blackFriday)`
 - E.** `plus(blackFriday, xmas)`
 - F.** `plus(xmas, blackFriday)`

18. Which statements about the following class are true?

```
package secure;
import java.io.*;
public class Login {
    public void clearPassword(char[] password) {
        for(int i=0; i<password.length; i++) {
            password[i] = 0;
        }
    }
    public String getPassword() {
        Console c = System.console();
        final char[] pass = c.readPassword("Enter your password: ");
        StringBuilder sb = new StringBuilder();
        for(char p : pass) {
            sb.append(p);
        }
        clearPassword(pass);
        return sb.toString();
    }
    public static void main(String[] webLogin) {
        String pass = new Login().getPassword();
    }
}
```

- I. The class compiles.
 - II. The design protects the password by clearing it from memory after it is entered.
 - III. The class may throw an exception at runtime.
- A. I only
 - B. II only
 - C. I and II only
 - D. I and III only
 - E. I, II, and III
19. Which of the following can fill in the blank to print out the numbers 161, 183, and 201 in any order?

```
class Runner {
    private int numberMinutes;
    public Runner(int n) {
        numberMinutes = n;
    }
}
```

```
public int getNumberMinutes() {  
    return numberMinutes;  
}  
}  
}  
public class Marathon {  
    public static void main(String[] args) {  
        Stream<Runner> runners = Stream.of(new Runner(183),  
            new Runner(161), new Runner(201));  
        OptionalInt opt = runners._____;  
    }  
}
```

- A. `map(Runner::getNumberMinutes)
 .peek(System.out::println).max()`
 - B. `mapToInt(Runner::getNumberMinutes)
 .peek(System.out::println).max()`
 - C. `peek(System.out::println)
 .mapToInt(Runner::getNumberMinutes).max()`
 - D. `peek(System.out::println)
 .mapToInt(Runner::getNumberMinutes).max()`
 - E. None of the above
20. What is the output of the following application?

```
1: package fruit;  
2: enum Season {  
3:     SPRING(1), SUMMER(2), FALL(3), WINTER(4)  
4:     public Season(int orderId) {}  
5: }  
6: public class PickApples {  
7:     public static void main(String... orchard) {  
8:         final Season s = Season.FALL;  
9:         switch(s) {  
10:             case 3:  
11:                 System.out.println("Time to pick!");  
12:             default:  
13:                 System.out.println("Not yet!");  
14:         }  
15:     }  
16: }
```

- A. Time to pick!
 - B. Time to pick! followed by Not yet!
 - C. One line of code does not compile.
 - D. Two lines of code do not compile.
 - E. Three lines of code do not compile.
 - F. The code compiles but prints an exception at runtime.
21. Which of the following expressions, inserted simultaneously into both blanks, allow the application to compile? (Choose three.)

```
package spooky;
import java.util.function.*;
abstract class Phantom {
    public void bustLater(DoubleConsumer buster, double value) {
        buster.accept(value);
    }
}
public class Ghost extends Phantom {
    public void bustNow(Consumer<Double> buster, double value) {
        buster.accept(value);
    }
    void call() {
        double value = 10;
        bustNow(_____,value);
        bustLater(_____,value);
    }
}
```

- A. System.out::print
 - B. a -> {System.out.println(a.intValue());}
 - C. g -> {System.out.println();}
 - D. u -> System.out.println((long)u)
 - E. v -> System.out.print(v)
 - F. w -> System.out::println
22. What is the output of the following?

```
package counter;
import java.util.*;

public class CountResource extends ListResourceBundle {
```

```
private int count = 0;

@Override
protected Object[][] getContents() {
    return new Object[][] { { "count", ++count } };
}
public static void main(String[] args) {
    ResourceBundle rb = ResourceBundle.getBundle("counter.CountResource");
    System.out.println(rb.getString("count") + " " + rb.getString("count"));
}
}

A. 0 0
B. 0 1
C. 1 1
D. 1 2
E. The code does not compile.
F. The code compiles but throws an exception at runtime.
```

23. In most of the United States, daylight savings time ends on November 5, 2017 at 02:00 a.m., and we repeat that hour. What is the output of the following?

```
import java.time.*;
public class FallBack {
    public static void main(String[] args) {
        LocalDate localDate = LocalDate.of(2017, 10, 5);
        LocalTime localTime = LocalTime.of(1, 0);
        ZoneId zone = ZoneId.of("America/New_York");
        ZonedDateTime z = ZonedDateTime.of(localDate, localTime, zone);

        for (int i = 0; i < 6; i++)
            z = z.plusHours(1);

        System.out.println(z.getHour());
    }
}
```

- A. 5
- B. 6
- C. 7
- D. The code does not compile.
- E. The code compiles, but throws an exception at runtime.

24. Assume the `/environment` directory exists and contains a file with a symbolic link to the `/environment` directory. In addition, assume all files within the directory are fully accessible. What is the result of executing the following program?

```
import java.nio.file.*;
import java.nio.file.attribute.*;
public class SearchEnvironment {
    public static void accessFile(Path p, long timeEpoch) {
        try {
            Files.readAttributes(p, BasicFileAttributes.class)
                .setTimes(null, null, FileTime.fromMillis(timeEpoch));
        } catch (Throwable e) {
        } finally {}
    }
    public static final void main(String[] unused) throws Exception {
        Path w = Paths.get("/environment");
        Files.walk(w)
            .forEach(q -> accessFile(q, System.currentTimeMillis()));
    }
}
```

- A. The code does not compile.
 - B. The program exits after successfully updating the creation time of files within the directory.
 - C. The program exits after successfully updating the last accessed time of files within the directory.
 - D. The program compiles but throws an exception at runtime.
 - E. The program enters an infinite loop and hangs at runtime.
25. Which command causes the following class to throw an `AssertionError` at runtime?

```
public class Watch {
    private static final short DEFAULT_HOUR = 12;
    private Watch() {
        super();
    }
    int checkHour() {
        assert DEFAULT_HOUR > 12;
        return DEFAULT_HOUR;
    }
    public static void main(String... ticks) {
        new Watch().checkHour();
    }
}
```

- A. `java -da Watch`
 - B. `java -ea:Watch -da Watch`
 - C. `java -ea -da:Watch Watch`
 - D. `java -enableassert Watch`
 - E. None of the above since the `Watch` class does not compile.
- 26.** What is the output of the following application?

```
package rope;
import java.util.concurrent.*;
import java.util.stream.IntStream;
public class Jump {
    private static void await(CyclicBarrier b) {
        try {
            b.await();
        } catch (InterruptedException | BrokenBarrierException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] chalk) {
        ExecutorService s = Executors.newFixedThreadPool(4);
        final CyclicBarrier b = new CyclicBarrier(4,
            () -> System.out.print("Jump!"));
        IntStream
            .iterate(1, q -> 2)
            .limit(10)
            .forEach(q -> s.execute(() ->await(b)));
        s.shutdown();
    }
}
```

- A. `Jump!` is printed and exits.
- B. `Jump!` is printed twice and exits.
- C. The code does not compile.
- D. The output cannot be determined ahead of time.
- E. The program hangs indefinitely at runtime because the `IntStream` is not parallel.
- F. None of the above

27. What is the output of the following code snippet? Assume the two directories referenced both exist and are symbolic links to the same location within the file system.

```
if(Files.isSameFile("/salad/carrot", "/fruit/apple"))
    System.out.println("Same!");
else System.out.println("Different!");
```

- A. Same!
- B. Different!
- C. The code does not compile.
- D. The code compiles but throws an exception at runtime.
- E. None of the above

28. Which can fill in the blank JavaProgrammerCert class to compile and logically complete the code? (Choose two.)

```
class JavaProgrammerCert extends Exam {
    private Exam oca;
    private Exam ocp;
    // assume getters and setters are here
}
public class Exam {
    boolean pass;
    protected boolean passed() {
        return pass;
    }
}
```

- A. boolean passed() { return oca.pass && ocp.pass; }
- B. boolean passed() { return oca.passed() && ocp.passed(); }
- C. boolean passed() { return super.passed(); }
- D. public boolean passed() { return oca.passed() && ocp.passed(); }
- E. public boolean passed() { return oca.pass && ocp.pass; }
- F. public boolean passed() { return super.passed(); }

29. Which statements about overriding a method are correct? (Choose two.)

- A. An overridden method must not throw a narrower version of any checked exception thrown by the inherited method.
- B. An overridden method must use a return type that is covariant with the inherited method.
- C. An overridden method must use a different set of input arguments as the inherited method.

- D. Overridden methods must use an access modifier at least as broad as their inherited method.
 - E. It is possible to override `private` and `static` methods.
 - F. The `@Override` annotation is required anytime a method is overridden.
30. Which of the following can be independently inserted into the blank so the code can run without error for at least one SQL query?

```
private static void choices(Connection conn, String sql)
    throws SQLException {
    try (Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        _____
    }
}
```

- I. `System.out.println(rs.getInt(1));`
 - II. `rs.next(); System.out.println(rs.getInt(1));`
 - III. `if (rs.next()) System.out.println(rs.getInt(1));`
- A. II
 - B. III
 - C. I and III
 - D. II and III
 - E. I, II, and III
 - F. None of the above
31. Starting with `DoubleBinaryOperator` and going downward, fill in the values for the table.

Functional Interface	# Parameters
<code>DoubleBinaryOperator</code>	
<code>LongToIntFunction</code>	
<code>ToLongBiFunction</code>	
<code>IntSupplier</code>	
<code>ObjLongConsumer</code>	

- A. 1, 0, 0, 0, 2
- B. 1, 2, 1, 0, 1
- C. 2, 1, 0, 1, 2
- D. 2, 1, 1, 0, 1
- E. 2, 1, 2, 0, 2
- F. 3, 0, 2, 1, 1

32. What is a possible output of the following?

```
LocalDate trainDay = LocalDate.of(2017, 5, 13);
LocalTime time = LocalTime.of(10, 0);
ZoneId zone = ZoneId.of("America/Los_Angeles");
ZonedDateTime zdt = ZonedDateTime.of(trainDay, time, zone);
Instant instant = zdt.toInstant();
instant = instant.plus(1, ChronoUnit.YEARS);
System.out.println(instant);
```

- A. 2017-05-13T10:00-07:00[America/Los_Angeles]
- B. 2017-05-13T17:00:00Z
- C. 2018-05-13T10:00-07:00[America/Los_Angeles]
- D. 2018-05-13T17:00:00Z
- E. The code does not compile.
- F. The code compiles but throws an exception at runtime.

33. How many lines does the following output?

```
import java.util.stream.*;

class Blankie {
    String color;
    boolean isPink() {
        return "pink".equals(color);
    }
}
public class PreSchool {
    public static void main(String[] args) {
        Blankie b1 = new Blankie();
        Blankie b2 = new Blankie();
        b1.color = "pink";
        Stream.of(b1, b2).filter(Blankie::isPink).forEach(System.out::println);
    }
}
```

- A.** None
 - B.** One
 - C.** Two
 - D.** The code does not compile.
 - E.** The code compiles but throws an exception at runtime.
- 34.** Which are the minimum changes needed to make this class immutable?
- ```
1: public class Tree {
2: String species;
3: public Tree(String species) {
4: this.species = species;
5: }
6: public String getSpecies() {
7: return species;
8: }
9: private final void setSpecies(String newSpecies) {
10: species = newSpecies;
11: }
12: }
```
- I.** Make species private and final.
  - II.** Make the getter method final.
  - III.** Remove the setter method.
- A.** None. It is already immutable.
  - B.** I
  - C.** I and II
  - D.** I and III
  - E.** II and III
  - F.** I, II, and III
- 35.** Let's say you needed to write a large List of Student objects to a data file. Which three concrete classes, chained together, would best accomplish this? (Choose three.)
- A.** BufferedOutputStream
  - B.** BufferedOutputStream
  - C.** FileOutputStream
  - D.** FileWriter
  - E.** ObjectOutputStream
  - F.** OutputStream

- 36.** Which statements when inserted independently will throw an exception at runtime?  
(Choose two.)

```
ArrayDeque<Integer> d = new ArrayDeque<>();
d.offer(18);
// INSERT CODE HERE
```

- A.** d.peek(); d.peek();
- B.** d.poll(); d.poll();
- C.** d.pop(); d.pop();
- D.** d.remove(); d.remove();

- 37.** What is the output of the following code snippet?

```
11: Path x = Paths.get(".", "song", "..", "/note");
12: Path y = Paths.get("/dance/move.txt");
13: x.normalize();
14: System.out.println(x.resolve(y));
15: System.out.println(y.relativize(x));
```

- A.** ./song/..../note/dance/move.txt  
/dance/move.txt
- B.** /dance/move.txt  
/dance/move.txt./song/..../note
- C.** /dance/move.txt  
/dance/move.txt/note
- D.** /note/dance/move.txt  
..../dance/move.txt/song
- E.** The code does not compile.
- F.** The code compiles but an exception is thrown at runtime.

- 38.** Given the following class, how many lines contain compilation errors?

```
package field;
import java.io.*;
class StungException extends Exception {}
class Suit implements Closeable {
 public void close() throws IOException {}
}
public class BeeCatcher {
 public static void main(String... bees) {
 try (Suit s = new Suit(), Suit t = new Suit()) {
 throw new StungException();
```

```
 } catch (Exception e) {
 } catch (StungException e) {
 } finally {
 }
 }
}
```

- A.** One
- B.** Two
- C.** Three
- D.** Four
- E.** None. The code compiles as is.

**39.** What is the output of the following application?

```
package homework;
import java.util.*;
import java.util.stream.*;
public class QuickSolution {
 public static int findFast(Stream<Integer> s) {
 return s.findAny().get();
 }
 public static int findSlow(Stream<Integer> s) {
 return s.parallel().findFirst().get();
 }
 public static void main(String[] pencil) {
 Stream<Integer> s1 = Arrays.asList(1,2,3,4,5).stream();
 Stream<Integer> s2 = Arrays.asList(1,2,3,4,5).stream();
 int val1 = findFast(s1);
 int val2 = findSlow(s2);
 System.out.print(val1+" "+val2);
 }
}
```

- A.** 1 1
- B.** 3 1
- C.** The answer cannot be determined until runtime.
- D.** The code does not compile.
- E.** The code compiles but throws an exception at runtime.

40. Given this property file used to load the `Properties` object `props` and this code snippet, what is the output?

```
mystery=bag
type=paper
18: System.out.print(props.getDefaultProperty("mystery", "?"));
19: System.out.print(" ");
20: System.out.print(props.getDefaultProperty("more", "?"));
```

- A. bag
- B. bag null
- C. bag ?
- D. The code does not compile.
- E. The code compiles but throws an exception at runtime.

41. Which statements about `try-with-resources` are true? (Choose two.)

- A. Any resource used must implement `Closeable`.
- B. If more than one resource is used, then the order in which they are closed is the reverse of the order in which they were created.
- C. If the `try` block and `close()` method both throw an exception, the one thrown by the `try` block is suppressed.
- D. Neither a `catch` nor a `finally` block is required.
- E. The `close()` method of the resources must throw a checked exception.

42. How many lines fail to compile?

```
class Roller<E extends Wheel> {
 public void roll(E e) { }
}
class Wheel { }
class CartWheel extends Wheel { }

public class RollingContest {
 Roller<CartWheel> wheel1 = new Roller<CartWheel>();
 Roller<Wheel> wheel2 = new Roller<CartWheel>();
 Roller<? extends Wheel> wheel3 = new Roller<CartWheel>();
 Roller<? extends Wheel> wheel4 = new Roller<Wheel>();
 Roller<? super Wheel> wheel5 = new Roller<CartWheel>();
 Roller<? super Wheel> wheel6 = new Roller<Wheel>();
}
```

- A. One
- B. Two
- C. Three

- D. Four
- E. Five
- F. Six

43. Which are the minimum changes needed to properly implement the singleton pattern?

```
1: public class Bookmark {
2: private static Bookmark bookmark;
3: private int pageNumber;
4: static {
5: bookmark = new Bookmark();
6: }
7: public static Bookmark getInstance() {
8: return bookmark;
9: }
10: public int getPageNumber() {
11: return pageNumber;
12: }
13: public void setPageNumber(int newNumber) {
14: pageNumber = newNumber;
15: }
16: }
```

- I. Add a private constructor.
  - II. Remove the setter method.
  - III. Remove the static block and change line 2 to instantiate Bookmark.
- A. None. It is already a singleton.
  - B. I
  - C. I and II
  - D. I and III
  - E. II and III
  - F. I, II, and III

44. Given an updatable ResultSet that contains the following and this code, what does the code snippet output?

| color<br>character varying(255) | count<br>integer |
|---------------------------------|------------------|
| black                           | 20               |
| blue                            | 5                |
| red                             | 0                |

```
rs.afterLast();
rs.previous();
rs.updateInt(2, 10);

rs = stmt.executeQuery("select * from pens where color = 'red'");
while (rs.next()) {
 System.out.println(rs.getInt(2));
}
```

- A.** 0
  - B.** 10
  - C.** The code does not compile.
  - D.** The code compiles but throws an exception at runtime.
- 45.** Which statements describe a `java.io` stream class and cannot be applied to a `java.util.stream.Stream` class? (Choose three.)
- A.** Can be used with `try-with-resources` statement
  - B.** Includes a class or set of classes used solely for working with character data
  - C.** Requires all data objects to implement `Serializable`
  - D.** Some classes contain a `flush()` method.
  - E.** Some classes contain a method to skip over data.
  - F.** Some classes contain a method to sort the data.
- 46.** Bill wants to create a program that reads all of the lines of all of his books using NIO.2. Unfortunately, Bill may have made a few mistakes writing his program. How many lines of the following class contain compilation errors?

```
1: package bookworm;
2: import java.io.*;
3: import java.nio.file.*;
4: public class ReadEverything {
5: public void readFile(Path p) {
6: try {
7: Files.readAllLines(p)
8: .parallel()
9: .forEach(System.out::println);
10: } catch (Exception e) {}
11: }
12: public void read(Path directory) throws Exception {
13: Files.walk(directory)
14: .filter(p -> File.isRegularFile(p))
15: .forEach(x -> readFile(x));
```

```
16: }
17: public static void main(String... books) throws IOException {
18: Path p = Path.get("collection");
19: new ReadEverything().read(p);
20: }
21: }
```

- A. None. Bill's implementation is correct.
- B. One
- C. Two
- D. Three
- E. Four
- F. Five
47. Assuming the following class is concurrently accessed by numerous threads, which statement about the CountSheep class is correct?

```
package fence;
import java.util.concurrent.atomic.*;
public class CountSheep {
 private static AtomicInteger counter = new AtomicInteger();
 private Object lock = new Object();
 public synchronized int increment1() {
 return counter.incrementAndGet();
 }
 public static synchronized int increment2() {
 return counter.getAndIncrement();
 }
 public int increment3() {
 synchronized(lock) {
 return counter.getAndIncrement();
 }
 }
}
```

- A. The class is thread-safe only if `increment1()` is removed.
- B. The class is thread-safe only if `increment2()` is removed.
- C. The class is thread-safe only if `increment3()` is removed.
- D. The class is already thread-safe.
- E. The class does not compile.
- F. The class compiles but may throw an exception at runtime.

**48.** Which of the following are not required parameters for the NIO.2 Files.find() method? (Choose two.)

- A.** BiPredicate
- B.** FileVisitOption...
- C.** int
- D.** long
- E.** Path

**49.** Which statements are correct? (Choose two.)

- A.** A Comparable implementation is often implemented by a lambda.
- B.** A Comparable object has a compare() method.
- C.** The compare() and compareTo() methods have the same contract for the return value.
- D.** There can be multiple Comparator implementations for the same class.
- E.** Two objects that return true for equals() will always return 0 when passed to compareTo().

**50.** What is the output of the following code snippet, assuming none of the files referenced exist within the file system?

```
Path t1 = Paths.get("/sky/../../stars.exe");
Path t2 = Paths.get("/stars.exe");
Path t3 = t1.resolve(t2);

boolean b1 = t1.equals(t2);
boolean b2 = t1.normalize().equals(t2);
boolean b3 = Files.isSameFile(t1.normalize(),t2);
boolean b4 = Files.isSameFile(t2,t3);

System.out.print(b1+","+b2+","+b3+","+b4);
```

- A.** false,false,true,true
- B.** false,true,true,false
- C.** false,true,true,true
- D.** true,false,true,false
- E.** The code does not compile.
- F.** The code compiles but throws an exception at runtime.

**51.** Let's say we have a Reader instance that will produce the characters with the numeric values {1,2,3,4,5,6,7}. Which of the following are possible outcomes of executing the checkLottoNumbers() method with this Reader instance? (Choose two.)

```
23: public String checkLottoNumbers(Reader r) throws IOException {
24: r.read();r.skip(1);
25: r.mark(5);
```

```
26: r.skip(1);
27: r.reset();
28: return r.read()+"-"+r.read(new char[5]);
29: }
```

- A. An IOException on line 25
  - B. An IOException on line 27
  - C. 'c'-4 is returned.
  - D. 'd'-3 is returned.
  - E. 3-4 is returned.
  - F. 4-3 is returned.
52. Fill in the blanks: The name of the abstract method in the Function interface is \_\_\_\_\_, while the name of the abstract method in the Consumer interface is \_\_\_\_\_.
- A. accept(), apply()
  - B. accept(), get()
  - C. apply(), accept()
  - D. apply(), apply()
  - E. apply(), test()
53. Assuming the following program is executed with assertions enabled, which is the first line to throw an exception at runtime?

```
1: package school;
2: public class Teacher {
3: public int checkClasswork(int choices) {
4: assert choices++==10 : 1;
5: assert true!=false : new StringBuilder("Answer2");
6: assert(null==null) : new Object();
7: assert ++choices==11 : "Answer4";
8: assert 2==3 : "";
9: return choices;
10: }
11: public final static void main(String... students) {
12: try {
13: new Teacher().checkClasswork(10);
14: } catch (Error e) {
15: System.out.print("Bad idea");
16: throw e;
17: }
18: }
19: }
```

- A. Line 4
  - B. Line 5
  - C. Line 6
  - D. Line 7
  - E. Line 8
  - F. None of the above since the class does not compile
54. Which of the following are valid functional interfaces in the `java.util.function` package? (Choose three.)
- A. `BooleanSupplier`
  - B. `CharSupplier`
  - C. `DoubleUnaryOperator`
  - D. `ObjectIntConsumer`
  - E. `ToLongBiFunction`
  - F. `TriPredicate`
55. Which statements about the following class are correct? (Choose two.)

```
package knowledge;
class InformationException extends Exception {}
public class LackOfInformationException extends InformationException {
 public LackOfInformationException() { // t1
 super("");
 }
 public LackOfInformationException(String s) { // t2
 this(new Exception(s));
 }
 public LackOfInformationException(Exception c) { // t3
 super();
 }
 @Override public String getMessage() {
 return "lackOf";
 }
}
```

- A. `LackOfInformationException` compiles without issue.
- B. The constructor declared at line t1 does not compile.
- C. The constructor declared at line t2 does not compile.
- D. The constructor declared at line t3 does not compile.
- E. The `getMessage()` method does not compile because of the `@Override` annotation.
- F. `LackOfInformationException` is a checked exception.

56. How many changes do you need to make in order for this code to compile?

```
public class Ready {
 private static double getNumber() {
 return .007;
 }
 public static void math() {
 Supplier<double> s = Ready::getNumber;
 double d = s.get();
 System.out.println(d);
 }
}
```

- A. None
- B. One
- C. Two
- D. Three
- E. Four

57. Which statement about the following class is correct?

```
package robot;
import java.util.concurrent.*;
public class PassButter extends RecursiveTask<String> { // j1
 final int remainder;
 public PassButter(int remainder) { // j2
 this.remainder = remainder;
 }
 @Override
 protected String compute() {
 if (remainder <= 1)
 return "1";
 else {
 PassButter otherTask = new PassButter(remainder - 1);
 String otherValue = otherTask.fork().join(); // j3
 return otherValue
 + new PassButter(remainder - 2).compute();
 }
 }
 public static void main(String[] purpose) {
 ForkJoinPool pool = new ForkJoinPool();
 ForkJoinTask<?> task = new PassButter(10);
```

```
 System.out.print(pool.invoke(task));
 pool.shutdown();
 }
}
```

- A.** The code does not compile due to line j1.
  - B.** The code does not compile due to line j2.
  - C.** The code does not compile due to line j3.
  - D.** The code compiles and properly implements the fork/join framework in a multi-threaded manner.
  - E.** The code compiles but does not implement the fork/join framework in a proper multi-threaded manner.
  - F.** The class compiles and prints an exception at runtime.
- 58.** Which can fill in the blank so this code outputs true?
- ```
import java.util.function.*;
import java.util.stream.*;

public class HideAndSeek {
    public static void main(String[] args) {
        Stream<Boolean> hide = Stream.of(true, false, true);
        Predicate<Boolean> pred = b -> b;
        boolean found = hide.filter(pred)._____ (pred);
        System.out.println(found);
    }
}
```
- A.** Only anyMatch()
 - B.** Only allMatch()
 - C.** Both anyMatch() and allMatch()
 - D.** Only noneMatch()
 - E.** The code does not compile with any of these options.
- 59.** Given the following code, Java will try to find a matching resource bundle. Which order will Java search to find a match?

```
Locale.setDefault(new Locale("en"));
ResourceBundle.getBundle("AB", new Locale("fr"));
```

- A.** AB.class, AB.properties, AB_en.properties, AB_fr.properties
- B.** AB.properties, AB.class, AB_en.properties, AB_fr.properties
- C.** AB_en.properties, AB_fr.properties, AB.class, AB.properties
- D.** AB_fr.properties, AB.class, AB.properties, AB_en.properties

- E. AB_fr.properties, AB_en.properties, AB.class, AB.properties
- F. AB_fr.properties, AB_en.properties, AB.properties, AB.class

60. What is the result of the following?

```
Set<Integer> dice = new TreeSet<>();  
dice.add(6);  
dice.add(6);  
dice.add(4);  
dice.stream().filter(n -> n != 4).forEach(System.out::println).count();
```

- A. It prints just one line.
- B. It prints one line and then the number 3.
- C. There is no output.
- D. The code does not compile.
- E. The code compiles but throws an exception at runtime.

61. Given the following two property files in the pod package, what does the following class output?

```
pod.container.properties
```

```
name=generic
```

```
number=2
```

```
pod.container_en.properties
```

```
name=Docker
```

```
type=container
```

```
package pod;  
import java.util.*;  
public class WhatKind {  
    public static void main(String[] args) {  
        Locale.setDefault(new Locale("ja"));  
        ResourceBundle rb = ResourceBundle.getBundle("pod.container");  
        String name = rb.getString("name");    // r1  
        String type = rb.getString("type");    // r2  
        System.out.println(name + " " + type);    }  
}
```

- A. Docker container
- B. generic container
- C. generic null
- D. The code does not compile.
- E. Line r1 throws an exception.
- F. Line r2 throws an exception.

62. What is the result of the following?

```
import java.util.stream.*;
public class StreamOfStreams {
    public static void main(String[] args) {
        Integer result =
            Stream.of(getNums(9, 8), getNums(22, 33)) // c1
                .filter(x -> !x.isEmpty()) // c2
                .flatMap(x -> x) // c3
                .max((a, b) -> a - b) // c4
                .get();
        System.out.println(result);
    }
    private static Stream<Integer> getNums(int num1, int num2) {
        return Stream.of(num1, num2);
    }
}
```

- A.** The code compiles and outputs 8.
- B.** The code compiles and outputs 33.
- C.** The code does not compile due to line c1.
- D.** The code does not compile due to line c2.
- E.** The code does not compile due to line c3.
- F.** The code does not compile due to line c4.

63. Which of the following shows a valid Locale format? (Choose two.)

- A.** de
- B.** DE
- C.** de_DE
- D.** DE_de

64. What is true of the following if the music database exists and contains a songs table with one row when run using a JDBC 4.0 driver? (Choose two.)

```
import java.sql.*;
public class Music {
    public static void main(String[] args) throws Exception {
        String url = "jdbc:derby:music";
        Connection conn = DriverManager.getConnection(url);
        Statement stmt = conn.createStatement();
        stmt.execute("update songs set name = 'The New Song'");
    }
}
```

- A. The code does not compile.
 - B. The code does not update the database because it calls `execute()` rather than `executeUpdate()`.
 - C. The code does not update the database because the `Statement` is never closed.
 - D. The code runs without error.
 - E. The `execute()` method returns a `boolean`.
 - F. The `execute()` method returns an `int`.
- 65.** How many of the following pairs of values can fill in the blanks to comply with the contract of the `hashCode()` and `equals()` methods?
- ```
class Sticker {
 @Override
 public int hashCode() {
 return _____;
 }
 @Override
 public boolean equals(Sticker o) {
 return _____;
 }
}
```
- I. 5, `false`
  - II. 5, `true`
  - III. `new Random().nextInt()`, `false`
  - IV. `new Random().nextInt()`, `true`
- A. None
  - B. One
  - C. Two
  - D. Three
  - E. Four
  - F. None of the above. The code does not compile with any of the options.
- 66.** What is the output of the following application?

```
package winter;

abstract class TShirt {
 abstract int insulate();
 public TShirt() {
 System.out.print("Starting...");
 }
}
```

```
public class Wardrobe {
 abstract class Sweater extends TShirt {
 int insulate() {return 5;}
 }
 private static void dress() {
 class Jacket extends Sweater { // v1
 int insulate() {return 10;}
 };
 final TShirt outfit = new Jacket() { // v2
 int insulate() {return 20;}
 };
 System.out.println("Insulation:"+outfit.insulate());
 }
 public static void main(String... snow) {
 new Wardrobe().dress();
 }
}
```

- A. Starting...Insulation:20
- B. Starting...Insulation:40
- C. The code does not compile because of line v1.
- D. The code does not compile because of line v2.
- E. The code does not compile for a different reason.

67. Which statements about the following application are true?

```
1: package armory;
2: import java.util.function.*;
3: class Shield {}
4: public class Sword {
5: public class Armor {
6: int count;
7: public final Function<Shield,Sword,Armor> dress = (h,w) -> new Armor();
8: public final IntSupplier<Integer> addDragon = () -> count++;
9: }
10: public static void main(String[] knight) {
11: final Armor a = new Armor();
12: a.dress.apply(new Shield(), new Sword());
13: a.addDragon.getAsInt();
14: }
15: }
```

- I. The lambda expression for `dress` on line 7 compiles without issue.
  - II. The lambda expression for `addDragon` on line 8 compiles without issue.
  - III. Not counting the lambda expressions on lines 7 and 8, the code does not contain any compilation errors.
- A. I only
  - B. I and II only
  - C. I, II, and III
  - D. II and III only
  - E. None of the above
- 68.** Which two conditions best describe a thread that appears to be active but is perpetually stuck and never able to finish its task? (Choose two.)
- A. Deadlock
  - B. Livelock
  - C. Loss of precision
  - D. Out of memory error
  - E. Race condition
  - F. Starvation
- 69.** Which statements are true about the following date/times? (Choose two.)
- 2017-04-01T17:00+03:00[Africa/Nairobi]  
2017-04-01T10:00-05:00[America/Panama]
- A. The first date/time is earlier.
  - B. The second date/time is earlier.
  - C. Both represent the same date/time.
  - D. The two date/times are zero hours apart.
  - E. The two date/times are one hour apart.
  - F. The two date/times are two hours apart.
- 70.** What is true about the following?

```
import java.util.*;
public class Yellow {
 public static void main(String[] args) {
 List list = Arrays.asList("Sunny");
 method(list); // c1
 }
 private static void method(Collection<?> x) { // c2
 x.forEach(a -> {}); // c3
 }
}
```

- A. The code doesn't compile due to line c1.
  - B. The code doesn't compile due to line c2.
  - C. The code doesn't compile due to line c3.
  - D. The code compiles and runs without output.
  - E. The code compiles but throws an exception at runtime.
71. What is true about the following code? (Choose two.)

```
public static void main(String[] args) throws Exception {
 String url = "jdbc:derby:hats;create=true";
 Connection conn = null;
 Statement stmt = null;

 try {
 conn = DriverManager.getConnection(url);
 stmt = conn.createStatement();
 stmt.executeUpdate(
 "CREATE TABLE caps (name varchar(255), size varchar(1))");
 } finally {
 conn.close();
 stmt.close();
 }
}
```

- A. If using a JDBC 3.0 driver, this code throws an exception.
  - B. If using a JDBC 4.0 driver, this code throws an exception.
  - C. The resources are closed in the wrong order.
  - D. The resources are closed in the right order.
  - E. The Connection is created incorrectly.
  - F. The Statement is created incorrectly.
72. How many lines of the following application contain a compilation error?

```
package puzzle;
final interface Finder {
 default long find() {return 20;}
}
abstract class Wanda {
 abstract long find();
}
final class Waldo extends Wanda implements Finder {
 long find() {return 40;}
```

```
public static final void main(String[] pictures) {
 final Finder f = new Waldo();
 System.out.print(f.find());
}
}
```

- A.** One
- B.** Two
- C.** Three
- D.** None. The code compiles and prints 20 at runtime.
- E.** None. The code compiles and prints 40 at runtime.

**73.** What is the output of the following?

```
1: package reader;
2: import java.util.stream.*;
3:
4: public class Books {
5: public static void main(String[] args) {
6: IntStream pages = IntStream.of(200, 300);
7: long total = pages.sum();
8: long count = pages.count();
9: System.out.println(total + "-" + count);
10: }
11: }
```

- A.** 2-2
- B.** 200-1
- C.** 500-0
- D.** 500-2
- E.** The code does not compile.
- F.** The code compiles but throws an exception at runtime.

**74.** What is the output of executing the following code snippet?

```
30: ExecutorService e = Executors.newSingleThreadExecutor();
31: Runnable r1 = () -> Stream.of(1,2,3).parallel();
32: Callable r2 = () -> Stream.of(4,5,6).parallel();
33:
34: Future<Stream> f1 = e.submit(r1);
35: Future<Stream> f2 = e.submit(r2);
36:
```

```

37: Stream<Integer> s = Stream.of(f1.get(),f2.get())
38: .flatMap(p -> p)
39: .parallelStream();
40:
41: ConcurrentMap<Boolean,List<Integer>> r =
42: s.collect(Collectors.groupingByConcurrent(i -> i%2==0));
43: System.out.println(r.get(false).size()+" "+r.get(true).size());

```

- A. 3 3  
 B. 2 4  
 C. The code does not compile due to one error.  
 D. The code does not compile due to two errors.  
 E. The code does not compile due to three errors.  
 F. The code compiles but a `NullPointerException` is thrown at runtime.
75. Fill in the blanks: If your application is \_\_\_\_\_, it must first have been \_\_\_\_\_ with respect to supporting multiple languages.
- A. extracted, internationalized  
 B. extracted, localized  
 C. internationalized, extracted  
 D. internationalized, localized  
 E. localized, extracted  
 F. localized, internationalized
76. Which statement about the following class is true? Assume the file system is available and able to be modified.

```

package forest;
import java.io.File;
public class CreateTree {
 public boolean createTree(String tree) {
 if(new File(tree).exists()) {
 return true;
 } else {
 return new File(tree).mkdir();
 }
 }
 public static void main(String[] seeds) {
 final CreateTree creator = new CreateTree();
 System.out.print(creator.createTree("/woods/forest"));
 }
}

```

- A. The class compiles and always prints true at runtime.
  - B. The class compiles and always prints false at runtime.
  - C. The class compiles but the output cannot be determined until runtime.
  - D. The class compiles but may throw an exception at runtime.
  - E. The class does not compile.
77. What does the following print?
- ```
1:  class SmartWatch extends Watch {  
2:      private String getType() { return "smart watch"; }  
3:      public String getName() {  
4:          return getType() + ",";  
5:      }  
6:  }  
7:  public class Watch {  
8:      private String getType() { return "watch"; }  
9:      public String getName(String suffix) {  
10:          return getType() + suffix;  
11:      }  
12:      public static void main(String[] args) {  
13:          Watch watch = new Watch();  
14:          Watch smartWatch = new SmartWatch();  
15:          System.out.print(watch.getName(", "));  
16:          System.out.print(smartWatch.getName(""));  
17:      }  
18:  }
```
- A. smart watch,smart watch
 - B. smart watch,watch
 - C. watch,smart watch
 - D. watch,watch
 - E. None of the above
78. In most of the United States, daylight savings time ends on November 5, 2017 at 02:00 a.m., and we repeat that hour. What is the output of the following?

```
import java.time.*;  
public class FallBack {  
    public static void main(String[] args) {  
        LocalDate localDate = LocalDate.of(2017, Month.NOVEMBER, 5);  
        LocalTime localTime = LocalTime.of(1, 0);  
        ZoneId zone = ZoneId.of("America/New_York");  
        ZonedDateTime z = ZonedDateTime.of(localDate, localTime, zone);
```

```
        for (int i = 0; i < 6; i++)
            z = z.plusHours(1);

        System.out.println(z.getHour());
    }
}
```

- A.** 5
- B.** 6
- C.** 7
- D.** The code does not compile.
- E.** The code compiles but throws an exception at runtime.

79. Which statements about the following application are true?

```
package party;
import java.util.concurrent.*;
public class Plan {
    private ExecutorService service = Executors.newCachedThreadPool();
    public void planEvents() {
        service.scheduleWithFixedDelay(
            () -> System.out.print("Check food stock"),
            1, TimeUnit.HOURS);
        service.scheduleAtFixedRate(
            () -> System.out.print("Check drink stock"),
            1, 1000, TimeUnit.SECONDS);
        service.execute(() -> System.out.print("Take out trash"));
    }
}
```

- I.** The `scheduleWithFixedDelay()` method call compiles.
 - II.** The `scheduleAtFixedRate()` method call compiles.
 - III.** The `execute()` method call compiles.
- A.** I only
 - B.** II only
 - C.** III only
 - D.** I and II
 - E.** I, II, and III
 - F.** None of the above

- 80.** Which of the following classes are checked exception? (Choose three.)
- A.** `java.io.NotSerializableException`
 - B.** `java.lang.AssertionError`
 - C.** `java.lang.IllegalArgumentException`
 - D.** `java.sql.SQLException`
 - E.** `java.text.ParseException`
 - F.** `java.util.MissingResourceException`
- 81.** Which of the following are valid functional interfaces? (Choose two.)
- A.** `interface CanClimb {default void climb() {}
 static void climb(int x) {}}`
 - B.** `interface CanDance {int dance() { return 5;}}`
 - C.** `interface CanFly {abstract void fly();}`
 - D.** `interface CanRun {void run();
 static double runFaster() {return 2.0;}}`
 - E.** `interface CanSwim {abstract Long swim();
 boolean test();}`
- 82.** How many of the following could be valid JDBC URL formats for an imaginary driver named `magic` and a database named `box`?
- I.** `jdbc;box;magic`
 - II.** `jdbc;magic;@127.0.0.1:1234`
 - III.** `jdbc;magic;//@127.0.0.1:1234`
 - IV.** `jdbc;magic;127.0.0.1:1234/box`
 - V.** `magic;jdbc;127.0.0.1:1234/box`
- A.** None
 - B.** One
 - C.** Two
 - D.** Three
 - E.** Four
 - F.** Five
- 83.** What is the output of the following?

```
Stream<String> s = Stream.of("speak", "bark", "meow", "growl");  
Map<Integer, String> map = s.collect(toMap(String::length, k -> k));  
System.out.println(map.size() + " " + map.get(4));
```

- A.** 2 bark
B. 2 meow
C. 4 bark
D. 4 meow
E. The output is not guaranteed.
F. The code compiles but throws an exception at runtime.
- 84.** What is the output of the following application?
- ```
package music;
interface DoubleBass {
 void strum();
 default int getVolume() {return 5;}
}
interface BassGuitar {
 void strum();
 default int getVolume() {return 10;}
}
class ElectricBass implements DoubleBass, BassGuitar {
 @Override public void strum() {System.out.print("A");}
}
public class RockBand {
 public static void main(String[] strings) {
 final class MyElectricBass extends ElectricBass {
 public void strum() {System.out.print("E");}
 }
 }
}
```
- A.** A  
**B.** E  
**C.** The code compiles and runs without issue but does not print anything.  
**D.** One line of code does not compile.  
**E.** Two lines of code do not compile.  
**F.** Three lines of code do not compile.
- 85.** Which NIO.2 Files methods return a Stream? (Choose three.)
- A.** find()  
**B.** lines()  
**C.** list()  
**D.** listFiles()  
**E.** readAllLines()  
**F.** walkFileTree()

# Appendix



# Answers to Review Questions

---

# Chapter 1: Java Basics

1. D. An entry point in a Java application consists of a `main()` method with a single `String[]` argument, return type of `void`, and modifiers `public` and `static`. The name of the variable in the input argument does not matter. Option A is missing the `static` modifier, Option B is missing the `String[]` argument, and Option C has the wrong access modifier and method name. Only Option D fulfills these requirements. Note that the modifier `final` is optional and may be added to an entry point method.
2. A. The diagram is an example of object-oriented design in Java, making Option B a true statement. Options C and D are also true, as they follow from the inheritance model in the diagram. Option A is the correct answer, since platform independence has nothing to do with the diagram.
3. C. The proper extension for a Java compiled bytecode file is `.class`, making Option C the correct answer.
4. B. The fact that the `Date` class exists in both packages does not impact the ability to import both packages, so lines 1 and 2 compile without issue, and Option A is incorrect. Line 4 will not compile because the `Date` class used is ambiguous, making Option B correct and Option D incorrect. Finally, Option C is incorrect because line 5 does compile, as the fully qualified name of the class is used.
5. A. Options B, C, and D are each attributes of traditional object-oriented programming. Option A is incorrect as an object-oriented project tends to group data and the actions related to that data into a single object.
6. D. Only local variables have such a small scope, making Option D the correct answer.
7. B. The package `java.lang` is imported into every Java class, so Option B is correct. The other options must be explicitly imported. Option A exists but must be explicitly imported. Options C and D do not exist in the standard Java runtime.
8. C. Java accepts Options A, B, and D as valid comments. Note that the `/* */` syntax can have additional (and uneven) star (\*) characters as shown in B and D. Option C is incorrect as hashtag (#) is not a valid comment character in Java.
9. D. A valid `.java` file may define any number of classes or interfaces but have at most one `public` class. It can also not define any `public` classes. For these reasons, Option A, B, and C are incorrect, leaving Option D as the only correct answer.
10. B. Notice in this question that `main()` is not a `static` method, therefore it can access both class and instance variables. Since there are two class variables and two instance variables defined, Option B is the correct answer.
11. B. A class will compile if it has unused or redundant `import` statements, making Option A and C incorrect. Option D is also incorrect as the compiler must be able to locate the class of the `import` statement. The correct answer is Option B. Removing unused `import` statements does not cause a class to become unable to be compiled.

12. A. The code does not compile because of line 5, making Option A the correct answer. For this question, it helps to understand variable scope. The `main()` method is `static` and does not have access to any class instance variables. The `birds` variable is not `static` and requires a class instance variable to access. Therefore, the code does not compile when the `static` method attempts to access a non-`static` variable without an instance of the class.
13. D. The `java` command can only execute compiled `.class` files, so I is false. Java is most certainly object oriented, one of the key design principles, so II is also false. The `javac` command compiles into bytecode, which must be run in a Java virtual machine (JVM), and is not native machine code, so III is false as well. Since none of the statements are true, Option D is the correct answer.
14. D. A class can start with a comment, an optional package statement, or an `import` statement if there is no package statement. It cannot start with a variable definition, making Option D the correct answer.
15. C. Classes may be defined without a package declaration and are placed in the default package, so Option A is incorrect. Option B is a completely false statement as no such file is required in Java. Option C is correct as it is one of the primary reasons for organizing your application into packages. Option D is incorrect as package-private allows access to methods and variables to be limited to those classes within the same package.
16. B. The compilation command requires the full or relative name of the file, including the `.java` extension, making Options A and C incorrect. The execution command requires the class name without a filename extension, making Option D incorrect. Option B is the only correct set of compilation and execution commands.
17. D. Encapsulation is the technique of removing access to a class's instance variables from processes outside the class, making Option D the correct answer.
18. D. The `height` variable is declared within the if-then statement block. Therefore, it cannot be referenced outside the if-then statement and the code does not compile.
19. A. A Java bytecode file is a binary encoded set of instructions designed to be run on any computer with a compatible JVM, making Option A the only correct answer. By compatible JVM, we mean one capable of running the class file. For example, a Java 6 JVM may have trouble executing a Java 8 compiled file. Option B is incorrect, and is more a facet of machine language compiled programming classes. Option C is also incorrect as binary data is not particularly human readable. Finally, Option D is incorrect as the compiled file can be distributed without its `.java` source file and execute without issue.
20. D. Unlike with some other programming languages, the proper way to terminate a line of code is with a semicolon (`;`), making D the only correct answer.
21. C. The code compiles and runs without issue, so Options A and B are incorrect. The question relies on your ability to understand variable scope. The variable `today` has local scope to the method in which it is executed. The variable `tomorrow` is re-declared in the method, but the reference used on line 7 is to the instance variable with a value of 10. Finally, the variable `tomorrow` is `static`. While using an instance reference to access a `static` variable is not recommended, it does not prevent the variable from being read. The result is line 7 evaluates and prints  $(20 + 10 + 1) = 31$ , making C the correct answer.

- 22.** C. Line 1 is missing the `class` keyword. Line 2 contains two types for the same variable. Line 3 is a valid definition for a method, making C the correct answer. Finally, line 4 contains an access modifier, `private`, after the return type, which is not allowed. In addition, `void` is an invalid type for variables.
- 23.** D. Platform independence is the property of Java that allows it to be run on a variety of different devices.
- 24.** A. Options B, C, and D are each correct statements about JVMs. Option A is incorrect. Not only is it not a statement about JVMs, it is actually false as Java bytecode can often be easily decoded/decompiled.
- 25.** B. There is no such thing as package variables, so Option A is incorrect. Option C is incorrect as the variable is only in scope within a specific instance of the class. Option D is also incorrect as the variable is only in scope for a single method that it is defined in. Option B is the only correct answer as class variables are in scope within the program.
- 26.** C. Option A is incorrect as the sub-package `recurring` is not included by the `import` statements. Option B is also incorrect as it uses the plural `directors` instead of the singular `director` used in the `import` statements. Option D is incorrect as the wildcard is applied to the sub-package `movie.director`, not the package `movie`. Finally, Option C is correct as it is a valid class accessible from the wildcard import.
- 27.** D. Java classes are defined in this order: package statement, `import` statements, class declaration, making Option D the only correct answer. Note that not all of these statements are required. For example, a class may not have a package statement, but if it does, it must come first in the file.
- 28.** D. The `import` statements for `stars.*` and `stars.Blackhole` are redundant `import` statements, since only the class `Blackhole` is used, and therefore one of them can be safely removed. The `import` statements `java.lang.*` and `java.lang.Object` are both not required as `java.lang` is automatically imported in every Java class. Therefore, three of the four `import` statements can be safely removed, making the correct answer Option D.
- 29.** C. The application prints the third argument of the input methods. Note that double quotes `""` group input arguments. Therefore, the third argument of Option A is `White-tailed deer`. The third argument of Option B is `3`. The third argument of Option C is `White-tailed`, making it the correct answer. Finally, Option D only has two input arguments, leading to an `ArrayIndexOutOfBoundsException` trying to read the third argument at runtime.
- 30.** B. The `javac` command compiles a `.java` file into a `.class` bytecode file, making Option B the correct answer.
- 31.** B. Java is object oriented, not procedural, so Option A is a false statement. Java allows method overloading in subclasses, so Option B is correct. Operator overloading is permitted in languages like C++, not Java, so Option C is also untrue. Finally, Option D is not a true statement as the JVM manages the location of objects in memory that can change and is transparent to the Java application.

32. D. Option A is incorrect as the return type of the method cannot be `null`. Option B is also incorrect as the return type cannot be `void` if the method uses a `return` statement. Option C is incorrect too as the `class` keyword is replaced with `int`. Option D is correct because it's the only answer that allows the code to compile without issue. Note that other values are possible for this question. For example, either `int` or `long` can be entered in the last blank. The key here is that only one of the available answer choices allows the code to compile.
33. A. The code compiles so Option D is incorrect. The input to the constructor is ignored, making the assignment of `end` to be 4. Since `start` is 2, the subtraction of 4 by 2 results in the application printing 2, followed by 5, making Option A the correct answer.
34. D. Option A is a false statement, while Options B and C are actually arguments against using inheritance. Option D is one of the most important reasons Java supports inheritance, to allow increased code reuse among classes.
35. A. The double slash `(//)` syntax can have any number of slashes as a comment, so long as it starts with two of them, making Option A the correct answer. The `(#)` is not a comment character in Java, regardless of whether it is followed by a `(!)`, so Option B is incorrect. Option C is incorrect as a single slash `(/)` is not a valid comment in Java. Finally, Option D is incorrect as Option A is a valid comment.
36. B. An entry point in a Java application consists of a `main()` method with a single `String[]` argument, return type of `void`, and modifiers `public` and `static`. Option D is the typical syntax for this method, although Options A and C are also valid forms of this method. Note that the modifier `final` is optional and may be added to the method signature. Furthermore, the `main()` method may take a vararg or array. Option B is the only invalid declaration as it does not take an array as an argument.
37. B. The line of code cannot be inserted at `a1` because no variables are allowed outside of the class declaration in this file, making Options A and D incorrect. The line of code can also not be inserted at `a3` as local variables defined within methods cannot have access modifiers such as `public`, making Option C incorrect. The code can be inserted independently at `a2` and `a4` as instance variables can be defined anywhere in the class outside a method. Therefore, Option B is the correct choice.
38. A. Option A is the only correct answer as a `class` definition is the only required component in a Java class file. Note that we said a Java class file here; Java also allows interfaces and enums to be defined in a file. A `package` statement and `import` statements are optional for declaring a class, making Options B and C incorrect. A class may also be defined with package-level access in a file, making Option D an incorrect answer.
39. D. The proper extension for a Java compiled bytecode file is `.java`, making Option D the correct answer.
40. C. Remember that `java.lang` is automatically imported in all Java classes, therefore both `java.lang.Math` and `pocket.complex.Math` are both imported into this class. Importing both sets of packages does not cause any compilation issues, making Option A incorrect. Line 3 is unnecessary import but including it does not prevent the class from compiling, making Option B incorrect. While both versions of `Math` may be imported into the class, the usage of the `Math` class requires a package name. Because of this, line 6 does not compile as the class reference is ambiguous, making Option C the correct answer and Option D incorrect.

- 41.** A. Options B and C are accessible within the class as they are covered by the `import` statements. Option D is also fine as `java.lang.Object` is available without an explicit import. The only class not automatically accessible within the class without the full package name is `dog.puppy.female.KC` as the `import` statements do not include sub-packages; therefore, Option A is the correct answer.
- 42.** B. Object-oriented programming is the technique of structuring data into objects, which may contain data and a set of actions that operate on the data, making Option B the correct answer.
- 43.** A. All of the `import` statements in this class are required. Removing any of them would cause the class to not compile, making Option A the correct answer.
- 44.** C. The `numLock` variable is not accessible in the `static main()` method without an instance of the `Keyboard` class; therefore, the code does not compile, and Option C is the correct answer.
- 45.** D. The code compiles and runs without issue, so Option A is incorrect. The question involves understanding the value and scope of each variable at the `print()` statement. The variables `feet` and `tracks` are locally scoped and set to 4 and 15, respectively, ignoring the value of `tracks` of 5 in the instance of the class. Finally, the `static` variable `s.wheels` has a value of 1. The result is the combined value is 20, making Option D the correct answer.
- 46.** B. First off, the `color` variable defined in the instance and set to `red` is ignored in the method `printColor()` as local scope overrides instance scope, so Option A is incorrect. The value of `color` passed to the `printColor()` method is `blue`, but that is lost by the assignment to `purple`, making Option B the correct answer and Option C incorrect. Option D is incorrect as the code compiles and runs without issue.
- 47.** C. The `javac` command takes a text-based `.java` file and returns a binary bytecode `.class` file, making II a true statement. The `java` command uses a period (.) to separate packages, not a slash (/), making I a true statement and III a false statement. For these reasons, Option C is the correct answer.
- 48.** D. The application compiles without issue, so Option C is incorrect. The application does not execute though, as the `main()` method does not have the correct method signature. It is missing the required input argument, an array of `String`. Trying to execute the application without a proper entry point produces an error, making Option D the correct answer.
- 49.** C. Option A does not compile because it is missing the closing bracket for the class. Option D does also not compile as `void` is not a valid type for a variable. Regardless, Options A and D are incorrect as they are missing the `getRating()` method. Note that Option A also uses an abbreviation for `numberOfPages`. Option B is incorrect as it is missing the `numberOfPages` attribute. Option C is the correct answer as it properly defines the attribute `numberOfPages` and method `getRating()`.
- 50.** C. Garbage collection can happen at any time while an application is running, especially if the available memory suddenly becomes low, making Option A incorrect. Option B is also incorrect, since it is trivial to create a Java application with an infinite loop that never terminates. Option D is incorrect because the computer must be able to run the JVM in order to execute a Java class. Option C is the only correct answer, as the JVM does require an entry point method to begin executing the application.

# Chapter 2: Working with Java Data Types

1. A. Option A does not compile because Java does not allow declaring different types as part of the same declaration. The other three options show various legal combinations of combining multiple variables in the same declarations with optional default values.
2. D. The `table` variable is initialized to "metal". However, `chair` is not initialized. In Java, initialization is per variable and not for all the variables in a single declaration. Therefore, the second line tries to reference an uninitialized local variable and does not compile, which makes Option D correct.
3. B. Instance variables have a default value based on the type. For any non-primitive, including `String`, that type is a reference to `null`. Therefore Option B is correct. If the variable was a local variable, Option C would be correct.
4. B. An identifier name must begin with a letter, `$`, or `_`. Numbers are only permitted for subsequent characters. Therefore, Option B is not a valid variable name.
5. B. In Java, class names begin with an uppercase letter by convention. Then they use lowercase with the exception of new words. Option B follows this convention and is correct. Option A follows the convention for variable names. Option C follows the convention for constants. Option D doesn't follow any Java conventions.
6. C. Objects have instance methods while primitives do not. Since `int` is a primitive, you cannot call instance methods on it. `Integer` and `String` are both objects and have instance methods. Therefore, Option C is correct.
7. C. Underscores are allowed between any two digits in a numeric literal. Underscores are not allowed at the beginning or end of the literal, making Option C the correct answer.
8. C. Option A is incorrect because `int` is a primitive. Option B is incorrect because it is not the name of a class in Java. While Option D is a class in Java, it is not a wrapper class because it does not map to a primitive. Therefore, Option C is correct.
9. C. There is no class named `integer`. There is a primitive `int` and a class `Integer`. Therefore, the code does not compile, and Option C is correct. If the type was changed to `Integer`, Option B would be correct.
10. C. The `new` keyword is used to call the constructor for a class and instantiate an instance of the class. A primitive cannot be created using the `new` keyword. Dealing with references happens after the object created by `new` is returned.
11. D. Java uses the suffix `f` to indicate a number is a `float`. Java automatically widens a type, allowing a `float` to be assigned to either a `float` or a `double`. This makes both lines `p1` and `p3` compile. Line `p2` does compile without a suffix. Line `p4` does not compile without a suffix and therefore is the answer.

- 12.** A. A byte is smaller than a char, making Option C incorrect. bigint is not a primitive, making Option D incorrect. A double uses twice as much memory as a float variable, therefore Option A is correct.
- 13.** D. The instance variables, constructor, and method names can appear in any order within a class declaration.
- 14.** B. Java does not allow multiple Java data types to be declared in the same declaration, making Option B the correct answer. If double was removed, both hot and cold would be the same type. Then the compiler error would be on x3 because of a reference to an uninitialized variable.
- 15.** C. Lines 2 and 7 illustrate instance initializers. Line 6 is a static initializer. Lines 3–5 are a constructor.
- 16.** A. Since defaultValue is a local variable, it is not automatically initialized. That means the code will not compile with any type. Therefore, Option A is correct. If this was an instance variable, Option C would be correct as int and short would be initialized to 0 while double would be initialized to 0.0.
- 17.** A. The finalize() method may not be called, such as if your program crashes. However, it is guaranteed to be called no more than once.
- 18.** D. String is a class, but it is not a wrapper class. In order to be a wrapper class, the class must have a one-to-one mapping with a primitive.
- 19.** C. Lines 15–17 create the three objects. Lines 18–19 change the references so link2 and link3 point to each other. The lines 20–21 wipe out two of the original references. This means the object with name as x is inaccessible.
- 20.** C. Options A and D are incorrect because byte and short do not store values with decimal points. Option B is tempting. However, 3.14 is automatically a double. It requires casting to float or writing 3.14f in order to be assigned to a float. Therefore, Option C is correct.
- 21.** B. Integer is the name of a class in Java. While it is bad practice to use the name of a class as your local variable name, this is legal. Therefore, k1 does compile. It is not legal to use a reserved word as a variable name. All of the primitives including int are reserved words. Therefore, k2 does not compile, and Option B is the answer. Line k4 doesn't compile either, but the question asks about the first line to not compile.
- 22.** B. Dot notation is used for both reading and writing instance variables, assuming they are in scope. It cannot be used for referencing local variables, making Option B the correct answer.
- 23.** C. Class names follow the same requirements as other identifiers. Underscores and dollar signs are allowed. Numbers are allowed, but not as the first character of an identifier. Therefore, Option C is correct. Note that class names begin with an uppercase letter by convention, but this is not a requirement.

24. D. This question is tricky as it appears to be about primitive vs. wrapper classes. Looking closely, there is an underscore right before the decimal point. This is illegal as the underscore in a numeric literal can only appear between two digits.
25. C. Local variables do not have a default initialization value. If they are referenced before being set to a value, the code does not compile. Therefore, Option C is correct. If the variable was an instance variable, Option B would be correct. Option D is tricky. A local variable will compile without an initialization if it isn't referenced anywhere or it is assigned a value before it is referenced.
26. C. Since `defaultValue` is an instance variable, it is automatically initialized to the corresponding value for that type. For `double`, that value is `0.0`. By contrast, it is `0` for `int`, `long`, and `short`. Therefore Option C is correct.
27. B. Option B is an example of autoboxing. Java will automatically convert from primitive to wrapper class types and vice versa. Option A is incorrect because you can only call methods on an object. Option C is incorrect because this method is used for converting to a wrapper class from a `String`. Option D is incorrect because autoboxing will convert the primitive to an object before adding it to the `ArrayList`.
28. C. Java does not allow calling a method on a primitive. While autoboxing does allow the assignment of an `Integer` to an `int`, it does not allow calling an instance method on a primitive. Therefore, the last line does not compile.
29. D. In order to call a constructor, you must use the `new` keyword. It cannot be called as if it was a normal method. This rules out Options A and B. Further, Option C is incorrect because the parentheses are required.
30. A. Option A (I) correctly assigns the value to both variables. II does not compile as `dog` does not have a type. Notice the semicolon in that line, which starts a new statement. III compiles but only assigns the value to `dog` since a declaration only assigns to one variable rather than everything in the declaration. IV does not compile because the type should only be specified once per declaration.
31. C. The wrapper class for `int` is `Integer` and the wrapper class for `char` is `Character`. All other primitives have the same name. For example, the wrapper class for `boolean` is `Boolean`.
32. A. Assuming the variables are not primitives, they allow a `null` assignment. The other statements are false.
33. A. An example of a primitive type is `int`. All the primitive types are lowercase, making Option A correct. Unlike object reference variables, primitives cannot reference `null`. `String` is not a primitive as evidenced by the uppercase letter in the name and the fact that we can call methods on it. You can create your own classes, but not primitives.
34. D. While you can suggest to the JVM that it might want to run a garbage collection cycle, the JVM is free to ignore your suggestion. Option B is how to make this suggestion. Since garbage collection is not guaranteed to run, Option D is correct.

- 35.** C. All three references point to the `String` `apple`. This makes the other two `String` objects eligible for garbage collection and Option C correct.
- 36.** B. A constructor can only be called with a class name rather than a primitive, making Options A and C incorrect. The newly constructed `Double` object can be assigned to either a `double` or `Double` thanks to autoboxing. Therefore, Option B is correct.
- 37.** B. First line 2 runs and sets the variable using the declaration. Then the instance initializer on line 6 runs. Finally, the constructor runs. Since the constructor is the last to run of the three, that is the value that is set when we print the result, so Option B is correct.
- 38.** C. Objects are allowed to have a `null` reference while primitives cannot. `int` is a primitive, so assigning `null` to it does not compile. `Integer` and `String` are both objects and can therefore be assigned a `null` reference. Therefore, Option C is correct.
- 39.** C. An instance variable can only be referenced from instance methods in the class. A `static` variable can be referenced from any method. Therefore, Option C is correct.
- 40.** B. Underscores are allowed between any two digits in a numeric literal. Underscores are not allowed adjacent to a decimal point, making Option B the correct answer.
- 41.** A. These four types represent nondecimal values. While you don't need to know the exact sizes, you do need to be able to order them from largest to smallest. A `byte` is smallest. A `short` comes next, followed by `int` and then `long`. Therefore, Option A is correct.
- 42.** A. Java uses dot notation to reference instance variables in a class, making Option A correct.
- 43.** B. If there was a `finalize()` method, this would be a different story. However, the method here is `finalizer`. Tricky! That's just a normal method that doesn't get called automatically. Therefore `clean` is never output.
- 44.** A. Options B and C do not compile. In Java, braces are for arrays rather than instance variables. Option A is the correct answer. It uses dot notation to access the instance variable. It also shows that a private variable is accessible in the same class and that a narrower type is allowed to be assigned to a wider type.
- 45.** B. The `parseInt()` methods return a primitive. The `valueOf()` methods return a wrapper class object. In real code, autoboxing would let you assign the return value to either a primitive or wrapper class. In terms of what gets returned directly, Option B is correct.
- 46.** B. On line 9, all three objects have references. The `elena` and `zoe` objects have a direct reference. The `diana` object is referenced through the `elena` object. On line 10, the reference to the `diana` object is replaced by a reference to the `zoe` object. Therefore, the `diana` object is eligible to be garbage collected, and Option B is correct.
- 47.** C. Options A and B are `static` methods rather than constructors. Option D is a method that happens to have the same name as the class. It is not a constructor because constructors don't have return types.

48. A. Remember that garbage collection is not guaranteed to run on demand. If it doesn't run at all, Option B would be output. If it runs at the requested point, Option C would be output. If it runs right at the end of the `main()` method, Option D would be output. Option A is the correct answer because `play` is definitely called twice. Note that you are unlikely to see all these scenarios if you run this code because we have not used enough memory for garbage collection to be worth running. However, you still need to be able to answer what could happen regardless of it being unlikely.
49. B. Each wrapper class has a constructor that takes the primitive equivalent. The methods mentioned in Options A, C, and D do not exist.
50. C. The `main()` method calls the constructor which outputs `a`. Then the main method calls the `run()` method. The `run()` method calls the constructor again, which outputs `a` again. Then the `run()` method calls the `Sand()` method, which happens to have the same name as the constructor. This outputs `b`. Therefore, Option C is correct.

## Chapter 3: Using Operators and Decision Constructs

1. B. A `switch` statement supports the primitive types `byte`, `short`, `char`, and `int` and the classes `String`, `Character`, `Byte`, `Short`, and `Integer`. It also supports enumerated types. Floating-point types like `float` and `double` are not supported, therefore Option B is the correct answer.
2. A. Remember that in ternary expressions, only one of the two right-most expressions are evaluated. Since `meal>6` is `false`, `--tip` is evaluated and `++tip` is skipped. The result is that `tip` is changed from 2 to 1, making Option A the correct answer. The value of `total` is 6, since the pre-increment operator was used on `tip`, although you did not need to know this to solve the question.
3. C. The first assignment creates a new `String "john"` object. The second line explicitly uses the `new` keyword, meaning a new `String` object is created. Since these objects are not the same, the `==` test on them evaluates to `false`. The `equals()` test on them returns `true` because the values they refer to are equivalent. Therefore, the correct answer is C.
4. D. This code does not compile because it has two `else` statements as part of a single if-then statement. Notice that the second `if` statement is not connected to the last `else` statement. For this reason, Option D, none of the above, is the correct answer.
5. C. A `default` statement inside a `switch` statement is optional and can be placed in any order within the `switch`'s `case` statements, making Options A and B incorrect. Option D is an incorrect statement as a `switch` statement can be composed of a single `default` statement and no `case` statements. Option C is correct because a `default` statement does not take a value, unlike a `case` statement.

6. B. The initial assignment of `thatNumber` follows the first branch of the ternary expression. Since `5 >= 5` evaluates to `true`, a value of 3 is assigned to `thatNumber`. In the next line, the pre-increment operator increments the value of `thatNumber` to 4 and returns a value of 4 to the expression. Since `4 < 4` evaluates to `false`, the if-then block is skipped. This leaves the value of `thatNumber` as 4, making Option B the correct answer.
7. B. The `break` statement exits a `switch` statement, skipping all remaining branches, making Option B the correct answer. In Option A, `exit` is not a statement in Java. In Option C, `goto` is a reserved word but unused in Java. Finally, in Option D, `continue` is a statement but only used for loops.
8. C. Option A is incorrect as only one of the two right-hand expressions is evaluated at runtime. Parentheses are often helpful for reading ternary expressions but are not required, making Option B incorrect. Option C is a correct statement about ternary operators as they are commonly used to replace short if-then-else statements. Finally, Option D is incorrect as only boolean expressions are permitted in the left-most operand of a ternary expression.
9. C. On line 4, `candidateA` and `candidateB` are numbers, but the `&&` operation can only be applied to boolean expressions. Therefore, the code does not compile because of line 4, making C the correct answer. All of the other lines are correct. Note that if line 4 is fixed, line 3 does not produce a `NullPointerException` at runtime. The conditional `||` and the preceding null check allows the code to only call `intValue()` if `candidateA` is not `null`.
10. A. The first step is to determine whether or not the if-then statement's expression is executed. The expression `6 % 3` evaluates to 0, since there is no remainder, and since `0 >= 1` is `false`, the expression `triceratops++` is not called. Notice there are no brackets `{}` in the if-then statement. Despite the `triceratops--` line being indented, it is not part of the if-then statement. Recall that Java does not use indentation to determine the beginning or end of a statement. Therefore, `triceratops--` is always executed, resulting in a value of 2 for `triceratops` and making Option A the correct answer.
11. D. Option A is incorrect because `else` statements are entirely optional. Option B is also incorrect. The target of an if-then statement is not evaluated if the boolean test is `false`. Option C is incorrect. While an if-then statement is often used to test whether an object is of a particular type in order to cast it, it is not required to cast an object. Option D is correct as an if-then statement may execute a single statement or a block of code `{}`.
12. D. For this question, it helps to notice that the second if-then statement is not connected to the first if-then statement, as there is no `else` joining them. When this code executes, the first if-then statement outputs `Not enough` since `flair` is  $\geq 15$  and  $< 37$ . The second if-then statement is then evaluated. Since `flair` is not 37, the expression `Too many` is outputted. Since two statements are outputted, Option D, none of the above, is the correct answer.
13. B. A case value must be a constant expression, such as a literal or final variable, so Options A and C are true statements about case values. A case statement may be terminated by a `break` statement, but it is not required, making Option B the false statement and correct answer. Option D is also a true statement about case values.

14. D. The question is about boolean operators. Since Options A and B are numeric operators, they can be instantly disregarded. The question then simplifies to which boolean expression, `&&` or `||`, corresponds to the truth table that only evaluates to `true` if both operands are `true`. Only the conjunctive logical `&&` operator represents this relationship, making Option D the correct answer.
15. C. The value of `jumps` and `hops` is unimportant because this code does not compile, making Option C the correct answer. Unlike some other programming languages, Java does not automatically convert integers to boolean values for use in if-then statements. The statement `if(jumps)` evaluates to `if(0)`, and since 0 is not a boolean value, the code does not compile. Note that the value of the `jumps` variable is irrelevant in this example; no integer evaluates to a boolean value in Java.
16. B. Prefix operators modify the variable and evaluate to the new value, while postfix operators modify the variable but return the original value. Therefore, Option B is the correct answer.
17. B. For this problem, it helps to recognize that parentheses take precedence over the operations outside the parentheses. Once we replace the variables with values, the expression becomes:  $3+2*(2+3)$ . We then calculate the value inside the parentheses to get  $3+2*5$ . Since the multiplication operator has higher precedence than addition, we evaluate it first, resulting in  $3+10 = 13$ , making Option B the correct answer.
18. B. Any value that can be implicitly promoted to `int` will work for the `case` statement with an `int` input. Since `switch` statements do not support `long` values, and `long` cannot be converted to `int` without a possible loss of data, Option B is the correct answer.
19. D. While parentheses are recommended for ternary operations, especially embedded ones, they are not required, so Option C is incorrect. The code does not compile because `day` is an `int`, not a boolean expression, in the second ternary operation, making Option D the correct answer. Remember that in Java, numeric values are not accepted in place of boolean expressions in if-then statements or ternary operations.
20. C. While the code involves numerous operations, none of that matters for solving this problem. The key to solving it is to notice that the line that assigns the `leaders` variable has an uneven number of parentheses. Without balanced parentheses, the code will not compile, making Option C the correct answer.
21. B. Remember that Java evaluates `+` from left to right. The first two values are both numbers, so the `+` is evaluated as numeric addition, resulting in a reduction to  $11 + "7" + 8 + 9$ . The next two terms,  $11 + "7"$ , are handled as string concatenation since one of the terms is a `String`. This allows us to reduce the expression to `"117" + 8 + 9`. Likewise, the final two terms are each evaluated one at a time with the `String` on the left. Therefore, the final value is 11789, making Option B the correct answer.
22. B. The subtraction `-` operator is used to find the difference between two numbers, while the modulus `%` operator is used to find the remainder when one number is divided by another, making Option B the correct answer. The other options use operators that do not match this description.

- 23.** B. The code compiles without issue, making Option D incorrect. The focus of this question is showing how the division and modulus of two numbers can be used to reconstitute one of the original operands. In this example, `partA` is the integer division of the two numbers. Since 3 does not divide 11 evenly, it is rounded down to 3. The variable `partB` is the remainder from the first expression, which is 2. The newDog variable is an expression that reconstitutes the original value for `dog` using the division value and the remainder. Note that due to operator precedence, the multiplication `*` operation is evaluated before the addition `+` operation. The result is the original value of 11 for `dog` is outputted by this program.
- 24.** B. The code compiles without issue, so Option D is incorrect. In this question's `switch` statement, there are no `break` statements. Once the matching `case` statement, 30, is reached, all remaining `case` statements will be executed. The variable `eaten` is increased by 1, then 2, then reduced by 1, resulting in a final value of 2, making Option B the correct answer.
- 25.** C. Ternary operations require both right-hand expressions to be of compatible data types. In this example, the first right-hand expression of the outer ternary operation is of type `String`, while the second right-hand expression is of type `int`. Since these data types are incompatible, the code does not compile, and Option C is the correct answer.
- 26.** A. For this question, remember that if two `String` objects evaluate to `true` using `==`, then they are the same object. If they are the same `String` object, `equals()` will trivially return `true`. Option A correctly reflects this principle. Option B is incorrect as two `String` objects that are not the same may still be equivalent in terms of `equals()`. For example, `apples == new String(apples)` evaluates to `false`, but `equals()` will evaluate to `true` on these `String` objects. Likewise, Options C and D are also incorrect because two `String` objects that are equivalent in terms of `equals()` may be different objects.
- 27.** B. The statement compiles and runs without issue, making Options C and D incorrect. Since we are given that `myTestVariable` is not `null`, the statement will always evaluate to `false`, making Option B the correct answer. Note that if `myTestVariable` was `null`, then the code would still compile but throw a `NullPointerException` calling `equals()` at runtime.
- 28.** D. The code does not compile, making Option D the correct answer. The reason the code does not compile is due to the test in the second if-then statement. The expression `(streets && intersections > 1000)` is invalid because `streets` is not a boolean expression and cannot be used as the left-hand side of the conjunctive logical `&&` operator. The line of code is designed to resemble the corrected expression `(streets > 1000 && intersections > 1000)`. Notice the fixed expression requires two relational `>` operators. If the second if-then statement was corrected, then the application would compile and produce two 1's, making Option C the correct answer.
- 29.** B. The `&` and `&&` (AND) operators are not interchangeable, as the conjunctive `&` operator always evaluates both sides of the expression, while the conditional conjunctive `&&` operator only evaluates the right-hand side of the expression if the left side is determined to be `true`. This is why conditional operators are often referred to as short-circuit operators, skipping the right-hand side expression at runtime. For these reasons, Option B is the correct answer. Note that Option C is an incorrect statement as well, since it describes disjunctive (OR) operators.

30. C. The code compiles, so Option A is incorrect. Since `w` starts out `true`, the third line takes the first right-hand side of the ternary expression returning and assigning 5 to `x` (post-increment operator) while incrementing `y` to 6. Note that the second right-hand side of the ternary expression `y--` is not evaluated since ternary operators only evaluate one right-hand expression at runtime. On the fourth line, the value of `w` is set to `!z`. Since `z` is `false`, the value of `w` remains `true`. The final line outputs the value of `(5+6)` and `(true ? 5 : 10)`, which is `11 5`, making Option C the correct answer.
31. A. The first assignment actually uses two `String` objects, the literal "bob" and the `String` created with the `new` keyword. Regardless, only the second object is assigned to the variable `bob`. The second variable, `notBob`, is assigned a reference to the value of the `bob` variable. This means that not only does the `equals()` test pass, but they are actually the same object, so the `==` test is `true` as well. Therefore, the correct answer is Option A.
32. B. The question is about operator precedence and order of operation. The multiplication `*` and modulus `%` operators have the highest precedence, although what is inside the parentheses needs to be evaluated first. We can reduce the expression to the following: `12 + 6 * 3 % 2`. Since multiplication `*` and modulus `%` have the same operator precedence, we evaluate them from left to right as follows:  $12 + 6 * 3 \% 2 \rightarrow 12 + 18 \% 2 \rightarrow 12 + 0 \rightarrow 12$ . We see that despite all of the operators on the right-hand side of the expression, the result is zero, leaving us a value of 12, making Option B the correct answer.
33. D. The XOR `^` operator evaluates to `true` if `p` and `q` differ and `false` if they are the same. Therefore, the missing values are `true` and `false`, making Option D the correct answer.
34. C. The key to understanding this question is to remember that the conditional conjunction `&&` operator only executes the right-hand side of the expression if the left-hand side of the expression is `true`. If `data` is an empty array, then the expression ends early and nothing is output. The second part of the expression will return `true` if `data`'s first element is `sound` or `logic`. Since we know from the first part of the statement that `data` is of length at least one, no exception will be thrown. The final part of the expression with `data.length<2` doesn't change the output when `data` is an array of size one. Therefore, `sound` and `logic` are both possible outputs. For these reasons, Option C is the only result that is unexpected at runtime.
35. C. In Option A, the division operator `/` incorrectly comes after the decrement `--` operator. In Option B, the subtraction operator `-` incorrectly comes after the modulus `%` operator. In Option D, the division operator `/` incorrectly comes after the subtraction `-` operator. The correct answer is Option C, where all three operators have the same order of precedence.
36. D. The exclusive or (XOR) `^` operator requires evaluating both operands to determine the result. For this reason, Options A and B are incorrect. For Option B, you can't have a short-circuit operation if both operands are always read, therefore `^^` does not exist. Option C is an incorrect statement as the `^` operator only returns `true` if exactly one operand is `true`. Finally, Option D is correct as the `^` is only applied to `boolean` values in Java.
37. C. The diagram represents the overlap of `x` and `y`, corresponding to when one of them is `true`. Therefore, `x || y`, Option C, most closely matches this relationship. Note that `z` is unused in the diagram and therefore is not required in any expression.

- 38.** D. The value of a case statement must be constant, a literal value, or final variable. Since red is missing the final attribute, no variable type allows the code to compile, making Option D the correct answer.
- 39.** C. The question is asking which operator represents greater than or equal to and which operator is strictly less than. The `>=` and `<` correspond to these operators, respectively. Therefore, Option C is the correct answer. Note that the question does not specify which order the operators needed to appear in, only to select the two operators that match the question description.
- 40.** B. The code compiles and runs without issue, making Options C and D incorrect. The key here is understanding operator precedence and applying the parentheses to override precedence correctly. The first expression is evaluated as follows:  
$$10 * (2 + (3 + 2) / 5) \rightarrow 10 * (2 + 5 / 5) \rightarrow 10 * (2 + 1) \rightarrow 10 * 3$$
, with a final value of 30 for turtle. Since turtle is not less than 5, a value of 25 is assigned to hare. Since turtle is not less than hare, the last expression evaluates to `Turtle wins!`, which is outputted to the console, making Option B the correct answer.
- 41.** A. All of the terms of `getResult()` in this question evaluate to 0, since they are all less than or equal to 5. The expression can therefore be reduced to `0+0+0+0+"!"`. Since Java evaluates the `+` operator from left to right, the four operands on the left are applied using numeric addition, resulting in the expression `0+"!"`. This expression just converts the value to a `String`, resulting in an output of 0, making Option A the correct answer.
- 42.** A. The code compiles without issue, so Option D is incorrect. The key here is that the if-then statement in the `runTest()` method uses the assignment operator (`=`) instead of the (`==`) operator. The result is that `spinner` is assigned a value of `true`, and the statement `(spinner = roller)` returns the newly assigned value. The method then returns up, making Option A the correct answer. If the (`==`) operator had been used in the if-then statement, then the process would have branched to the `else` statement, with `down` being returned by the method.
- 43.** D. The conditional disjunction (OR) `||` operator is `true` if either of the operands are `true`, while the logical complement (`!`) operator reverses or flips a boolean value, making Option D the correct answer. The other options use operators that do not match this description. In particular, Options A and C include operators that can only be applied to numerical values, not boolean ones.
- 44.** A. While parentheses are recommended for ternary operations, especially embedded ones, they are not required, so Option C is incorrect.. The first ternary operation evaluates `characters <= 4` as `false`, so the second ternary operation is executed. Since `story > 1` is `true`, the final value of `movieRating` is `2.0`, making Option A the correct answer.
- 45.** B. Barring any JVM limitations, a `switch` statement can have any number of `case` statements (including none) but at most one `default` statement, with Option B correctly identifying this relationship.
- 46.** A. The application uses the conditional conjunction `&&` operator to test if `weather[0]` is `null`, but unfortunately this test does not work on zero-length arrays. Therefore, it is possible this code will throw an `ArrayIndexOutOfBoundsException` at runtime. The second

part of the expression evaluates to `true` if the first input of `weather` matches `sunny`. The final part of the expression, `&& !false`, is a tautology in that it is always `true` and has no impact on the expression. Either an exception will be thrown or text will be output, based on the value of `weather`, therefore Option A is the correct answer.

47. D. The question looks a lot more difficult than it is. In fact, to solve it you don't have to compute anything! You just have to notice that the logical complement operator (`!`), which can only be applied to boolean values, is being applied to a numeric value. Therefore, the answer is that the expression wouldn't compile or run, making Option D the correct answer.
48. C. The disjunctive logical `||` operator evaluates to `true` if either operand is `true`. Another way to look at it is that it only evaluates to `false` if both operands are `false`. Therefore, the missing values are both `true`, making Option C the correct answer.
49. A. In Option B, the subtraction operator `-` incorrectly comes after the decrement `--` operator. In Option C, the division operator `/` incorrectly comes after the increment `++` operator. In Option D, the modulus operator `%` incorrectly comes after the increment `++` operator. The correct answer is Option A, where the subtraction `-` and addition `+` operators are followed by the division `/` and multiplication `*` operators.
50. C. The key to solving this problem is remembering that the type of the value returned by a ternary operation is determined by the expressions on the right-hand side. On line `p1`, the expressions are of type `int`, but the assignment is to the variable `game`, of type `String`. Since the assignment is invalid, the code does not compile, and Option C is correct.

## Chapter 4: Creating and Using Arrays

1. B. Three dots `(...)` are the syntax for a method parameter of type `varargs`. It is treated like an array.
2. B. Array indexes are zero based in Java. A `varargs` parameter is simply another way of passing in data to a method. From within the method, it is treated just like you had written `Frisbee[] f` as the method parameter. Therefore, the first element uses the 0th index, and Option B is correct.
3. D. Trick question! While `int` is a primitive, all arrays are objects. One way to tell is that an array has a public instance variable called `length`. Another way is that you can assign it a variable of type `Object`. Therefore, Option D is correct.
4. C. The array braces are allowed to appear before or after the variable name, making the `tiger` and `bear` declarations correct. The braces are not allowed to appear before the type making the `lion` declaration incorrect. Therefore, Option C is correct.
5. C. From within a method, an array or `varargs` parameter is treated the same. However, there is a difference from the caller's point of view. A `varargs` parameter can receive either an array or individual values, making Options A and B compile. However, an array parameter can only take an array, which prevents Option C from compiling.

6. A. Arrays use the `length` variable to determine the number of elements, making Option A correct. For an `ArrayList`, Option D would have been the answer.
7. C. A two-dimensional array is declared by listing both sizes in separate pairs of braces. Option C correctly shows this syntax.
8. B. There is nothing wrong with this code. It correctly creates a seven-element array. The loop starts with index 0 and ends with index 6. Each line is correctly output. Therefore, Option B is correct.
9. B. Sorry. This is just something you have to memorize. The `sort()` and `binarySearch()` methods do sorting and searching, respectively.
10. B. The elements of the array are of type `String` rather than `int`. Therefore, we use alphabetical order when sorting. The character 1 sorts before the character 9, alphabetically making Option A incorrect. Shorter strings sort before longer strings when all the other characters are the same, making Option B the answer.
11. B. Array indices start with 0, making Options C and D incorrect. The `length` attribute refers to the number of elements in an array. It is one past the last valid array index. Therefore, Option B is correct.
12. C. When using an array initializer, you are not allowed to specify the size separately. The size is inferred from the number of elements listed. Therefore, `tiger` and `ohMy` are incorrect. When you're not using an array initializer, the size is required. An empty array initializer is allowed. Option C is correct because `lion` and `bear` are legal.
13. B. Since no elements are being provided when creating the arrays, a size is required. Therefore, `lion` and `bear` are incorrect. The braces containing the size are required to be after the type, making `ohMy` incorrect. The only one that is correct is `tiger`, making the correct answer Option B.
14. C. The `binarySearch()` method requires a sorted array in order to return a correct result. If the array is not sorted, the results of a binary search are undefined.
15. A. An `ArrayList` expands automatically when it is full. An array does not, making Option A the answer. The other three statements are true of both an array and an `ArrayList`.
16. C. This code creates a two-dimensional array of size  $1 \times 2$ . Lines `m1` and `m2` assign values to both elements in the outer array. Line `m3` attempts to reference the second element of the outer array. Since there is no such position, it throws an exception, and Option C is correct.
17. B. The code sorts before calling `binarySearch()`, so it meets the precondition for that method. The target string of "Mac" is the second element in the sorted array. Since array indices begin with zero, the second position is index 1, and Option B is correct.
18. A. A multi-dimensional array is created with multiple sets of size parameters. The first line should be `char[] ticTacToe = new char[3][3];`. Therefore, Option A is the answer.
19. B. The first line creates one object; the array itself. While there are four references to `null` in that array, none of those are objects. The second line creates one object and points one

of the array references to it. So far there are two objects: the array itself and one object it is referencing. The third line does the same, bringing up the object count to three. Therefore, Option B is correct.

20. B. As with a one-dimensional array, the braces must be after the type, making `alpha` and `beta` illegal declarations. For a multi-dimensional array, the braces are allowed to be before and/or after the variable name. They do not need to be in the same place. Therefore, the remaining three are correct, and Option B is correct.
21. B. Options A, C and D represent  $3 \times 3$  2D arrays. Option B best represents the array in the code. It shows there are three different arrays of different lengths.
22. D. `names.length` is the number of elements in the array. The last valid index in the array is one less than `names.length`. In Java, arrays do not resize automatically. Therefore, the code throws an `ArrayIndexOutOfBoundsException`.
23. C. The code `days.size()` would be correct if this was an `ArrayList`. Since it is an array, `days.length` is the correct code. Therefore, the code does not compile, and Option C is the answer.
24. C. Since the braces in the declaration are before the variable names, the variable type `boolean[][][]` applies to both variables. Therefore, both `bools` and `moreBools` can reference a 3D array.
25. C. Calling `toString()` on an array doesn't output the contents of the array, making Option C correct. If you wanted Option A to be the answer, you'd have to call `Arrays.toString(strings)`.
26. B. Arrays begin with an index of 0. This array is a  $3 \times 3$  array. Therefore, only indexes 0, 1, and 2 are valid. Line `r2` throws an `ArrayIndexOutOfBoundsException`. Therefore, Option B is correct.
27. D. Three dots in a row is a varargs parameter. While varargs is used like an array from within the method, it can only be used as a method parameter. This syntax is not allowed for a variable, making Option D the answer.
28. D. Line 6 assigns an `int` to a cell in a 2D array. This is fine. Line 7 casts to a general `Object[]`. This is dangerous, but legal. Why is it dangerous, you ask? That brings us to line 8. The compiler can't protect us from assigning a `String` to the `int[]` because the reference is more generic. Therefore, line 8 throws an `ArrayStoreException` because the type is incorrect, and Option D is correct. You couldn't have assigned an `int` on line 8 either because `obj[3]` is really an `int[]` behind the scenes and not an `int`.
29. C. The code sorts before calling `binarySearch`, so it meets the precondition for that method. The target string of "RedHat" is not found in the sorted array. If it was found, it would be between the second and third element. The rule is to take the negative index of where it would be inserted and subtract 1. It would need to be inserted as the third element. Since indexes are zero based, this is index 2. We take the negative, which is -2, and subtract 1, giving -3. Therefore, Option C is correct.

- 30.** B. Array indexes begin with zero. `FirstName` is the name of the class, not an argument. Therefore, the first argument is `Wolfie`, and Option B is correct.
- 31.** C. The name of the program is `Count` and there are two arguments. Therefore, the program outputs 2, and Option C is correct.
- 32.** B. This class is called with two arguments. The first one (`seed`) is stored in the variable `one`. Then the array is sorted, meeting the precondition for binary search. Binary search returns 1 because `seed` is the second element in the sorted array, and Java uses zero-based indexes. Option B is correct.
- 33.** D. Options A and B show the braces can be before or after the variable name and produce the same array. Option C specifies the same array the long way with two arrays of length 1. Option D is the answer because it is different than the others. It instead specifies an array of length 1 where that element is of length 2.
- 34.** C. Arrays are indexed using numbers, not strings, making Options A and B incorrect. Since array indexes are zero based, Option C is the answer.
- 35.** D. In Java, arrays are indexed starting with 0. While it is unusual for the loop to start with 1, this does not cause an error. What does cause an error is the loop ending at `data.length`, because the `<=` operator is used instead of the `<` operator. The last loop index is 6, not 7. On the last iteration of the loop, the code throws an `ArrayIndexOutOfBoundsException`. Therefore, Option D is correct.
- 36.** C. Array indexes begin with zero. `FirstName` is the name of the class, not an argument. The first and only argument is `Wolfie`. There is not a second argument, so Option C is correct.
- 37.** D. This code is correct. Line `r1` correctly creates a 2D array. The next three lines correctly assign a value to an array element. Line `r3` correctly outputs 3 in a row!
- 38.** D. Arrays expose a `length` variable. They do not have a `length()` method. Therefore, the code does not compile, and Option D is correct.
- 39.** B. This one is tricky since the array braces are split up. This means that `bools` is a 3D array reference. The braces both before and after the variable name `count`. For `moreBools`, it is only a 2D array reference because there are only two pairs of braces next to the type. In other words, `boolean[][]` applies to both variables. Then `bools` gets another dimension from the braces right after the variable name. However, `moreBools` stays at 2D, making Option B correct.
- 40.** B. Since no arguments are passed from the command line, this creates an empty array. Sorting an empty array is valid and results in an empty array. Therefore, Option B is correct.
- 41.** D. Java requires having a sorted array before calling `binarySearch`. Since the array is not sorted, the result is undefined, and Option D is correct. It may happen that you get 1 as the result, but this behavior is not guaranteed. You need to know for the exam that this is undefined even if you happen to get the “right” answer.

42. B. Line 8 attempts to store a `String` in an array meant for an `int`. Line 8 does not compile, and Option B is correct.
43. A. This array has two elements, making `listing.length` output 2. While each array element does not have the same size, this does not matter because we are only looking at the first element. The first element has one. This makes the answer Option A.
44. C. `FirstName` is the name of the class, not an argument. There are no other arguments, so `names` is an empty array. Therefore, Option C is correct.
45. A. In Java, arrays are indexed starting with 0. While it is unusual for the loop to start with 1, this does not cause an error. It does cause the code to output six lines instead of seven since the loop doesn't cover the first array element. Therefore, Option A is correct.
46. B. The name of the program is `Count`, and there is only one argument because double quotes are used around the value. That argument is a `String` with three characters: 1, a space, and 2. Therefore, the program outputs 1, and Option B is correct.
47. A. Java requires having a sorted array before calling `binarySearch()`. You do not have to call `Arrays.sort` to perform the sort though. This array happens to already be sorted, so it meets the precondition. The target string of "Linux" is the first element in the array. Since Java uses zero-based indexing, the answer is Option A.
48. A. From within a method, an array parameter and a varargs parameter are treated the same. From the caller, an array parameter is more restrictive. Both types can receive an array. However, only a varargs parameter is allowed to automatically turn individual parameters into an array. Therefore, statement I is correct and the answer is Option A.
49. B. All of the variables except `nums2b` point to a 4D array. Don't create a 4D array; it's confusing. The options show the braces can be before or after the variable in any combination. Option B is the answer because `nums2b` points to a 3D array. It only has three pairs of braces before the variable and none after. By comparison, `nums2a` has three pairs of braces before the variable and the fourth pair of braces after.
50. C. Binary search returns an `int` representing the index of a match or where a match would be. An `int` cannot be stored in a `String` variable. Therefore, the code does not compile and the answer is Option C.

## Chapter 5: Using Loop Constructs

1. D. A `while` loop has a condition that returns a `boolean` that controls the loop. It appears at the beginning and is checked before entering the loop. Therefore, Option D is correct. A traditional `for` loop also has a `boolean` condition that is checked before entering the loop. However, it is best known for having a counter variable, making Option B incorrect. Option A is incorrect because the `boolean` condition on a `do-while` loop is at the end of the loop. Option C is incorrect because there is no condition as part of the loop construct.

2. B. A traditional for loop is best known for having a loop variable counting up or down as the loop progresses. Therefore, Option B is correct. Options A and D are incorrect because do-while and while loops are known for their boolean conditions. Option C is incorrect because the for-each loop iterates through without an index.
3. A. A do-while loop checks the loop condition after execution of the loop body. This ensures it always executes at least once, and Option A is correct. Option B is incorrect because there are loops you can write that do not ever enter the loop body, such as `for (int i=0;i<1;i++)`. Similarly, Option D is incorrect because a while loop can be written where the initial loop condition is false. Option C is incorrect because a for-each loop does not enter the loop body when iterating over an empty list.
4. C. While a traditional for loop often loops through an array, it uses an index to do so, making Option B incorrect. The for-each loop goes through each element, storing it in a variable. Option C is correct.
5. B. The continue keyword is used to end the loop iteration immediately and resume execution at the next iteration. Therefore, Option B is correct. Option A is incorrect because the break statement causes execution to proceed after the loop body. Options C and D are incorrect because these are not keywords in Java.
6. A. The break keyword is used to end the loop iteration immediately, skip any remaining executions of the loop, and resume execution immediately after the loop. Therefore, Option A is correct. Option B is incorrect because execution proceeds at the next execution of the current loop for continue. Options C and D are incorrect because these are not keywords in Java.
7. B. A traditional for loop is best known for having an initialization statement, condition statement, and update statement. Option B is correct.
8. C. With a traditional for loop, you control the order in which indexes are visited in code. This means you can loop through an array in ascending or descending order, and Option C is correct.
9. A. With a for-each loop, the loop order is determined for you. With an array, this means starting with index 0, and Option A is correct. A traditional for loop allows you to control the order and iterate in either order.
10. A. A do-while loop has a condition that returns a boolean at the end of the loop. Therefore, Option A is correct. Option D is incorrect because a while loop has this condition at the beginning of the loop. A traditional for loop is best known for having a loop variable, making Option B incorrect. Option C is incorrect because there is no condition as part of the loop construct.
11. B. A while loop requires a boolean condition. While `singer` is a variable, it is not a boolean. Therefore, the code does not compile, and Option B is correct.
12. B. This is a correct loop to go through an `ArrayList` or `List` starting from the end. It starts with the last index in the list and goes to the first index in the list. Option B is correct.
13. A. The first time through the loop, the index is 0 and `glass`, is output. The break statement then skips all remaining executions on the loop and the `main()` method ends. If there was no break keyword, this would be an infinite loop because there's no incrementor.

14. A. Immediately after `letters` is initialized, the loop condition is checked. The variable `letters` is of length 0, which is not equal to 2 so the loop is entered. In the loop body, `letters` becomes length 1 with contents "a". The loop index is checked again and now 1 is not equal to 2. The loop is entered and `letters` becomes length 2 and contains "aa". Then the loop index is checked again. Since the length is now 2, the loop is completed and aa is output. Option A is correct.
15. D. There are three arguments passed to the program. This means that `i` is 3 on the first iteration of the loop. The program prints `args`. Then `i` is incremented to 4. Which is also greater than or equal to 0. Since `i` never gets smaller, this code produces an infinite loop and the answer is Option D.
16. B. Since `count` is a class variable that isn't specifically initialized, it defaults to 0. On the first iteration of the loop, "Washington", is 11 characters and `count` is set to 1. The `if` statement's body is not run. The loop then proceeds to the next iteration. This time, the post-increment operator uses index 1 before setting `count` to 2. "Monroe" is checked, which is only 6 characters. The `break` statement sends the execution to after the loop and 2 is output. Option B is correct.
17. C. At first this code appears to be an infinite loop. However, the `count` variable is declared inside the loop. It is not in scope after the loop where it is referenced by the `println()`. Therefore, the code does not compile, and Option C is correct.
18. D. A `for` loop is allowed to have all three segments left blank. In fact, `for(;;) {}` is an infinite loop.
19. C. It is not possible to create an infinite loop using a `for-each` because it simply loops through an array or `ArrayList`. The other types allow infinite loops, such as, for example, `do {} while(true)`, `for(;;)` and `while(true)`. Therefore, Option C is correct. And yes, we know it is possible to create an infinite loop with `for-each` by creating your own custom `Iterable`. This isn't on the OCA or OCP exam though. If you think the answer is Option D, this is a great reminder of what not to read into on the real exam!
20. A. This is a correct loop to go through an `ArrayList` or `List` starting from the beginning. It starts with index 0 and goes to the last index in the list. Option A is correct.
21. D. Braces are optional around loops if there is only one statement. Parentheses are not allowed to surround a loop body though, so the code does not compile, and Option D is correct.
22. B. The `for-each` loop uses a variable and colon as the syntax, making Option B correct.
23. C. In this figure, we want to end the inner loop and resume execution at the `letters` label. This means we only want to break out of the inner loop. A `break` statement does just that. It ends the current loop and resumes execution immediately after the loop, making `break;` a correct answer. The `break numbers;` statement explicitly says which loop to end, which does the same thing, making it correct as well. By contrast, `break letters;` ends the outer loop, causing the code only to run the `println()` once. Therefore, two statements correctly match the diagram, and Option C is correct.

- 24.** B. In this figure, we want to end the inner loop and resume execution at the `letters` label. The `continue letters;` statement does that. The other two statements resume execution at the inner loop. Therefore, only the second statement correctly matches the diagram, and Option B is correct.
- 25.** C. A `while` loop checks the boolean condition before entering the loop. In this code, that condition is false, so the loop body is never run. No output is produced, and Option C is correct.
- 26.** C. A `for-each` loop is allowed to be used with arrays and `ArrayList` objects. `StringBuilder` is not an allowed type for this loop, so Option C is the answer.
- 27.** B. This is a correct `do-while` loop. On the first iteration of the loop, the `if` statement executes and prints `inflate-`. Then the loop condition is checked. The variable `balloonInflated` is `true`, so the loop condition is `false` and the loop completes.
- 28.** D. Immediately after `letters` is initialized, the loop condition is checked. The variable `letters` is of length 0, which is not equal to 3, so the loop is entered. In the loop body, `letters` becomes length 2 and contains "ab". The loop index is checked again and now 2 is not equal to 3. The loop is entered and `letters` becomes length 4 with contents "abab". Then the loop index is checked again. Since the length 4 is not equal to 3, the loop body is entered again. This repeats for 6, 8, 10, etc. The loop never ends, and Option D is correct.
- 29.** B. In a `for` loop, the segments are an initialization expression, a boolean conditional, and an update statement in that order. Therefore, Option B is correct.
- 30.** B. On the first iteration through the outer loop, `chars` becomes 1 element. The inner loop is run once and `count` becomes 9. On the second iteration through the outer loop, `chars` becomes 2 elements. The inner loop runs twice so `count` becomes 7. On the third iteration through the outer loop, `chars` becomes 3 elements. The inner loop runs three times so `count` becomes 4. On the fourth iteration through the outer loop, `chars` becomes 4 elements. The inner loop runs four times so `count` becomes 0. Then both loops end. Therefore, Option B is correct.
- 31.** A. On the first iteration of the outer loop, `i` starts out at 10. The inner loop sees that  $10 > 3$  and subtracts 3, making the 7 the new value of `i`. Since  $7 > 3$ , we subtract 3 again, making `i` set to 4. Yet again  $4 > 3$ , so `i` becomes 1. Then `k` is finally incremented to 1. The outer loop decrements `i` `i`, making it 0. The boolean condition sees that 0 is not greater than 0. The outer loop ends and 1 is printed out. Therefore, Option A is correct.
- 32.** D. Options A and C do not compile as they do not use the correct syntax for a `for-each` loop. The `for-each` loop is only able to go through an array in ascending order. It is not able to control the order, making Option C incorrect. Therefore, Option D is the answer.
- 33.** C. Since there are no brackets around the `for` statement, the loop body is only one line. The `break` statement is not in the loop. Since `break` cannot be used at the top level of a method, the code does not compile, and Option C is correct.
- 34.** C. Multiple update expressions are separated with a comma rather than a semicolon. Tricky, we know. But it is an important distinction. This makes Option C correct.

35. D. There are three arguments passed to the program. This means that `i` is 3 on the first iteration of the loop. The program attempts to print `args[3]`. Since indexes are zero based in Java, it throws an `ArrayIndexOutOfBoundsException`.
36. B. The first time the loop condition is checked, the variable `tie` is `null`. The loop body executes, setting `tie`. Despite the indentation, there are no brackets surrounding the loop body so the `print` does not run yet. Then the loop condition is checked and `tie` is not `null`. The `print` runs after the loop, printing out `shoelace` once, making Option B correct.
37. C. The code compiles as is. However, we aren't asked about whether the code compiles as is. Line 27 refers to a loop label. While the label is still present, it no longer points to a loop. This causes the code to not compile, and Option C is correct.
38. C. The `continue` statement is useless here since there is no code later in the loop to skip. The `continue` statement merely resumes execution at the next iteration of the loop, which is what would happen if the if-then statement was empty. Therefore, `count` increments for each element of the array. The code outputs 4, and Option C is correct.
39. C. A `do-while` loop requires a boolean condition. The `builder` variable is a `StringBuilder` and not a boolean. The code does not compile, and Option C is correct.
40. A. At first this code appears to be an infinite loop. However, there is a `break` statement. On line 6, `count` is set to 0. On line 9, it is changed to 1. Then the condition on line 10 runs. `count` is less than 2 so the inner loop continues. Then `count` is set to 2 on the next iteration of the inner loop. The loop condition on line 10 runs again and this time is false. The inner loop is completed. Then line 11 of the outer loop runs and sends execution to after the loop on line 13. At this point `count` is still 2, so Option A is correct.
41. C. Option A breaks out of the inner loop, but the outer loop is still infinite. Option B has the same problem. Option C is correct because it breaks out of both loops.
42. B. This code is correct. It initializes two variables and uses both variables in the condition check and the update statements. Since it checks the size of both arrays correctly, it prints the first two sets of elements, and Option B is correct.
43. B. Looping through the same list multiple times is allowed. The outer loop executes twice. The inner loop executes twice for each of those iterations of the outer loop. Therefore, the inner loop executes four times, and Option B is correct.
44. B. The initializer, which is `alpha`, runs first. Then Java checks the condition, which is `beta`, to see if loop execution should start. Since `beta` returns `false`, the loop is never entered, and Option B is correct.
45. B. The initializer, which is `alpha`, runs first. Then Java checks the condition, which is `beta`, to see if loop execution should start. Then the loop body, which is `delta`, runs. After the loop execution, the updater, which is `gamma`, runs. Then the loop condition, which is `beta`, is checked again. Therefore, Option B is correct.

- 46.** C. Option A goes through five indexes on the iterations: 0, 1, 2, 3 and 4. Option B also goes through five indexes: 1, 2, 3, 4 and 5. Option D goes through five iterations as well, from 0 to 4. However, Option C goes through six iterations since the loop condition is at the end of the loop. Therefore it is not like the others, and Option C is the answer.
- 47.** D. The first time the loop condition is checked, the variable `tie` is `null`. However, the loop body is empty due to the semicolon right after the condition. This means the loop condition keeps running with no opportunity for `tie` to be set. Therefore, this is an infinite loop, and Option D is correct.
- 48.** C. Remember to look for basic errors before wasting time tracking the flow. In this case, the label of the loop is trying to use the keyword `for`. This is not allowed, so the code does not compile. If the label was valid, Option A would be correct.
- 49.** D. On the first iteration of the loop, the `if` statement executes printing `inflate-`. Then the loop condition is checked. The variable `baloonInflated` is `true`, so the loop condition is true and the loop continues. The `if` statement no longer runs, but the variable never changes state again, so the loop doesn't end.
- 50.** B. In a `for` loop, the type is only allowed to be specified once. A comma separates multiple variables since they are part of the same statement. Therefore, Option B is correct.

## Chapter 6: Working with Methods and Encapsulation

1. C. The `protected` modifier allows access by subclasses and members within the same package, while the package-private modifier allows access only to members in the same package. Therefore, the `protected` access modifier allows access to everything the package-private access modifier, plus subclasses, making Option C the correct answer. Options A, B, and D are incorrect because the first term is a more restrictive access modifier than the second term.
2. B. The `super()` statement is used to call a constructor in a parent class, while the `this()` statement is used to call a constructor in the same class, making Option B correct and Option A incorrect. Options C and D are incorrect because they are not constructors.
3. D. The `sell()` method does not compile because it does not return a value if both of the if-then statements' conditional expressions evaluate to `false`. While logically, it is true that `price` is either less than `10` or greater than or equal to `10`, the compiler does not know that. It just knows that if both if-then statements evaluate to `false`, then it does not have a return value, therefore it does not compile.
4. D. The three overloaded versions of `nested()` compile without issue, since each method takes a different set of input arguments, making Options B and C incorrect. The code does not compile, though, due to the first line of the `main()` method, making Option A incorrect. The no-argument version of the `nested()` method does not return a value, and trying to output a `void` return type in the `print()` method throws an exception at runtime.

5. B. Java uses pass-by-value to copy primitives and references of objects into a method. That means changes to the primitive value or reference in the method are not carried to the calling method. That said, the data within an object can change, just not the original reference itself. Therefore, Option B is the correct answer, and Options C and D are incorrect. Option A is not a real term.
6. C. Option A is incorrect because the getter should return a value. Option B is incorrect because the setter should take a value. Option D is incorrect because the setter should start with `set` and should not return a value. Option C is a correct setter declaration because it takes a value, uses the `void` return type, and uses the correct naming convention.
7. B. Options A, C, and D are true statements about calling `this()` inside a constructor. Option B is incorrect because a constructor can only call `this()` or `super()` on the first line of the constructor, but never both in the same constructor. If both constructors were allowed to be called, there would be two separate calls to `super()`, leading to duplicate initialization of parent constructors, since the other constructor referenced by `this()` would also call `super()` (or be chained to one that eventually calls `super()`).
8. B. Option A is incorrect because the `public` access modifier starts with a lowercase letter. Options C and D are incorrect because the return types, `void` and `String`, are incompatible with the method body that returns an integer value of 10. Option B is correct and has package-private access. It also uses a return type of `Long` that the integer value of 10 can be easily assigned to without an explicit cast.
9. C. The only variables always available to all instances of the class are those declared `static`; therefore, Option C is the correct answer. Option A may seem correct, but `public` variables are only available if a reference to the object is maintained among all instances. Option B is incorrect because there is no `local` keyword in Java. Option D is also incorrect because a `private` instance variable is only accessible within the instance that created it.
10. A. First off, all of the lines compile but they produce various different results. Remember that the default initialization of a `boolean` instance variable is `false`, making `outside` `false` at line `p1`. Therefore, `this(4)` will cause `rope` to be set to 5, while `this(5)` will cause `rope` to be set to 6. Since 5 is the number we are looking for, Option A is correct, and Option C is incorrect. Option B is incorrect. While the statement does create a new instance of `Jump`, with `rope` having a value of 5, that instance is nested and the value of `rope` does not affect the surrounding instance of `Jump` that the constructor was called in. Option D is also incorrect. The value assigned to `rope` is 4, not the target 5.
11. B. Options A, C, and D are true statements. In particular, Option C allows us to write the `equals()` methods between two objects that compare private attributes of the class. Option D is true because `protected` access also provides package-private access. Option B is false. Package-private attributes are only visible if the two classes are in the same package, regardless of whether one extends the other.
12. D. The class data, `stuff`, is declared `public`, allowing any class to modify the `stuff` variable and making the implementation inherently unsafe for encapsulation. Therefore, there are no values that can be placed in the two blanks to ensure the class properly encapsulates its data, making Option D correct. Note that if `stuff` was declared `private`, Options A, B, and C would all be correct. Encapsulation does not require JavaBean syntax, just that the internal attributes are protected from outside access, which all of these sets of values do achieve.

- 13.** C. Option A is incorrect because Java only inserts a no-argument constructor if there are no other constructors in the class. Option B is incorrect because the parent can have a default no-argument constructor, which is inserted by the compiler and accessible in the child class. Finally, Option D is incorrect. A class that contains two no-argument constructors will not compile because they would have the same signature. Finally, Option C is correct. If a class extends a parent class that does not include a no-argument constructor, the default no-argument constructor cannot be automatically inserted into the child class by the compiler. Instead, the developer must explicitly declare at least one constructor and explicitly define how the call to the parent constructor is made.
- 14.** A. A method may contain at most one varargs parameter, and it must appear as the last argument in the list. For this reason, Option A is correct, and Options B, C, and D are incorrect.
- 15.** C. To solve this problem, it helps to remember that Java is a pass-by-value language in which copies of primitives and object references are sent to methods. This also means that an object's data can be modified within a method and shared with the caller, but not the reference to the object. Any changes to the object's reference within the method are not carried over to the caller. In the `slalom()` method, the `Ski` object is updated with an age value of 18. Although, the last line of the `slalom()` method changes the variable value to `null`, it does not affect the `mySkier` object or reference in the `main()` method. Therefore, the `mySkier` object is not `null` and the age variable is set to 18, making Options A and D incorrect. Next, the `name` variable is reassigned to the `Wendy` object, but this does not change the reference in the `main()` method, so `myName` remains `Rosie`. Finally, the `speed` array is assigned a new object and updated. Since the array is updated after the reference is reassigned, it does not affect the `mySpeed` array in the `main()` method. The result is that `mySpeed` continues to have a single element with the default `int` value of 0. For these reasons, Option B is incorrect, and Option C is correct.
- 16.** B. Options A and D would not allow the class to compile because two methods in the class cannot have the same name and arguments, but a different return value. Option C would allow the class to compile, but it is not a valid overloaded form of our `findAverage()` method since it uses a different method name. Option B is a valid overloaded version of the `findAverage()` method, since the name is the same but the argument list differs.
- 17.** D. Implementing encapsulation prevents internal attributes of a class from being modified directly, so Option C is a true statement. By preventing access to internal attributes, we can also maintain class data integrity between elements, making Option B a true statement. Option A is also a true statement about encapsulation, since well-encapsulated classes are often easier to use. Option D is an incorrect statement. Encapsulation makes no guarantees about performance and concurrency.
- 18.** A. Option B is incorrect because `String` values are immutable and cannot be modified. Options C and D are also incorrect since variables are passed by value, not reference, in Java. Option A is the correct answer. The contents of an array can be modified when passed to a method, since a copy of the reference to the object is passed. For example, the method can change the first element of a non-empty array.

- 19.** B. Option A is not a valid syntax in Java. Option C would be correct if there was a `static` import, but the question specifically says there are not any. Option D is almost correct, since it is a way to call the method, but the question asks for the best way to call the method. In that regard, Option B is the best way to call the method, since we are given that two classes are in the same package, therefore the package name would not be required.
- 20.** D. Options A and B are incorrect because a method with a non-void return type requires that the method return a value using the `return` statement. Option C is also incorrect since a method with a `void` return type can still call the `return` command with no values and exit the method. Therefore, Option D is the correct answer.
- 21.** C. The `finish()` method modifies two variables that are marked `final`, `score` and `result`. The `score` variable is modified by the post-increment `++` operator, while the `result` variable is modified by the compound addition `+=` operator. Removing both `final` modifiers allows the code to compile. For this reason, Option C is the correct answer.
- 22.** D. The `super()` statement is used to call a constructor in the parent class, while `super` is used to reference a member of the parent class. The `this()` statement is used to call a constructor in the current class, while `this` is used to reference a member of the current class. For these reasons, Option D is the correct answer.
- 23.** B. The method signature has package-private, or default, access; therefore, it is accessible to classes in the same package, making Option B the correct answer.
- 24.** A. The access modifier of `strength` is `protected`, meaning subclasses and classes within the same package can modify it. Changing the value to `private` would improve encapsulation by making the `Protect` class the only one capable of directly modifying it. For these reasons, the first statement is correct. Alternatively, the second and third statements do not improve the encapsulation of the class. While having getters and setters for `private` variables is helpful, they are not required. Encapsulation is about protecting the data elements. With this in mind, it is clear the `material` variable is already protected. Therefore, Option A is the correct answer.
- 25.** A. Option A is correct since method names may include the underscore `_` character as well as the dollar `$` symbol. Note that there is no rule that requires a method start with a lower-case character; it is just a practice adopted by the community. Option B is incorrect because the hyphen `-` character may not be part of a method name. Option C is incorrect since `new` is a reserved word in Java. Finally, Option D is incorrect. A method name must start with a letter, the dollar `$` symbol, or an underscore `_` character.
- 26.** D. The code does not compile, regardless of what is inserted into the line because the method signature is invalid. The return type, `int`, should go before the method name and after any access, `final`, or `static` modifiers. Therefore, Option D is the correct answer. If the method was fixed, by swapping the order of `int` and `static` in the method declaration, then Option C would be the correct answer. Options A and B are still incorrect, though, since each uses a return type that cannot be implicitly converted to `int`.

- 27.** B. Java uses pass-by-value, so changes made to primitive values and object references passed to a method are not reflected in the calling method. For this reason, Options A and C are incorrect statements. Option D is also an invalid statement because it is a special case of Option A. Finally, Option B is the correct answer. Changes to the data within an object are visible to the calling method since the object that the copied reference points to is the same.
- 28.** C. The code contains a compilation problem in regard to the `contents` instance variable. The `contents` instance variable is marked `final`, but there is a `setContents()` instance method that can change the value of the variable. Since these two are incompatible, the code does not compile, and Option C is correct. If the `final` modifier was removed from the `contents` variable declaration, then the expected output would be of the form shown in Option A.
- 29.** A. JavaBean methods use the prefixes `get`, `set`, and `is` for boolean values, making Option A the correct choice.
- 30.** C. Option A is incorrect because the keywords `static` and `import` are reversed. The `Closet` class uses the method `getClothes()` without a reference to the class name `Store`, therefore a `static` import is required. For this reason, Option B is incorrect since it is missing the `static` keyword. Option D is also incorrect since `static` imports are used with members of the class, not a class name. Finally, Option C is the correct answer since it properly imports the method into the class using a `static import`.
- 31.** D. In Java, the lack of an access modifier indicates that the member is package-private, therefore Option D is correct. Note that the `default` keyword is used for interfaces and `switch` statements, and is not an access modifier.
- 32.** B. The code does not compile, so Option A is incorrect. The class contains two constructors and one method. The first method, `Stars()`, looks a lot like a no-argument constructor, but since it has a return value of `void`, it is a method, not a constructor. Since only constructors can call `super()`, the code does not compile due to this line. The only constructor in this class, which takes an `int` value as input, performs a pointless assignment, assigning a variable to itself. While this assignment has no effect, it does not prevent the code from compiling. Finally, the `main()` method compiles without issue since we just inserted the full package name into the class constructor call. This is how a class that does not use an `import` statement could call the constructor. Since the method is in the same class, and therefore the same package, it is redundant to include the package name but not disallowed. Because only one line causes the class to fail to compile, Option B is correct.
- 33.** A. An instance method or constructor has access to all `static` variables, making Option A correct. On the other hand, `static` methods and `static` initializers cannot reference instance variables since they are defined across all instances, making Options B and C incorrect. Note that they can access instance variables if they are passed a reference to a specific instance, but not in the general case. Finally, Option D is incorrect because `static final` variables must be set when they are declared or in a `static initialization block`.

- 34.** B. The method `calculateDistance()` requires a return type that can be easily converted to a `short` value. Options A, C, and D are incorrect because they each use a larger data type that requires an explicit cast. Option D also does not compile because the `Short` constructor requires an explicit cast to convert the value of 4, which is assumed to be an `int`, to a `short`, as shown in `new Short((short)4)`. Option B is the correct answer since a `byte` value can be easily promoted to `short` and returned by the method.
- 35.** C. Overloaded methods have the same name but a different list of parameters, making the first and third statements true. The second statement is false, since overloaded methods can have the same or different return types. Therefore, Option C is the correct answer.
- 36.** C. The declaration of `monday` does not compile, because the value of a `static final` variable must be set when it is declared or in a `static` initialization block. The declaration of `tuesday` is fine and compiles without issue. The declaration of `wednesday` does not compile because there is no data type for the variable. Finally, the declaration of `thursday` does not compile because the `final` modifier cannot appear before the access modifier. For these reasons, Option C is the correct answer.
- 37.** D. The `Puppy` class does not declare a constructor, so the default no-argument constructor is automatically inserted by the compiler. What looks like a constructor in the class is actually a method that has a return type of `void`. Therefore, the line in the `main()` method to create the new `Puppy(2)` object does not compile, since there is no constructor capable of taking an `int` value, making Option D the correct answer.
- 38.** A. The `public` modifier allows access to members in the same class, package, subclass, or even classes in other packages, while the `private` modifier allows access only to members in the same class. Therefore, the `public` access modifier allows access to everything the `private` access modifier does, and more, making Option A the correct answer. Options B, C, and D are incorrect because the first term is a more restrictive access modifier than the second term.
- 39.** A. The code compiles without issue, so Option D is incorrect. The key here is that Java uses `pass by value` to send object references to methods. Since the `Phone` reference `p` was reassigned in the first line of the `sendHome()` method, any changes to the `p` reference were made to a new object. In other words, no changes in the `sendHome()` method affected the object that was passed in. Therefore, the value of `size` was the same before and after the method call, making the output 3 and Option A the correct answer.
- 40.** B. Options A and D are equivalent and would allow the code to compile. They both are proper ways to access a `static` method from within an instance method. Option B is the correct answer. The class would not compile because `this.Drink` has no meaning to the compiler. Finally, Option C would still allow the code to compile, even though it is considered a poor coding practice. While `static` members should be accessed in a `static` way, it is not required.
- 41.** C. The method signature requires one `int` value, followed by exactly one `String`, followed by `String` `varargs`, which can be an array of `String` values or zero or more individual `String` values. Only Option C conforms to these requirements, making it the correct answer.

- 42.** D. Option A is a statement about `final static` variables, not all `static` variables. Option B only applies to `static` variables marked `private`, not `final`. Option C is false because `static` imports can be used to reference both variables and methods. Option D is the correct answer because a `static` variable is accessible to all instances of the class.
- 43.** A. Option A is the correct answer because the first line of a constructor could be `this()` or `super()`, making it an untrue statement. Option B is a true statement because the compiler will insert the default no-argument constructor if one is not defined. Option C is also a true statement, since zero or more arguments may be passed to the parent constructor, if the parent class defines such constructors. Option D is also true. The value of a `final` instance variable should be set when it is declared, in an initialization block, or in a constructor.
- 44.** D. The last `static` initialization block accesses `height`, which is an instance variable, not a `static` variable. Therefore, the code will not compile no matter how many `final` modifiers are removed, making Option D the correct answer. Note that if the line `height = 4;` was removed, then no `final` modifiers would need to be removed to make the class compile.
- 45.** D. Since a constructor call is not the first line of the `RainForest()` constructor, the compiler inserts the no-argument `super()` call. Since the parent class, `Forest`, does not define a no-argument `super()` constructor, the `RainForest()` constructor does not compile, and Option D is correct.
- 46.** A. The code compiles without issue, so Option D is incorrect. In the `main()` method, the value 2 is first cast to a `byte`. It is then increased by one using the addition `+` operator. The addition `+` operator automatically promotes all `byte` and `short` values to `int`. Therefore, the value passed to the `choose()` in the `main()` method is an `int`. The `choose(int)` method is called, returning 5 and making Option A the correct answer. Note that without the addition operation in the `main()` method, `byte` would have been used as the parameter to the `choose()` method, causing the `choose(short)` to be selected as the next closest type and outputting 2, making Option B the correct answer.
- 47.** C. The variable `startTime` can be automatically converted to `Integer` by the compiler, but `Integer` is not a subclass of `Long`. Therefore, the code does not compile due the wrong variable type being passed to the `getScore()` method on line `m2`, and Option C is correct.
- 48.** A. Java methods must start with a letter, the dollar `$` symbol, or underscore `_` character. For these reasons, Options B and D are incorrect, and Option A is correct. Option C is incorrect. The hashtag `(#)` symbol cannot be included in a method name.
- 49.** B. The `protected` modifier allows access by any subclass or class that is in the same package, therefore Option B is the correct answer.
- 50.** D. A `static` import is used to import `static` members of another class. In this case, the `withdrawal()` and `deposit()` methods in the `Bank` class are not marked `static`. They require an instance of `Bank` to be used and cannot be imported as `static` methods. Therefore, Option D is correct. If the two methods in the `Bank` class were marked `static`, then Option A would be the correct answer since wildcards can be used with `static` imports to import more than one method. Option B reverses the keywords `static` and `import`, while Option C incorrectly imports a class, which cannot be imported via a `static` import.

# Chapter 7: Working with Inheritance

1. C. The code does not compile, so Option A is incorrect. This code does not compile for two reasons. First, the name variable is marked `private` in the `Cinema` class, which means it cannot be accessed directly in the `Movie` class. Next, the `Movie` class defines a constructor that is missing an explicit `super()` statement. Since `Cinema` does not include a no-argument constructor, the no-argument `super()` cannot be inserted automatically by the compiler without a compilation error. For these two reasons, the code does not compile, and Option C is the correct answer.
2. D. All abstract interface methods are implicitly `public`, making Option D the correct answer. Option A is incorrect because `protected` conflicts with the implicit `public` modifier. Since `static` methods must have a body and abstract methods cannot have a body, Option B is incorrect. Finally, Option C is incorrect. A method, whether it be in an interface or a class, cannot be declared both `final` and `abstract`, as doing so would prevent it from ever being implemented.
3. C. A class cannot contain two methods with the same method signature, even if one is `static` and the other is not. Therefore, the code does not compile because the two declarations of `playMusic()` conflict with one another, making Option C the correct answer.
4. A. Inheritance is often about improving code reusability, by allowing subclasses to inherit commonly used attributes and methods from parent classes, making Option A the correct answer. Option B is incorrect. Inheritance can lead to either simpler or more complex code, depending on how well it is structured. Option C is also incorrect. While all objects inherit methods from `java.lang.Object`, this does not apply to primitives. Finally, Option D is incorrect because methods that reference themselves are not a facet of inheritance.
5. A. Recall that `this` refers to an instance of the current class. Therefore, any superclass of `Canine` can be used as a return type of the method, including `Canine` itself, making Option C an incorrect answer. Option B is also incorrect because `Canine` implements the `Pet` interface. An instance of a class can be assigned to any interface reference that it inherits. Option D is incorrect because `Object` is the superclass of instances in Java. Finally, Option A is the correct answer. `Canine` cannot be returned as an instance of `Class` because it does not inherit `Class`.
6. B. The key here is understanding which of these features of Java allow one developer to build their application around another developer's code, even if that code is not ready yet. For this problem, an interface is the best choice. If the two teams agree on a common interface, one developer can write code that uses the interface, while another developer writes code that implements the interface. Assuming neither team changes the interface, the code can be easily integrated once both teams are done. For these reasons, Option B is the correct answer.
7. B. The `drive()` method in the `Car` class does not override the version in the `Automobile` class since the method is not visible to the `Car` class. Therefore, the `final` attribute in the `Automobile` class does not prevent the `Car` class from implementing a method with the same signature. The `drive()` method in the `ElectricCar` class is a valid override of the method in the `Car` class, with the access modifier expanding in the subclass. For these reasons, the code compiles, and Option D is incorrect. In the `main()` method, the object created is an `ElectricCar`, even if it is assigned to a `Car` reference. Due to polymorphism, the method from the `ElectricCar` will be invoked, making Option B the correct answer.

8. D. While Java does not allow a class to extend more than one class, it does allow a class to implement any number of interfaces. Multiple inheritance is, therefore, only allowed via interfaces, making Option D the correct answer.
9. C. There are three problems with this method override. First, the `watch()` method is marked `final` in the `Television` class. The `final` modifier would have to be removed from the method definition in the `Television` class in order for the method to compile in the `LCD` class. Second, the return types `void` and `Object` are not covariant. One of them would have to be changed for the override to be compatible. Finally, the access modifier in the child class must be the same or broader than in the parent class. Since `package-private` is narrower than `protected`, the code will not compile. For these reasons, Option C is the correct answer.
10. C. First off, the return types of an overridden method must be covariant. Next, it is true that the access modifier must be the same or broader in the child method. Using a narrower access modifier in the child class would not allow the code to compile. Overridden methods must not throw any new or broader checked exceptions than the method in the superclass. For these reasons, Options A, B, and D are true statements. Option C is the false statement. An overridden method is not required to throw a checked exception defined in the parent class.
11. C. The `process()` method is declared `final` in the `Computer` class. The `Laptop` class then attempts to override this method, resulting in a compilation error, making Option C the correct answer.
12. A. The code compiles without issue, so Option D is incorrect. The rule for overriding a method with exceptions is that the subclass cannot throw any new or broader checked exceptions. Since `FileNotFoundException` is a subclass of `IOException`, it is considered a narrower exception, and therefore the overridden method is allowed. Due to polymorphism, the overridden version of the method in `HighSchool` is used, regardless of the reference type, and 2 is printed, making Option A the correct answer. Note that the version of the method that takes the varargs is not used in this application.
13. B. Interface methods are implicitly `public`, making Option A and C incorrect. Interface methods can also not be declared `final`, whether they are `static`, `default`, or `abstract` methods, making Option D incorrect. Option B is the correct answer because an interface method can be declared `static`.
14. C. Having one class implement two interfaces that both define the same `default` method signature leads to a compiler error, unless the class overrides the `default` method. In this case, the `Sprint` class does override the `walk()` method correctly, therefore the code compiles without issue, and Option C is correct.
15. B. Interfaces can extend other interfaces, making Option A incorrect. On the other hand, an interface cannot implement another interface, making Option B the correct answer. A class can implement any number of interfaces, making Option C incorrect. Finally, a class can extend another class, making Option D incorrect.
16. D. The code does not compile because `super.height` is not visible in the `Rocket` class, making Option D the correct answer. Even though the `Rocket` class defines a `height` value, the `super` keyword looks for an inherited version. Since there are none, the code does not compile. Note that `super.getWeight()` returns 3 from the variable in the parent class, as polymorphism and overriding does not apply to instance variables.

17. D. An abstract class can contain both abstract and concrete methods, while an interface can only contain abstract methods. With Java 8, interfaces can now have `static` and `default` methods, but the question specifically excludes them, making Option D the correct answer. Note that concrete classes cannot contain any abstract methods.
18. C. The code does not compile, so Option D is incorrect. The `IsoscelesRightTriangle` class is abstract; therefore, it cannot be instantiated on line `g3`. Only concrete classes can be instantiated, so the code does not compile, and Option C is the correct answer. The rest of the lines of code compile without issue. A concrete class can extend an abstract class, and an abstract class can extend a concrete class. Also, note that the override of `getDescription()` has a widening access modifier, which is fine per the rules of overriding methods.
19. D. The `play()` method is overridden in `Saxophone` for both `Horn` and `Woodwind`, so the return type must be covariant with both. Unfortunately, the inherited methods must also be compatible with each other. Since `Integer` is not a subclass of `Short`, and vice versa, there is no subclass that can be used to fill in the blank that would allow the code to compile. In other words, the `Saxophone` class cannot compile regardless of its implementation of `play()`, making Option D the correct answer.
20. C. A class can implement an interface, not extend it. Alternatively, a class extends an abstract class. Therefore, Option C is the correct answer.
21. A. The code compiles and runs without issue, making Options C and D incorrect. Although `super.material` and `this.material` are poor choices in accessing `static` variables, they are permitted. Since `super` is used to access the variable in `getMaterial()`, the value `papyrus` is returned, making Option A the correct answer. Also, note that the constructor `Book(String)` is not used in the `Encyclopedia` class.
22. B. Options A and C are both true statements. Either the `unknownBunny` reference variable is the same as the object type or it can be explicitly cast to the `Bunny` object type, therefore giving it access to all its members. This is the key distinction between reference types and object types. Assigning a new reference does not change the underlying object. Option D is also a true statement since any superclass that `Bunny` extends or interface it implements could be used as the data type for `unknownBunny`. Option B is the false statement and the correct answer. An object can be assigned to a reference variable type that it inherits, such as `Object unknownBunny = new Bunny();`.
23. D. An abstract method cannot include the `final` or `private` method. If a class contained either of these modifiers, then no concrete subclass would ever be able to override them with an implementation. For these reasons, Options A and B are incorrect. Option C is also incorrect because the `default` keyword applies to concrete interface methods, not abstract methods. Finally, Option D is correct. The `protected`, `package-private`, and `public` access modifiers can each be applied to abstract methods.
24. D. The declaration of `Sphere` compiles without issue, so Option C is incorrect. The `Mars` class declaration is invalid because `Mars` cannot extend `Sphere`, an interface, nor can `Mars` implement `Planet`, a class. In other words, they are reversed. Since the code does not compile, Option D is the correct answer. Note that if `Sphere` and `Planet` were swapped in the `Mars` class definition, the code would compile and the output would be `Mars`, making Option A the correct answer.

- 25.** B. A reference to a class can be implicitly assigned to a superclass reference without an explicit class, making Option B the correct answer. Assigning a reference to a subclass, though, requires an explicit cast, making Option A incorrect. Option C is also incorrect because an interface does not inherit from a class. A reference to an interface requires an explicit cast to be assigned to a reference of any class, even one that implements the interface. An interface reference requires an explicit cast to be assigned to a class reference. Finally, Option D is incorrect. An explicit cast is not required to assign a reference to a class that implements an interface to a reference of the interface.
- 26.** B. Interface variables are implicitly `public`, `static`, and `final`. Variables cannot be declared as `abstract` in interfaces, nor in classes.
- 27.** C. The class is loaded first, with the `static` initialization block called and 1 is outputted first. When the `BlueCar` is created in the `main()` method, the superclass initialization happens first. The instance initialization blocks are executed before the constructor, so 32 is outputted next. Finally, the class is loaded with the instance initialization blocks again being called before the constructor, outputting 45. The result is that 13245 is printed, making Option C the correct answer.
- 28.** C. Overloaded methods share the same name but a different list of parameters and an optionally different return type, while overridden methods share the exact same name, list of parameters, and return type. For both of these, the one commonality is that they share the same method name, making Option C the correct answer.
- 29.** A. Although the casting is a bit much, the object in question is a `SoccerBall`. Since `SoccerBall` extends `Ball` and implements `Equipment`, it can be explicitly cast to any of those types, so no compilation error occurs. At runtime, the object is passed around and, due to polymorphism, can be read using any of those references since the underlying object is a `SoccerBall`. In other words, casting it to a different reference variable does not modify the object or cause it to lose its underlying `SoccerBall` information. Therefore, the code compiles without issue, and Option A is correct.
- 30.** C. Both of these descriptions refer to variable and `static` method hiding, respectively, making Option C correct. Only instance methods can be overridden, making Options A and B incorrect. Option D is also incorrect because *replacing* is not a real term in this context.
- 31.** B. The code does not compile, so Option D is incorrect. The issue here is that the override of `getEqualSides()` in `Square` is invalid. A `static` method cannot override a non-`static` method and vice versa. For this reason, Option B is the correct answer.
- 32.** C. The application does not compile, but not for any reason having to do with the cast in the `main()` method. The `Rotorcraft` class includes an abstract method, but the class itself is not marked `abstract`. Only interfaces and abstract classes can include abstract methods. Since the code does not compile, Option C is the correct answer.
- 33.** B. A class can trivially be assigned to a superclass reference variable but requires an explicit cast to be assigned to a subclass reference variable. For these reasons, Option B is correct.

34. C. A concrete class is the first non-abstract subclass that extends an abstract class and implements any inherited interfaces. It is required to implement all inherited abstract methods, making Option C the correct answer.
35. D. First of all, interfaces can only contain abstract, final, and default methods. The method `fly()` defined in `CanFly` is not marked `static` or `default` and defines an implementation, an empty {}, meaning it cannot be assumed to be abstract; therefore, the code does not compile. Next, the implementation of `fly(int speed)` in the `Bird` class also does not compile, but not because of the signature. The method body fails to return an `int` value. Since it is an overloaded method, if it returned a value it would compile without issue. Finally, the `Eagle` class does not compile because it extends the `Bird` class, which is marked `final` and therefore, cannot be extended. For these three reasons, Option D is the correct answer.
36. B. Abstract classes and interfaces can both contain static and abstract methods as well as static variables, but only an interface can contain default methods. Therefore, Option B is correct.
37. C. Java does not allow multiple inheritance, so having one class extend two interfaces that both define the same default method signature leads to a compiler error, unless the class overrides the method. In this case, though, the `talk(String...)` method defined in the `Performance` class is not an overridden version of method defined in the interfaces because the signatures do not match. Therefore, the `Performance` class does not compile since the class inherits two default methods with the same signature and no overridden version, making Option C the correct answer.
38. A. In Java, only non-static, non-final, and non-private methods are considered virtual and capable of being overridden in a subclass. For this reason, Option A is the correct answer.
39. B. An interface can only extend another interface, while a class can only extend another class. A class can also implement an interface, although that comparison is not part of the question text. Therefore, Option B is the correct answer.
40. A. The code compiles without issue, so Option D is incorrect. Java allows methods to be overridden, but not variables. Therefore, marking them `final` does not prevent them from being reimplemented in a subclass. Furthermore, polymorphism does not apply in the same way it would to methods as it does to variables. In particular, the reference type determines the version of the `secret` variable that is selected, making the output 2 and Option A the correct answer.
41. D. Options A and C are incorrect because an overridden method cannot reduce the visibility of the inherited method. Option B is incorrect because an overridden method cannot declare a broader checked exception than the inherited method. Finally, Option D is the correct answer. The removal of the checked exception, the application of a broader access modifier, and the addition of the `final` attribute are allowed for overridden methods.

- 42.** C. The `setAnimal()` method requires an object that is `Dog` or a subclass of `Dog`. Since `Husky` extends `Dog`, Options A and B both allow the code to compile. Option D is also valid because a `null` value does not have a type and can be assigned to any reference variable. Option C is the only value that prevents the code from compiling because `Wolf` is not a subclass of `Dog`. Even though `Wolf` can be assigned to the instance `Canine` variable, the setter requires a compatible parameter.
- 43.** A. An interface method can be abstract and not have a body, or it can be `default` or `static` and have a body. An interface method cannot be `final` though, making Option A the correct answer.
- 44.** A. It looks like `getSpace()` in the `Room` class is an invalid override of the version in the `House` class since `package-private` is a more restrictive access modifier than `protected`, but the parameter list changes; therefore, this is an overloaded method, not an overridden one. Furthermore, the `Ballroom` class is abstract so no object is instantiated, but there is no requirement that an abstract class cannot contain a runnable `main()` method. For these reasons, the code compiles and runs without issue, making Option A correct.
- 45.** D. Trick question! Option A seems like the correct answer, but the second part of the sentence is false, regardless of whether you insert *overloaded* or *overridden*. Overridden methods must have covariant return types, which may not be exactly the same as the type in the parent class. Therefore, Option D is the correct answer.
- 46.** B. If a parent class does not include a no-argument constructor, a child class can still explicitly declare one; it just has to call an appropriate parent constructor with `super()`, making Option A incorrect. If a parent class does not include a no-argument constructor, the child class must explicitly declare a constructor, since the compiler will not be able to insert the default no-argument constructor, making Option B correct. Option C is incorrect because a parent class can have a no-argument constructor, while its subclasses do not. If Option C was true, then all classes would be required to have no-argument constructors since they all extend `java.lang.Object`, which has a no-argument constructor. Option D is also incorrect. The default no-argument constructor can be inserted into any class that directly extends a class that has a no-argument constructor. Therefore, no constructors in the subclass are required.
- 47.** D. The object type relates to the attributes of the object that exist in memory, while the reference type dictates how the object is able to be used by the caller. For these reasons, Option D is correct.
- 48.** A. The `play()` method is overridden in `Violin` for both `MusicCreator` and `StringInstrument`, so the return type must be covariant with both. `Long` is a subclass of `Number`, and therefore, it is covariant with the version in `MusicCreator`. Since it matches the type in `StringInstrument`, it can be inserted into the blank and the code would compile. While `Integer` is a subclass of `Number`, meaning the override for `MusicCreator` is valid, it is not a subclass of `Long` used in `StringInstrument`. Therefore, using `Integer` would cause the code to not compile. Finally, `Number` is compatible with the version of the method in `MusicCreator` but not with the version in `StringInstrument`, because `Number` is a superclass of `Long`, not a subclass. For these reasons, `Long` is the only class that allows the code to compile, making Option A the correct answer.

- 49.** B. The primary motivation for adding default interface methods to Java was for backward compatibility. These methods allow developers to update older classes with a newer version of an interface without breaking functionality in the existing classes, making Option B the correct answer. Option A is nonsensical and not the correct answer. Options C and D sound plausible, but both could be accomplished with static interface methods alone.
- 50.** C. The rule for overriding a method with exceptions is that the subclass cannot throw any new or broader checked exceptions. Since `IOException` is a superclass of `EOFException`, from the question description, we see that this is a broader exception and therefore not compatible. For this reason, the code does not compile, and Option C is the correct answer.

## Chapter 8: Handling Exceptions

1. D. A try block must include either a catch or finally block, or both. The `think()` method declares a try block but neither additional block. For this reason, the code does not compile, and Option D is the correct answer. The rest of the lines compile without issue, including `k1`.
2. B. The correct order of blocks is try, catch, and finally, making Option B the correct answer.
3. D. Option D is the correct model. The class `RuntimeException` extends `Exception`, and both `Exception` and `Error` extend `Throwable`. Finally, like all Java classes, they all inherit from `Object`. Notice that `Error` does not extend `Exception`, even though we often refer to these generally as exceptions.
4. A. While `Exception` and `RuntimeException` are commonly caught in Java applications, it is recommended `Error` not be caught. An `Error` often indicates a failure of the JVM which cannot be recovered from. For this reason, Option A is correct, and Options C and D are incorrect. Option B is not a class defined in the Java API; therefore, it is also incorrect.
5. D. The application does not compile because `score` is defined only within the try block. The other three places it is referenced, in the catch block, in the finally block, and outside the try-catch-finally block at the end, are not in scope for this variable and each does not compile. Therefore, the correct answer is Option D.
6. B. `ClassCastException`, `ArrayIndexOutOfBoundsException`, and `IllegalArgumentException` are unchecked exceptions and can be thrown at any time. `IOException` is a checked exception that must be handled or declared when used, making Option B the correct answer.
7. A. The `throws` keyword is used in method declarations, while the `throw` keyword is used to throw an exception to the surrounding process, making Option A the correct answer. The `catch` keyword is used to handle exceptions, not to create them or in the declaration of a method.

8. B. `IOException` is a subclass of `Exception`, so it must appear first in any related catch blocks. If `Exception` was to appear before `IOException`, then the `IOException` block would be considered unreachable code because any thrown `IOException` is already handled by the `Exception` catch block. For this reason, Option B is correct.
9. C. The application first enters the `try` block and outputs A. It then throws a `RuntimeException`, but the exception is not caught by the catch block since `RuntimeException` is not a subclass of `ArrayIndexOutOfBoundsException` (it is a superclass). Next, the `finally` block is called and C is output. Finally, the `RuntimeException` is thrown by the `main()` method and a stack trace is printed. For these reasons, Option C is correct.
10. C. The application does not compile, so Option D is incorrect. The `openDrawbridge()` method compiles without issue, so Options A and B are incorrect. The issue here is how the `openDrawbridge()` method is called from within the `main()` method on line p3. The `openDrawbridge()` method declares the checked exception, `Exception`, but the `main()` method from which it is called does not handle or declare the exception. In order for this code to compile, the `main()` method would have to have a try-catch statement around line p3 that properly handles the checked exception, or the `main()` would have to be updated to declare a compatible checked exception. For these reasons, line p3 does not compile, and Option C is the correct answer.
11. B. `NullPointerException` and `ArithmaticException` both extend `RuntimeException`, which are unchecked exceptions and not required to be handled or declared in the method in which they are thrown. On the other hand, `Exception` is a checked exception and must be handled or declared by the method in which it is thrown. Therefore, Option B is the correct answer.
12. A. The code compiles and runs without issues, so Options C and D are incorrect. The `try` block throws a `ClassCastException`. Since `ClassCastException` is not a subclass of `ArrayIndexOutOfBoundsException`, the first catch block is skipped. For the second catch block, `ClassCastException` is a subclass of `Throwable`, so that block is executed. Afterward, the `finally` block is executed and then control returns to the `main()` method with no exception being thrown. The result is that 1345 is printed, making Option A the correct answer.
13. C. A `finally` block can throw an exception, in which case not every line of the `finally` block would be executed. For this reason, Options A and D are incorrect. Option B is also incorrect. The `finally` block is called regardless of whether or not the related catch block is executed. Option C is the correct answer. Unlike an if-then statement, which can take a single statement, a `finally` statement requires brackets `{}`.
14. C. The code does not compile because the catch blocks are used in the wrong order. Since `IOException` is a superclass of `FileNotFoundException`, the `FileNotFoundException` is considered unreachable code. For this reason, the code does not compile, and Option C is correct.
15. C. A `try` statement requires a catch or a `finally` block. Without one of them, the code will not compile; therefore, Option D is incorrect. A `try` statement can also be used with both a catch and `finally` block, making Option C the correct answer. Note that `finalize` is not a keyword, but a method inherited from `java.lang.Object`.

- 16.** B. Option A is a true statement about exceptions and when they are often applied. Option B is the false statement and the correct answer. An application that throws an exception can choose to handle the exception and avoid termination. Option C is also a true statement. For example, a `NullPointerException` can be avoided on a `null` object by testing whether or not the object is `null` before attempting to use it. Option D is also a correct statement. Attempting to recover from unexpected problems is an important aspect of proper exception handling.
- 17.** D. The code does not compile because the `catch` block uses parentheses `()` instead of brackets `{}`, making Option D the correct answer. Note that `Boat` does not extend `Transport`, so while the override on line `j1` appears to be invalid since `Exception` is a broader checked exception than `CapsizedException`, that code compiles without issue. If the `catch` block was fixed, the code would output 4, making Option A the correct answer.
- 18.** B. Overridden methods cannot throw new or broader checked exceptions than the one they inherit. Since `Exception` is a broader checked exception than `PrintException`, Option B is not allowed and is the correct choice. Alternatively, declaring narrower or the same checked exceptions or removing them entirely is allowed, making Options A and C incorrect. Since Option B is correct, Option D is incorrect.
- 19.** D. All three of those classes belong to the `java.lang` package, so Option C seems like the correct answer. The Java compiler, though, includes `java.lang` by default, so no `import` statement is actually required to use those three classes, making Option D the correct answer.
- 20.** C. The code does not compile because the `catch` block is missing a variable type and name, such as `catch (Exception e)`. Therefore, Option C is the correct answer. Both implementations of `getSymbol()` compile without issue, including the overridden method. A subclass can swallow a checked exception for a method declared in a parent class; it just cannot declare any new or broader checked exceptions.
- 21.** B. Checked exceptions must be handled or declared or the program will not compile, while unchecked exceptions can be optionally handled. On the other hand, `java.lang.Error` should never be handled by the application because it often indicates an unrecoverable state in the JVM, such as running out of memory. For these reasons, Option B is the correct answer.
- 22.** B. The application does not compile, so Option D is incorrect. The checked `KnightAttackingException` thrown in the `try` block is handled by the associated `catch` block. The `ClassCastException` is an unchecked exception, so it is not required to be handled or declared and line `q1` compiles without issue. The `finally` block throws a checked `CastleUnderSiegeException`, which is required to be handled or declared by the method, but is not. There is no `try-catch` around line `q2`, and the method does not declare a compatible checked exception, only an unchecked exception. For this reason, line `q2` does not compile, and Option B is the correct answer. Lastly, line `q3` compiles without issue because the unchecked `RuntimeException` is not required to be handled or declared by the call in the `main()` method.

- 23.** A. If an exception matches multiple catch blocks, the first one that it encounters will be the only one executed, making Option A correct, and Options B and C incorrect. Option D is also incorrect. It is possible to write two consecutive catch blocks that can catch the same exception, with the first type being a subclass of the second. In this scenario, an exception thrown of the first type would match both catch blocks, but only the first catch block would be executed, since it is the more specific match.
- 24.** C. The code does not compile due to the call to `compute()` in the `main()` method. Even though the `compute()` method only throws an unchecked exception, its method declaration includes the `Exception` class, which is a checked exception. For this reason, the checked exception must be handled or declared in the `main()` method in which it is called. While there is a try-catch block in the `main()` method, it is only for the unchecked `NullPointerException`. Since `Exception` is not a subclass of `NullPointerException`, the checked `Exception` is not properly handled or declared and the code does not compile, making Option C the correct answer.
- 25.** D. A `NullPointerException` can be thrown if the value of `list` is `null`. Likewise, an `ArrayIndexOutOfBoundsException` can be thrown if the value of `list` is an array with fewer than 10 elements. Finally, a `ClassCastException` can be thrown if `list` is assigned an object that is not of type `Boolean[]`. For example, the assignment `list = (Boolean[]) new Object()` will compile without issue but throws a `ClassCastException` at runtime. Therefore, the first three options are possible, making Option D the correct answer.
- 26.** B. A `StackOverflowError` occurs when a program recurses too deeply into an infinite loop. It is considered an error because the JVM often runs out of memory and cannot recover. A `NullPointerException` occurs when an instance method or variable on a `null` reference is used. For these reasons, Option B is correct. A `NoClassDefFoundError` occurs when code available at compile time is not available at runtime. A `ClassCastException` occurs when an object is cast to an incompatible reference type. Finally, an `IllegalArgumentException` occurs when invalid parameters are sent to a method.
- 27.** C. Checked exceptions are commonly used to force a caller to deal with an expected type of problem, such as the inability to write a file to the file system. Without dealing with all checked exceptions thrown by the method, the calling code does not compile, so Option A is a true statement. Option B is also a true statement. Declaring various different exceptions informs the caller of the potential types of problems the method can encounter. Option C is the correct answer. There may be no recourse in handling an exception other than to terminate the application. Finally, Option D is also a true statement because it gives the caller a chance to recover from an exception, such as writing file data to a backup location.
- 28.** D. This code does not compile because the `catch` and `finally` blocks are in the wrong order, making Option D the correct answer. If the order was flipped, the output would be `Finished! Joyce Hopper`, making Option B correct.
- 29.** A. A `try` statement is not required to have a `finally` block, but if it does, there can be at most one. Furthermore, a `try` statement can have any number of `catch` blocks or none at all. For these reasons, Option A is the correct answer.

30. D. The code compiles without issue, so Option C is incorrect. The key here is noticing that `count`, an instance variable, is initialized with a value of 0. The `getDuckies()` method ends up computing  $5/0$ , which leads to an unchecked `ArithmeticException` at runtime, making Option D the correct answer.
31. B. If both the `catch` and `finally` blocks throw an exception, the one from the `finally` block is propagated to the caller, with the one from the `catch` block being dropped, making Option B the correct answer. Note that Option C is incorrect due to the fact that only one exception can be thrown to the caller.
32. A. The application does not compile because the `roar()` method in the `BigCat` class uses `throw` instead of `throws`, making Option A the correct answer. Note that if the `correct` keyword was used, the code would compile without issues, and Option D would be correct. Also the override of `roar()` in the `Lion` class is valid, since the overridden method has a broader access modifier and does not declare any new or broader checked exceptions.
33. A. Although this code uses the `RuntimeException` and `Exception` classes, the question is about casting. `Exception` is not a subclass of `RuntimeException`, so the assignment on the second line throws a `ClassCastException` at runtime, making Option A correct.
34. C. All exceptions in Java inherit from `Throwable`, making Option C the correct answer. Note that `Error` and `Exception` extend `Throwable`, and `RuntimeException` extends `Exception`.
35. B. If both values are valid non-null `String` objects, then no exception will be thrown, with the statement in the `finally` block being executed first, before returning control to the `main()` method; therefore, the second statement is a possible output. If either value is `null`, then the `toString()` method will cause a `NullPointerException` to be thrown. In both cases, the `finally` block will execute first, printing `Posted:`, even if there is an exception. For this reason, the first statement is not a possible output, and Option B is correct.
36. A. `ClassCastException` is a subclass of `RuntimeException`, so it must appear first in any related catch blocks. If `RuntimeException` was to appear before `ClassCastException`, then the `ClassCastException` block would be considered unreachable code, since any thrown `ClassCastException` is already handled by the `RuntimeException` catch block. For this reason, Option A is correct.
37. C. Option A is incorrect. You should probably seek help if the computer is on fire! Option B is incorrect because code that does not compile cannot run and therefore cannot throw any exceptions. Option C is the best answer, since an `IllegalArgumentException` can be used to alert a caller of missing or invalid data. Option D is incorrect; finishing sooner is rarely considered a problem.
38. C. The code does not compile due to an invalid override of the `operate()` method. An overridden method must not throw any new or broader checked exceptions than the method it inherits. Even though `RuntimeException` is a subclass of `Exception`, `Exception` is considered a new checked exception, since `RuntimeException` is an unchecked exception. Therefore, the code does not compile, and Option C is correct.

- 39.** D. A `NullPointerException` is an unchecked exception. While it can be handled by the surrounding method, either through a try-catch block or included in the method declaration, these are optional. For this reason, Option D is correct.
- 40.** D. In this application, the `throw RuntimeException(String)` statement in the `zipper()` method does not include the `new` keyword. The `new` keyword is required to create the object being thrown, since `RuntimeException(String)` is a constructor. For this reason, the code does not compile, and Option D is correct. If the keyword `new` was inserted properly, then the try block would throw a `CastClassException`, which would be replaced with a `RuntimeException` to the calling method by the catch block. The catch block in the `main()` method would then be activated, and no output would be printed, making Option C correct.
- 41.** C. For this question, notice that all the exceptions thrown or caught are unchecked exceptions. First, the `ClassCastException` is thrown in the try block and caught by the second catch block since it inherits from `RuntimeException`, not `IllegalArgumentException`. Next, a `NullPointerException` is thrown, but before it can be returned the finally block is executed and a `RuntimeException` replaces it. The application exits and the caller sees the `RuntimeException` in the stack trace, making Option C the correct answer. If the finally block did not throw any exceptions, then Option B would be the correct answer.
- 42.** D. Trick question! Options A, B, and C are each invalid overrides of the method because the return type must be covariant with `void`. For this reason, Option D is the correct answer. If the return types were changed to be `void`, then Option A would be a valid override. Options B and C would still be incorrect, since overridden methods cannot throw broader checked exceptions than the inherited method.
- 43.** D. The code does not compile because the catch block is missing a variable name, such as `catch (Error e)`. Therefore, Option D is the correct answer. If a variable name was added, the application would produce a stack trace at runtime and Option C would be the correct answer. Because `IllegalArgumentException` does not inherit from `Error`, the catch block would be skipped and the exception sent to the `main()` method at runtime. Note that the declaration of `RuntimeException` by both methods is unnecessary since it is unchecked, although allowed by the compiler.
- 44.** D. The `openDrawbridge()` is capable of throwing a variety of exceptions, including checked `Exception` and `DragonException` as well as an unchecked `RuntimeException`. All of these are handled by the fact that the method declares the checked `Exception` class in the method signature, which all the exceptions within the class inherit. For this reason, the `openDrawbridge()` method compiles without issue. The call to `openDrawbridge()` in the `main()` method also compiles without issue because the `main()` method declares `Exception` in its signature. For these reasons, the code compiles but a stack trace is printed at runtime, making Option D the correct answer. In case you are wondering, the caller would see `RuntimeException`: Or maybe this one in the stack trace at runtime, since the exception in the finally block replaces the one from the try block. Note that the exception in the catch block is never reached because the `RuntimeException` type declared in the catch block does not handle `Exception`.
- 45.** C. Both `IllegalArgumentException` and `ClassCastException` inherit `RuntimeException`, but neither is a subclass of the other. For this reason, they can be listed in either order, making Option C the correct statement.

46. D. The class `RuntimeException` is not an interface and it cannot be implemented. For this reason, the `Problem` class does not compile, and Option D is the correct answer. Note that this is the only compilation problem in the application. If `implements` was changed to `extends`, the code would compile and `Problem?Fixed!` would be printed, making Option A the correct answer.
47. D. The question is designed to see how closely you pay attention to `throw` and `throws!` The `try` block uses the incorrect keyword, `throws`, to create an exception. For this reason, the code does not compile, and Option D is correct. If `throws` was changed to `throw`, then the code would compile without issue, and Option B would be correct.
48. D. A Java application tends to only throw an `Error` when the application has entered a final, unrecoverable state. Options A and C are incorrect. These types of errors are common and expected in most software applications, and should not cause the application to terminate. Option B uses the word *temporarily*, meaning the network connection will come back. In this case, a regular exception could be used to try to recover from this state. Option D is the correct answer because running out of memory is usually unrecoverable in Java.
49. C. While a `catch` block is permitted to include an embedded `try-catch` block, the issue here is that the variable name `e` is already used by the first `catch` block. In the second `catch` block, it is equivalent to declaring a variable `e` twice. For this reason, line `z1` does not compile, and Option C is the correct answer. If a different variable name was used for either `catch` block, then the code would compile without issue, and Option A would be the correct answer.
50. B. The `finally` block of the `snore()` method throws a new checked exception on line `x1`, but there is no `try-catch` block around it to handle it, nor does the `snore()` method declare any checked exceptions. For these reasons, line `x1` does not compile, and Option B is the correct answer. The rest of the lines of code compile without issue, even line `x3` where a `static` method is being accessed using an instance reference. Note that the code inside the `try` block, if it ran, would produce an `ArrayIndexOutOfBoundsException`, which would be caught by the `RuntimeException` `catch` block, printing `Awake!`. What happens next would depend on how the `finally` block was corrected.

## Chapter 9: Working with Selected Classes from the Java API

1. C. Option A is incorrect because `StringBuilder` does not support multiple threads. In fact, threads aren't even covered on the OCA, which should be your clue that this answer is wrong! You don't need to know this for the exam, but `StringBuffer` supports multiple threads. Option B is incorrect because `==` compares references, not values. Option D is incorrect because both `String` and `StringBuilder` support languages and encodings. Option C is correct and the primary reason to use `StringBuilder`. `String` often creates a new object each time you call certain methods on the object like `concat()`. `StringBuilder` optimizes operations like `append()` because it is mutable.

2. D. A `String` can be created using a literal rather than calling a constructor directly, making Option A incorrect. A string pool exists for `String` reuse, making Option B incorrect. `String` is final and immutable, making Option C incorrect and Option D correct.
3. D. This question is testing whether you understand how method chaining works. Option A creates an empty `StringBuilder` and then adds the five characters in `clown` to it. Option B simply creates the `clown` when calling the constructor. Finally, Option C creates the same value, just in two parts. Therefore, Option D is correct.
4. B. Since `StringBuilder` is mutable, each call to `append` adds to the value. When calling `print`, `toString()` is automatically called and `333 806 1601` is output. Therefore, Option B is correct.
5. B. `List` is an interface and not a class. It cannot be instantiated. While `Object` is a concrete class, it does not implement the `List` interface so it cannot be assigned to `frisbees`. Note that if you were to add an explicit cast, it would compile and throw an exception at runtime. Of the three options, only `ArrayList` can fill in the blank, so Option B is correct.
6. C. An `ArrayList` does not automatically sort the elements. It simply remembers them in order. Since Java uses zero-based indexes, Option C is correct.
7. C. Calling the constructor and then `insert()` is an example of method chaining. However, the `sb.length()` call is a problem. The `sb` reference doesn't exist until after the chained calls complete. Just because it happens to be on a separate line doesn't change when the reference is created. Since the code does not compile, Option C is correct.
8. A. While the `ArrayList` is declared with an initial capacity of one element, it is free to expand as more elements are added. Each of the three calls to the `add()` method adds an element to the end of the `ArrayList`. The `remove()` method call deletes the element at index 2, which is `Art`. Therefore, Option A is correct.
9. C. On line 12, the value of the `StringBuilder` is `12`. On line 13, it becomes `123`. Since `StringBuilder` is mutable, storing the result in the same reference is redundant. Then on line 14, the value is reversed, giving us `321` and making Option C correct.
10. D. Option A is incorrect as it describes autoboxing. Options B and C are not possible in Java. Option D is correct as it describes lambdas. Lambdas use deferred execution and can be run elsewhere in the codebase.
11. D. A `StringBuilder` is mutable, so the length is two after line 6 completes. The `StringBuilder` methods return a reference to the same object so you can chain method calls. Therefore, `line` and `anotherLine` refer to the same object. This means that line 7 prints `true`. Then on line 9, both references point to the same object of length 2, and Option D is correct.
12. D. The `add()` and `get()` methods are available on `ArrayList`. However, `ArrayList` uses `size` rather than `length` to get the number of elements. Therefore, Option D is correct. If `length` was changed to `size`, Option B would compile if put in the blank. Option A still wouldn't compile in the blank because a cast would be needed to store the value in `str`.

13. D. Option A is tricky, but incorrect. While a lambda can have zero parameters, a Predicate cannot. A Predicate is defined as a type mapping to a boolean. Option B is clearly incorrect as `->` separates the parts of a lambda. Options C and D are similar. Option C is incorrect because `return` is only allowed when the brackets are present. Option D is correct.
14. A. Lines 20–22 create an `ArrayList` with two elements. Line 23 replaces the second one with a new value. Now `chars` is `[a, c]`. Then line 24 removes the first element, making it just `[c]`. Option A is correct because there is only one element, but it is not the value `b`.
15. D. Trick question. There is no `reverse` method on the `String` class. There is one on the `StringBuilder` class. Therefore, the code does not compile, and Option D is correct.
16. A. When creating a lambda with only one parameter, there are a few variants. The `pred1` approach shows the shortest way, where the type is omitted and the parentheses are omitted. The `pred2` approach is similar except it includes the parentheses. Both are legal. The `pred4` approach is the long way with both the parentheses and type specified. The only one that doesn't compile is `pred3`. The parentheses are required if including the type.
17. A. This is a correct example of code that uses a lambda. The interface has a single abstract method. The lambda correctly takes one `double` parameter and returns a `boolean`. This matches the interface. The lambda syntax is correct. Since `45` is greater than `5`, Option A is correct.
18. A. Since `String` is immutable, each call to `concat()` returns a new object with the new value. However, that return value is ignored and the `teams` variable never changes in value. Therefore it stays as `694`, and Option A is correct.
19. A. The `ArrayList` class is in the `java.util` package, making I correct. The `LocalDate` class is in the `java.time` package, making II incorrect. The `String` class is in the `java.lang` package, which means you can use it without typing an import, making III incorrect. Therefore, Option A is correct.
20. C. Option A is straightforward and outputs `radical robots`. Option B does the same in a convoluted manner. First Option B removes all the characters after the first one. It doesn't matter that there aren't actually 100 characters to delete. Then it appends `obots` to the end, making the builder contain `robots`. Finally, it inserts the remainder of the string immediately after the first index. Try drawing the flow if this is hard to envision. Option D also creates the same value by inserting `robots` immediately after the end of the `StringBuilder`. Option C is close, but it has an off-by-one error. It inserts `robots` after the letter `l` rather than after the space. This results in the value `radicalrobots` followed by a space. Option C is different than the others and the correct answer.
21. A. Since we are creating the list from an array, it is a fixed size. We are allowed to change elements. At the end of this code, `museums` is `[Art, Science]`. Therefore, it contains `Art`, and Option A is correct.
22. D. Options A and B are not true if the `String` is `"deabc"`. Option C is not true if the `String` is `"abcde"`. Option D is true in all cases.

- 23.** D. Line 25 does not compile. On an `ArrayList`, the method to get the number of elements is `size`. The `length()` method is used for a `String` or `StringBuilder`.
- 24.** B. The `toString()` method call doesn't help in narrowing things down as all Java objects have that method available. The other two methods are more helpful. `String` is the only type of these three to have a `startsWith()` method, making Option B correct. `String` also has the `replace()` method declared here. If you memorized the whole API, you might know that `StringBuilder` also has a `replace()` method, but it requires three parameters instead of two. Please don't memorize the API in that level of detail. We included what you need to know in our study guide. If you do have this outside knowledge, be careful not to read into the questions!
- 25.** B. The `<>` is known as the diamond operator. Here, it works as a shortcut to avoid repeating the generic type twice for the same declaration. On the right side of the expression, this is a handy shortcut. Java still needs the type on the left side so there is something to infer. In the figure, position P is the left side and position Q is the right side. Therefore, Option B is correct.
- 26.** D. The type in the lambda must match the generic declared on the `Predicate`. In this case, that is `String`. Therefore, Options A and B are incorrect. While Option C is of the correct type, it uses the variable `s`, which is already in use from the `main()` method parameter. Therefore, none of these are correct, and Option D is the answer.
- 27.** A. A `String` is immutable so a different object is returned on line 6. The object `anotherLine` points to is of length 2 after line 6 completes. However, the original `line` reference still points to an object of length 1. Therefore, Option A is correct.
- 28.** C. While it is common for a `Predicate` to have a generic type, it is not required. However, it is treated like a `Predicate` of type `Object` if the generic type is missing. Since `startsWith()` does not exist on `Object`, the first line does not compile, and Option C is correct.
- 29.** B. `LocalDate` only includes the date portion and not the time portion. There is no class named `LocalTimeStamp`. The other two, `LocalDateTime` and `LocalTime`, both include the time elements, making Option B correct.
- 30.** D. Line 4 creates a `String` of length 5. Since `String` is immutable, line 5 creates a new `String` with the value 1 and assigns it to `builder`. Remember that indexes in Java begin with 0, so the `substring()` method is taking the values from the fifth element through the end. Since the first element is the last element, there's only one character in there. Then line 6 tries to retrieve the second indexed element. Since there is only one element, this gives a `StringIndexOutOfBoundsException`, and Option D is correct.
- 31.** D. When you're using brackets, both the `return` keyword and semicolon are needed for the lambda to compile, making Option D correct.
- 32.** B. Java 8 date and time classes are immutable. The `plusDays` method returns a `LocalDate` object presenting Christmas Eve (December 24th). However, this return value is ignored. The `xmas` variable still represents the original value, so Option B is correct.

33. A. Line 3 creates an empty `StringBuilder`. Line 4 adds three characters to it. Line 5 removes the first character, resulting in `ed`. Line 6 deletes the characters starting at position 1 and ending right before position 2, which removes the character at index 1, which is `d`. The only character left is `e`, so Option A is correct.
34. B. While it is common for a `Predicate` to have a generic type, it is not required. When the generic is omitted, it is treated like a `Predicate` of type `Object`. Since the `equals()` method exists on `Object`, this is fine. Option B is correct because the `Predicate` tests as `false`.
35. C. In Java, most things use zero-based indexes, including arrays and a `String`. Months are an exception to this convention starting Java 8. This makes the answer either Option C or D. However, `LocalTime` does not contain date fields, so it has to be Option C.
36. C. `Predicate` is an interface with one method. The method signature is `boolean test(T t)`. Option C is the answer because the method accepts one parameter rather than two.
37. B. Be careful here. The `Period` class uses a `static` helper method to return the period. It does not chain method calls, so `period1` only represents three days. Since three days is less than 10 days, `period2` is larger, and Option B is correct.
38. B. The code starts by correctly creating a date representing January 1, 2017, and a period representing one day. It then explicitly defines the format as month followed by day followed by year. Finally, the code subtracts a day, giving us the formatted version of December 31, 2016.
39. C. The `trim()` method returns a `String` with all leading and trailing white space removed. In this question, that's the seven-character `String: ":) - (:"`. Options A and B are incorrect because they do not remove the first blank space in `happy`. Option D is incorrect because it does not remove the last character in `happy`. Therefore, Option C is correct.
40. C. The `Period` class creates immutable objects and is usually used to add/subtract from a `LocalDate` or `LocalDateTime` object. It allows creating date, week, month, or year periods. Since it cannot be used for time, Option C is the answer.
41. D. Line 4 creates a `StringBuilder` of length 5. Pay attention to the `substring()` method `StringBuilder`. It returns a `String` with the value `321`. It does not change the `StringBuilder` itself. Then line 6 is retrieving the second indexed element from that unchanged value, which is `4`. Therefore, Option D is correct.
42. B. This one is tricky. There are two `remove()` methods available on `ArrayList`. One removes an element by index and takes an `int` parameter. The other removes an element by value. Due to the generics, it takes an `Integer` parameter in this example. Since the `int` primitive is a better match, the element with index 2 is removed, which is the value of `1`. Therefore, Option B is correct.
43. C. `ArrayList` has a `size()` method rather than a `length()` method, making Option A incorrect. The `charAt()` and `length()` methods are declared on both `String` and `StringBuilder`. However, the `insert()` method is only declared on a `StringBuilder` and not a `String`. Therefore, Option C is correct.

- 44.** C. The `minusNanos` and `plusNanos` are the smallest units available, making Option C correct. Option D is incorrect because `LocalTime` is not that granular. Note that while you can add milliseconds by adding many nanoseconds, there isn't a method for it. A millisecond is also larger than a nanosecond. Finally, don't be tricked by the fact that `LocalTime` is immutable. You can still add time; it just gets returned as a different object.
- 45.** D. When creating a formatter object, remember that `MM` represents month while `mm` represents minute. Since there are not minutes defined on a `LocalDate` object, the code throws an `UnsupportedTemporalTypeException`. You don't need to know the name of the exception, but you do need to know that an exception is thrown.
- 46.** D. There are two signatures for the `replace()` method. One takes two `char` parameters. The other signature takes a `CharSequence`. Both `String` and `StringBuilder` implement this interface. This makes all three alternatives correct, and Option D is correct.
- 47.** C. Pay attention to the data types. The `print()` method is looping through a list of `String` objects. However, the `Predicate` expects an `Integer`. Since these don't match, the `if` statement does not compile.
- 48.** D. Line 12 creates an empty `ArrayList`. While it isn't recommended to use generics on only the left side of the assignment operator, this is allowed. It just gives a warning. Lines 13 and 14 add two elements. Line 15 resets to an empty `ArrayList`. Line 16 adds an element, so now we have an `ArrayList` of size 1. Line 17 attempts to remove the element at index 1. Since Java uses zero-based indexes, there isn't an element there and the code throws an `IndexOutOfBoundsException`.
- 49.** C. The declaration of `witch` is incorrect. It tries to store a `char` into a `String` variable reference. This does not compile, making Option C correct. If this was fixed, the answer would be Option B.
- 50.** C. The Java 8 date and time classes are immutable. This means they do not contain setter methods and the code does not compile.

## Chapter 10: OCA Practice Exam

1. E. The first time through the loop, we are calling `indexOf` on an empty `StringBuilder`. This returns `-1`. Since we cannot insert at index `-1`, the code throws a `StringIndexOutOfBoundsException`.
2. C, E. In Option A, the assignment operator `=` incorrectly comes after the addition `+` operator. In Option B, the addition operator `+` incorrectly comes after the division `/` operator. In Option D, the subtraction operator `-` incorrectly comes after the multiplication `*` operator. This leaves Options C and E as the correct answers. For these answers, it may help to remember that the modulus operator `%`, multiplication operator `*`, and division operator `/` have the same operator precedence.
3. B, C, F. Option A is incorrect because a getter should not take a value. Option D is incorrect because the prefix `is` should only be with boolean values. Option E is incorrect.

because `gimme` is not a valid JavaBean prefix. Options B, C, and F are each proper JavaBean method signatures.

4. A, E. Line 24 does not compile because arrays use `length`. It is `ArrayList` that uses `size()`. All of the other lines compile, making Option A correct. It is allowed to split up the braces in the 2D array declaration on line 20. The code is also allowed to use `crossword.length` as the loop condition on line 22, although this is not a good idea. The array starts out with all 200 of the cells initialized to the default value for an `int` of 0. Both loops iterate starting at 0 and stopping before 10, which causes only half of the array to be set to 'x'. The other half still has the initial default value of 0, making Option E correct.
5. B, D. Options A and E are incorrect because they indicate states that the application can possibly recover from. An `Error` generally indicates an unrecoverable problem. While it is possible to catch an `Error`, it is strongly recommended that an application never do so, making Options B and D correct. Finally, Option C is incorrect because `Error` extends from `Throwable`, not `Exception`, and is unchecked.
6. A, C, D. The first `import` statement allows only the class `forest.Bird` to be available, making Option A correct and Options E and F incorrect. Option B is incorrect since the third `import` statement only allows access to classes within the `savana` package, not any sub-packages. Option C is correct because the second `import` statement allows any class in the `jungle.tree` package to be accessible. Finally, Option D is correct because `java.lang.*` is implicitly included in all Java classes.
7. C. *Mutable* means the object can change state. *Immutable* means the object cannot change state. An `ArrayList` stores a collection of objects. It mutates as the elements change. A `StringBuilder` is also mutable as it improves performance by not creating a new object each time it changes. A `String` is immutable. Methods that look like they change the value simply return a different `String` object. The date/time objects added in Java 8, such as `LocalDateTime`, are also immutable. Therefore, Option C is correct with `String` and `LocalDateTime` as the immutable object types.
8. C. On the first iteration through the loop, the first five characters are removed and `builder` becomes `s` growing. Since there are more than five characters left, the loop iterates again. This time, five more characters are removed and `builder` becomes `wing`. This matches Option C.
9. D. The code compiles without issue, so Option E is incorrect. The key here is that none of the variables are assigned the same object due to the use of the `new` keyword. Comparing any two variables with `==` will always result in an evaluation of `false`, making the first two values of the `print` statement be `false` and `false`. On the other hand, they all have an underlying `String` value equivalent to `up`, so calling `equals()` on any two variables will return `true`. Option D is the correct answer that matches what the application will print.
10. C. Lines 4 and 5 both print `false` since a `String` should be compared with a method rather than `==`, especially when not comparing two values from the string pool. Line 6 also prints `false` because one value is uppercase and the other is lowercase. Line 7 prints `true` because both values are uppercase. Lines 8 and 9 print `true` because they don't look at the case. This makes Option C the answer.

11. A, B, C. Let's look at each one in turn. Option A is correct because the labels are not referenced. Option B is correct because the outer `while` is broader than the inner `while`. Since there is no other code in the loop, it is not needed. Option C is also correct because a label is not used. Option D is incorrect because the inner loop is more specific than the outer loop. While the code still compiles, it prints one less chapter. Options E and F are incorrect because you cannot remove one half of a loop construct and have it compile.
12. B, C. A `long` cannot contain a number with decimal points, preventing Options B and C from compiling. Options D and E show you can force a number to be a `long` by ending it with an upper- or lowercase `L`. This does not work if the number has a decimal point. Option F shows how to use underscores to break up a number.
13. A. A `while` loop checks the condition before executing. Since the hour is not less than one, the loop never enters, and Option A is correct. This is good, because we'd have an infinite loop if the loop was entered since the result of `plusHours` is ignored.
14. D. This question appears to ask you about involved array logic. Instead, it is checking to see if you remember that instance and class variables are initialized to `null`. Line 6 throws a `NullPointerException`. If the array was declared, the answer would be E because the code would throw an `ArrayStoreException` on line 8.
15. C, E. The diamond operator is only allowed to be used when instantiating rather than declaring. In other words, it can't go on the left side of the equal sign. Therefore, Options B, D, and F are incorrect. The remaining three options compile. However, Option A produces a warning because generics are not used on the right side of the assignment operator. Therefore, Options C and E are correct. Option C is better than Option E since it uses the diamond operator rather than specifying a redundant type.
16. B, D. At the end of the method, `shoe1` and `shoe3` both point to "flip flop". `shoe2` points to "croc". Since there are no references to "sandal", it is eligible for garbage collection, making Option B correct. However, garbage collection is not guaranteed to run, so Option D is also correct.
17. C. The code does not compile, so Options A and B are incorrect. The `getFish()` method is declared properly in the `Fish` class and successfully overridden in the `Clownfish` class. An overridden method must not declare any new or broader checked exceptions, but it is allowed to declare narrower exceptions or drop checked exceptions. The overridden method also uses a covariant return type. The use of `final` on the method and class declarations has no meaningful impact, since the methods and classes are not extended in this application. So where does the compilation error occur? In the `main()` method! Even though the `Clownfish` version of `getFish()` does not declare a checked exception, the call `f.getFish()` uses a `Fish` reference variable. Since the `Fish` reference variable is used and that version of the method declares a checked `Exception`, the compiler enforces that the checked exception must be handled by the `main()` method. Since this checked exception is not handled with a try-catch block nor by the `main()` method declaration, the code does not compile, and Option C is the correct answer.
18. A. This is a correct example of using lambdas. The code creates an `ArrayList` with three elements. The `print()` method loops through and checks for negative numbers. Option A is correct.

- 19.** F. A `try` statement requires a `catch` or a `finally` block. It can also have both a `catch` and a `finally` block. Since no option matches these rules, Option F is the correct answer. Note that `finalize` is not a keyword but a method inherited from `java.lang.Object`. Lastly, the `throws` keyword can be applied to method declarations and is not used as part of a `try` statement.
- 20.** A. On line 12, `result` is first set to 8. On line 13, the boolean condition is true because  $8 > 7$ . On line 13, `result` is incremented to 9. Then the inner loop runs, decrementing `result` until it is no longer greater than 5. On line 18, loop execution is completed because `result` is equal to 5. The `break` statement says to skip to after the labeled loop, which is line 20. Then `result` is printed as 5, making Option A correct.
- 21.** C. The code compiles and runs without exception, making Options E and F incorrect. The question is testing your knowledge of variable scope. The `teeth` variable is `static` in the `Alligator` class, meaning the same value is accessible from all instances of the class, including the `static main()` method. The `static` variable `teeth` is incremented each time the constructor is called. Since `teeth` is a local variable within the `snap()` method, the argument value is used, but changes to the local variable do not affect the `static` variable `teeth`. Since the local variable `teeth` is not used after it is decremented, the decrement operation has no meaningful effect on the program flow or the `static` variable `teeth`. Since the constructor is called twice, with `snap()` executed after each constructor call, the output printed is 1 2, making Option C the correct answer.
- 22.** A. A `String` is immutable. Since the result of the `concat()` method call is ignored, the value of `witch` never changes. It stays as a single letter, and Option A is correct.
- 23.** A, C, F. An interface method is exactly one of three types: `default`, `static`, or `abstract`. For this reason, Options A, C, and F are correct. An interface method cannot be `protected` nor `private` because the access modifier is always `public`, even when not specified, making Options B and D incorrect. Option E is also incorrect because `final` cannot be applied to `static` methods, since they cannot be overridden. It can also not be applied to `default` and `abstract` methods because they are always able to be overridden.
- 24.** D. Look at the loop condition carefully. It tries to assign `null` to a `String` variable. This is not an expression that returns a `boolean`. Therefore, the code does not compile, and Option D is correct. If this was fixed by making the loop condition `tie == null`, then Option B would be correct.
- 25.** B, F. A class may be defined without a `package` statement, making the class part of the default package. For this reason, Options A and D are incorrect. Every Java class implicitly imports exactly one package, `java.lang.*`, making Option B correct and Option C incorrect. Option E is incorrect because an `import` statement is not required. Finally, Option F is correct; any class that does not extend another class implicitly extends `java.lang.Object`.
- 26.** D. A class cannot inherit two interfaces that declare the same `default` method, unless the class overrides them. In this case, the version of `grow()` in the `Tree` class is an overloaded method, not an overridden one. Therefore, the code does not compile due to the declaration of `Tree` on line `m1`, and Option D is the correct answer.

- 27.** D. Variables are allowed to start with an underscore and are allowed to contain a \$. Therefore, all the variable declarations compile, making Options A, B, and C incorrect. However, the `println()` refers to the uninitialized local boolean. Since local variables are not automatically initialized, the code does not compile, and Option D is correct.
- 28.** A. Prefix operators, such as `-x` and `++x`, modify the variable and evaluate to the new value, while postfix operators, such as `x--` and `x++`, modify the variable but return the original value. Therefore, Option A is the correct answer.
- 29.** B, C, E. The constructors declared by Options A, D, and F compile without issue. Option B does not compile. Since there is no call to a parent constructor or constructor in the same class, the compiler inserts a no-argument `super()` call as the first line of the constructor. Because `Big` does not have a no-argument constructor, the no-argument constructor `Trouble()` does not compile. Option C also does not compile because `super()` and `this()` cannot be called in the same constructor. Note that if the `super()` statement was removed, it would still not compile since this would be a recursive constructor call. Finally, Option E does not compile. There is no matching constructor that can take a `String` followed by a `long` value. If the input argument `deep` was an `int` in this constructor, then it would match the constructor used in Option D and compile without issue.
- 30.** E, F. A `static` method is not allowed to access instance variables without an instance of the class, making Options E and F correct. Notice that only `max` is initialized to `100` in Option E. Since `min` doesn't have a value specified, it gets the default value, which is `0`.
- 31.** B, E. The ternary `? :`  operator only evaluates one of the two right-hand expressions at runtime, so Option A is incorrect. A `switch` statement may contain at most one optional `default` statement, making Option B correct. A single if-then statement can have at most one `else` statement, so Option C is incorrect. Note that you can join if-then-else statements together, but each `else` requires an additional if-then statement. The disjunctive `|` operator will always evaluate both operands, while the disjunctive short-circuit `||` operator will only evaluate the right-hand side of the expression if the left-hand side evaluates to `false`. Therefore, they are not interchangeable, especially if the right-hand side of the expression modifies a variable. For this reason, Option D is incorrect. Finally, Option E is correct. The logical complement `!` operator may only be applied to boolean expressions, not numeric ones.
- 32.** C. Line 3 creates an empty `StringBuilder`. Line 4 adds three characters to it. Line 5 removes the first character resulting in `ed`. Line 6 deletes the characters starting at position 1 and ending right before position 1. Since there are no indexes that meet that description, the line has no effect. Therefore, Option C is correct.
- 33.** A, D. Java methods must start with a letter, the dollar `$` symbol, or the underscore `_` character. For this reason, Option B is incorrect, and Options A and D are correct. Despite how Option A looks, it is a valid method signature in Java. Options C, E, and F do not compile because the symbols `-`, `\`, and `#` are not allowed in method names, respectively.
- 34.** B, C. First off, Option A is incorrect, since whether or not `static` or inherited methods are chosen is a matter of design and individual preference. Options B and C are true statements about inheritance and two of the most important reasons Java supports inheritance. Option D is incorrect because all Java classes extend `java.lang.Object`. Option E is incorrect. Whether or not inheritance simplifies or complicates a design is based on the skills of the developer creating the application.

- 35.** E. All of the statements are true statements about Java, making Option E the correct answer. Java was built with object-oriented programming and polymorphism in mind. Also, Java supports functional programming using lambda expressions.
- 36.** C. This array has three elements, making `listing.length` output 3. It so happens that each element references an array of the same size. But the code checks the first element and sees it is an array of size two, making the answer Option C.
- 37.** A, B, E. A switch statement supports the primitive types `byte`, `short`, `char`, and `int` and their associated wrapper classes `Character`, `Byte`, `Short`, and `Integer`. It also supports the `String` class and enumerated types. Floating-point types like `float` and `double` are not supported, nor is the `Object` class. For these reasons, Options A, B, and E are correct.
- 38.** D. The lambda syntax is incorrect. It should be `->`, not `=>`. Therefore, Option D is correct. If this was fixed, Option A would be correct.
- 39.** B, C, E. The `/* */` syntax can have additional (and uneven) `*` characters in Java, making Options B and E correct. Option C is the standard way to comment a single line with two slashes `//`. Option A contains a `*/` in the middle of the expected comment, making the part after the comment `Insert **/` invalid. Option D is incorrect because a single slash `/` is not valid comment in Java. Finally, the `#` is not a comment character in Java, so Option F is incorrect.
- 40.** A, F. A `static` import is used to import `static` members of another class. Option A is correct because the method `getGrass` and variable `seeds` are imported. Option F is also correct because a wildcard on the `Grass` class for all visible `static` members is allowed. Option B is incorrect because the wildcard must be on a class, not a package. Options C and E are incorrect since the keywords `import` and `static` are reversed. Option D is incorrect because the `static` keyword is missing.
- 41.** D. When converting an array to a `List`, Java uses a fixed-sized backed list. This means that the list uses an array in the implementation. While changing elements to new values is allowed, adding and removing elements is not.
- 42.** A, D. Variable names can begin with an underscore, making Option A correct. To use an underscore in a numeric literal, it must be between two digits, making Option D correct.
- 43.** B. While no arguments are passed from the command line, this doesn't matter because the `main()` method redefines the `args` array. Remember that `String` values sort alphabetically rather than by number. Therefore, `01` sorts before `1`, and Option B is correct.
- 44.** D. The `public` modifier allows access members in the same class, package, subclass, or even classes in other packages, while the `static` modifier allows access without an instance of the class. For these reasons, Option D is the correct answer. Option A is incorrect because `final` is not related to access, and package-private prevents access from classes outside the package. Option B is incorrect because `class` is not a modifier; it is a keyword. Option C is incorrect because `instance` is not a Java keyword or modifier, and `protected` prevents classes that are not subclasses and are outside the package from accessing the variable. Finally, Option E is incorrect. The `default` keyword is for interface methods and `switch` statements, not class variables.

- 45.** A. Looping through the same list multiple times is allowed. Notice how there are not braces around the loops. This means that only the `print` statement is inside the loop. It executes four times. However, the `println()` only executes once at the end, making Option A the answer.
- 46.** C, D. The `javac` command compiles a `.java` file into a `.class` bytecode file, making Option C a correct answer, while also making Options B, E, and F incorrect. The `javac` command compiles to a set of `java` instructions, or bytecode, not machine instructions, making Option A incorrect and Option D correct.
- 47.** C. The `parseInt()` method returns an `int` primitive. Thanks to autoboxing, we can also assign it to an `Integer` wrapper class object reference. The `char` and `short` types are smaller than `int` so they cannot store the result. Therefore, lines 3 and 4 compile, and Option C is correct.
- 48.** B, D, F. The compiler will broaden the data type on a numeric value until it finds a compatible signature. There are two versions of the `drive()` methods that return a value of 3, one that takes a `short` and one that takes a `double`. Option A is incorrect because `boolean` cannot be converted to either of these types and trying to do so triggers a compiler error. Option B is correct because the data type `short` matches our message signature. Options C and E are incorrect. Remember that `int` and `long` are larger than `short` and will trigger different overloaded versions of `drive()` to be called, one that returns 5 and one that returns 2, respectively. Option D is correct. The `byte` value can be implicitly converted to `short`, and there are no other matching method signatures that take a `byte` value. Finally, Option F is correct because `float` can be implicitly converted to `double`, and there is no other version of `drive()` that takes a `float` value.
- 49.** A. Trick question. This appears to be about equality, but it is really about you recognizing that the `main()` method is missing the `static` keyword. Running this problem gives a runtime exception because the `main()` method is not properly declared. Therefore, Option A is the answer. If this was fixed, the answer would be Option C because the `int` and `String` comparisons return `true`.
- 50.** D. The code compiles without issue, so Options E and F are incorrect. Note that line `p2` accesses a `static` method using an instance reference, which is discouraged but permitted in Java. First, a varargs `int` array of `[0, 0]` is passed to the `swing()` method. The `try` block throws `ArrayIndexOutOfBoundsException`, since the third element is requested and the size of the array is two. For this reason, the `print()` statement in the `try` block is not executed. Next, since `ArrayIndexOutOfBoundsException` is a subclass of `RuntimeException`, the `RuntimeException` `catch` block is executed and 2 is printed. The rest of the `catch` blocks are skipped, since the first one was selected. The `finally` block then executes and prints 4. Lastly, control is returned to the `main()` method without an exception being thrown, and 5 is printed. Since 245 is printed, Option D is the correct answer.
- 51.** E. In the first iteration through the loop, `container` is 2 and `cup` is printed. Notice how the loop body subtracts 1 to account for indexes being zero based in Java. Then the update statement runs, setting `container` to 3. The condition is run and sees that 3 is in fact greater than 0. The loop body subtracts 1 and tries to get the element at index 2. There isn't one and the code throws an exception. This makes Option E correct. You might be tempted to think this is an infinite loop. If the body did not throw an exception, it would be!
- 52.** A, E, F. An entry point in a Java application consists of a `main()` method with a single `String[]` or vararg `String...` argument, return type of `void`, and modifiers `public` and

`static`. Note that the name of the variable in the input argument does not matter and the `final` modifier is optional. Options A, E, and F match this description and are correct. Option B is incorrect because the argument is a single `String`. Option C is incorrect, since the access modifier is incorrectly marked `protected`. Finally, Option D is incorrect because it has two return types, `int` and `void`.

53. C, D, E. For this question, it helps to remember that the value of a `case` statement must be a literal expression or a `final` constant variable, and have a compatible data type. For these reasons, Lines 10 and 12 do not compile, making Options C and E correct answers. Line 10 uses a constant value, but `long` is not compatible with `switch` statements, while Line 12 uses a variable that is not marked `final`. Next, a `switch` statement may only have one `default` block. Therefore, Line 11 or 14 must be removed. Since Line 14 is not in the list of options, Option D becomes the last correct answer. The rest of the lines are fine since removing Lines 10, 11, and 12 allows the code to compile.
54. A, B, C. All of the compilation issues with this code involve access modifiers. First, all interface methods are implicitly `public`, and explicitly setting an interface method to `protected` causes a compilation error on line `h1`, making Option A correct. Next, lines `h2` and `h3` both override the interface method with the package-private access modifier. Since this reduces the implied visibility of `public`, the overrides are invalid and neither line compiles. Therefore, Options B and C are also correct. Note that the `RuntimeException` is allowed in an overridden method even though it is not in the parent method signature because only new checked exceptions in overridden methods cause compilation errors. Line `h4` is valid. An object can be implicitly cast to a superclass or inherited interface. Finally, lines `h5` and `h6` will compile without issue but independently throw a `ClassCastException` and a `NullPointerException` at runtime, respectively. Since the question only asks about compilation problems, neither of these are correct answers.
55. B, E, F. Unchecked exceptions inherit the `RuntimeException` class and are not required to be caught in the methods where they are declared. Since `ArithmetricException` and `IllegalArgumentException` extend `RuntimeException`, they are included as unchecked exceptions, making Options B, E, and F correct. The rest are checked exceptions, which inherit `Exception` but not `RuntimeException`.
56. F. The code compiles without issue, making Options D and E incorrect. Applying the ternary `? : operator`, the variable `ship` is assigned a value of `10.0`. The expression in the first if-then statement evaluates to `true`, so `Goodbye` is printed. Note that there is no `else` statement between the first and second if-then statements, therefore the second if-then statement is also executed. The expression in the second if-then statement evaluates to `false`, so the `else` statement is called and `See you again` is also printed. Therefore, Option F is the correct answer, with two statements being printed.
57. B, C, D. The `clock` variable is accessed by a class in the same package; therefore, it requires package-private or less restrictive access (`protected` and `public`). The `getTime()` method is accessed by a subclass in a different package; therefore, it requires `protected` or less restrictive access (`public`). Options B, C, and D conform to these rules, making them the correct answer. Options A and F cause the `Snooze` class to fail to compile because the `getTime()` method is not accessible outside the package, even though `Snooze` is a subclass of `Alarm`. Option E causes the `Coffee` class to fail to compile because the `clock` variable is only visible within the `Alarm` class.

- 58.** B. This problem appears to be to be about overriding a method, but in fact, it is much simpler. The class `CarbonStructure` is not declared `abstract`, yet it includes an `abstract` method. To fix it, the definition of `CarbonStructure` would have to be changed to be an `abstract` class, or the `abstract` modifier would need to be removed from `getCount()` in `CarbonStructure` and a method body added. Since the only answer choice available is to change the `getCount()` method on line `q1`, Option B is the correct answer. Note that the rest of the application, including the override on line `q2`, is correct and compiles without issue. The return types `Long` and `Number` are covariant since `Number` is a superclass of `Long`. Likewise, the exception thrown in the subclass method is narrower, so no compilation error occurs on `q2`.
- 59.** C. Line 5 does not declare a `main()` method that can be the entry point to the program. It does correctly declare a regular instance method and does compile. Line 6 does not compile because `LocalDate` needs to use a static method rather than a constructor. Line 7 is incorrect because `Period` methods should not be chained. However, it does compile, returning a period of 1 day. Line 8 does not compile because the correct class name is `DateTimeFormatter`. Line 9 is correct. Option C is correct because lines 6 and 8 do not compile.
- 60.** A, E. A `try` block can have zero or more `catch` blocks, and zero or one `finally` blocks, but must be accompanied by at least one of these blocks. For these reasons, Options B, D, and F are incorrect, and Option E is correct. A `finally` block must appear after the last `catch` block, if there are any, making Option C incorrect, and Option A correct.
- 61.** B. The code compiles without issue, so Option E is incorrect. For this problem, it helps to remember that `+` and `*` have a higher precedence than the ternary `? :`  operator. In the first expression, `1 + 2 * 5` is evaluated first, resulting in a reduction to `11 >= 2 ? 4 : 2`, and then `fish` being assigned a value of 4. In the second expression, the first ternary expression evaluates to false resulting in a reduction to the second right-hand expression `5 >= 5 ? 9 : 7`, which then assigns a value of 9 to `mammals`. In the `print()` statement, the first `+` operator is an addition operator, since the operands are numbers, resulting in the value of `4 + 9`, 13. The second `+` operator is a concatenation since one of the two operands is a `String`. The result 13 is printed, making Option B the correct answer.
- 62.** A, C, E. An object can be cast to a superclass or inherited interface type without an explicit cast. Furthermore, casting an object to a reference variable does not modify the object in any way; it just may change what methods and variables are immediately accessible. For these reasons, Options A, C, and E are correct. Option B is incorrect; since the compiler can try to block or warn about invalid casts, it cannot prevent them. For example, any object can be implicitly cast to `java.lang.Object`, then explicitly cast to any other object, leading to a `ClassCastException` at runtime. Option D is also incorrect because assigning an object to a subclass reference variable requires an explicit cast. Finally, Option F is incorrect. An object can always be cast to one of its inherited types, superclass or interface, without a `ClassCastException` being thrown.
- 63.** F. The array is not sorted. It does not meet the pre-condition for a binary search. Therefore, the output is not guaranteed and the answer is Option F.
- 64.** B. While `shoe3` goes out of scope after the `shopping()` method, the `croc` object is referenced by `shoe1` and therefore cannot be garbage collected. Similarly, the `sandal` object is now referenced by `shoe2`. No variables reference the `flip flop` object, so it is eligible to be garbage collected, and Option B is correct.

- 65.** E. The `throws` keyword is used in method declarations, while the `throw` keyword is used to throw an exception to the surrounding process, and the `finally` keyword is used to add a statement that is guaranteed to execute even if an exception is thrown. For these reasons, Option E is the correct answer.
- 66.** B, E. The first two iterations through the loop complete successfully, making Option B correct. However, the two arrays are not the same size and the `for` loop only checks the size of the first one. The third iteration throws an `ArrayIndexOutOfBoundsException`, making Option E correct.
- 67.** E. For this question, it helps to try all answers out. Most of them do not make any sense. For example, overloading a method is not a facet of inheritance. Likewise, concrete and abstract methods can both be overridden, not just one. The only answer that is valid is Option E. Without virtual methods, overriding a method would not be possible, and Java would not truly support polymorphism.
- 68.** E. The code does compile. Line `s1` is a bit tricky because `length` is used for an array and `length()` is used for a `String`. Line `s1` stores the length of the `Fall` in a variable, which is 4. Line `s2` throws an `ArrayIndexOutOfBoundsException` because 4 is not a valid index for an array with four elements. Remember that indices start counting with zero. Therefore, Option E is correct.
- 69.** D. The code definitely does not compile, so Option A is incorrect. The first problem with this code is that the `Drum` class is missing a constructor causing the class declaration on line 8 to fail to compile. The default no-argument constructor cannot be inserted if the super-class, `Instrument`, does not define a no-argument constructor. The second problem with the code is that line 11 does not compile, since it calls `super.play(5)`, but the version of `play()` in the parent class does not take any arguments. Finally, line 15 does not compile. While `mn` may be a reference variable that points to a `Drum()` object, the `concert()` method cannot be called unless it is explicitly cast back to a `Drum` reference. For these three reasons, the code does not compile, and Option D is the correct answer.
- 70.** B. The application compiles and runs without issue, so Options E and F are incorrect. Java uses pass-by-value, so even though the change to `length` in the first line of the `adjustPropellers()` method does not change the value in the `main()` method, the value is later returned by the method and used to reassign the `length` value. The result is that `length` is assigned a value of 6, due to it being returned by the method. For the second parameter, while the `String[]` reference cannot be modified to impact the reference in the calling method, the data in it can be. Therefore, the value of the first element is set to `LONG`, resulting in an output of 6, `LONG`, making Option B the correct answer.
- 71.** D. The first compilation problem with the code is that the second catch block in `openDrawbridge()` is unreachable since `CableSnapException` is a subclass of `OpenDoorException`. The catch blocks should be ordered with the more narrow exception classes before the broader ones. Next, the variable `ex` is declared twice within the same scope since it appears in the second catch block as well as the embedded try-catch block. Finally, the `openDrawbridge()` method declares the checked `Exception` class, but it is not handled in the `main()` method with a try-catch block, nor in the `main()` method declaration. For these three reasons, Option D is correct.

- 72.** D. Object orientation is the property of structuring an object with its related data and methods. Encapsulation is the property of removing direct access to the underlying data from processes outside the class. The two go hand and hand to improve class design, making Option D the correct choice.
- 73.** E. In Java, `String` is a class and not a primitive. This means it needs to begin with an uppercase letter in the declaration. The code does not compile, making Option E correct. If this was fixed, the answer would be Option B.
- 74.** A. This class is called with three command-line arguments. First the array is sorted, which meets the pre-condition for binary search. At this point, the array contains `[flower, plant, seed]`. The key is to notice the value of `args[0]` is now `flower` rather than `seed`. Calling binary search to find the position of `flower` returns `0`, which is the index matching that value. Therefore, the answer is Option A.
- 75.** B, C, D. A `for-each` loop is a specialized loop that just iterates through an array or list. It can be rewritten using explicit indexing code in any of the other three loop types. Therefore, Options B, C, and D are correct. Option A is incorrect because a `do-while` loop is guaranteed to execute at least once. Option E is incorrect because the traditional `for` loop can loop backwards or by skipping indexes. Option F is incorrect because non-index-related boolean conditions are allowed to be used in a `while` loop.
- 76.** E. The `LocalDate` class is only for day/month/year values. It does not support time, so `getHour()` and `plusHours()` do not compile, making Option E the answer.
- 77.** C. All arrays are objects regardless of whether they point to primitives or classes. That means both `balls` and `scores` are objects. Both are set to `null` so they are eligible for garbage collection. The `balls` array is initialized to all `null` references. There are no objects inside. The `scores` array is initialized to all 0 values. Therefore, only two objects exist to be eligible for garbage collection, and Option C is correct.
- 78.** B. Since there are not brackets around the `while` loop, only line 17 is in the loop body. Line 18 gets executed once after the loop completes. This means that `count` will be 1 assuming the loop completes. Subtracting a month from JANUARY results in DECEMBER. Since the loop completes E is incorrect and Option B is the answer. Note that if the brackets were added as the indentation suggests, Option D would be the answer since we are counting months backwards.
- 79.** D. Line 10 does not compile because the override reduces the visibility of an inherited method, with the package-private modifier being more restrictive than the `protected` modifier. Line 11 does also not compile, since the left-hand side of a compound assignment operator must be used with a variable, not a method. Finally, Line 12 does not compile because `super.grunt()` is inherited as an `abstract` method in the `PolarBear` class, meaning the parent class has no implementation. For these three reasons, Option D is the correct answer.
- 80.** B, E. Package-private, or default, access is denoted by the absence of an access modifier, making Option A incorrect. Option B is correct, since a `switch` statement can contain a default execution path. Options C and F are incorrect because keywords in Java cannot be used as method or variable names. Finally, interfaces can contain default interface methods but they must be concrete with a method body. For this reason, Option E is correct and Option D is incorrect.

# Chapter 11: Java Class Design

1. D. The `toString()` method is declared in the `Object` class. Therefore it is available to be called in any Java class and is overridden in some. Java automatically calls the `toString()` method when you print an object, making Option D correct. Option C is incorrect because `toString()` is a method, not a variable.
2. B. This code is not a singleton because it has a `public` constructor. Remember that a `public` no-argument constructor is provided automatically if no constructor is coded. This code is well encapsulated because the instance variable is `private`. It is not immutable since there is a setter method. Therefore, Option B is correct.
3. C. The singleton pattern ensures there will be no more than one instance of the object. Depending on how it is implemented, it is possible for there to be zero instances. But it is not possible to have more than one, making Option C correct. Option D means the variable is shared across instances or even without an instance being created but does not limit the number of the instances of the class itself.
4. C. Both objects are instances of the class `Laptop`. This means the `startup()` method in the `Laptop` class gets called both times thanks to polymorphism.
5. D. We know that the variable `o` that `equals()` is called on isn't `null`, since we can't call instance methods on a `null` reference. However, a `null` reference could be passed as a method parameter. If a `null` is passed in, the method should return `false` since an object and a `null` are not equal. Options A and B are incorrect because the first line of those methods should return `false` rather than `true`. Option C is incorrect because the cast is missing. The `Object` class does not have a `text` variable available. Option D shows a properly implemented `equals()` method and is correct.
6. A. Option A is correct because mutability means the state can change and immutability means it cannot. In Option C, static means the state isn't tied to an instance. In Option B, rigidity is not a common programming term.
7. B. The `Hammer` class is a subclass of the `Tool` class. Since the `use()` method in `Hammer` is intended to override the one in `Tool`, there are certain rules. One is that the access modifier must not be more specific. Therefore, trying to make it `private` is a problem. Option B is correct and `r2` is the only line with a compiler error in this code.
8. D. The singleton pattern requires that only one instance of the class exist. Neither of these classes meets that requirement since they have the default no-argument constructor available. There should have been a `private` constructor in each class. Therefore, Option D is correct. Remember that the exam doesn't always include import statements to simplify the code you need to read.
9. B. While using `null` with `instanceof` compiles, it always returns `false`. The other two `instanceof` calls show that `instanceof` can be used with both classes and interfaces. They both return `true`, making Option B correct.
10. D. The `static` keyword is used to create a class-level variable, making Option D correct. Note that a singleton is where you limit a class so only one instance can be created. This means there are not multiple instances to share a variable across.

11. A. Option A is a requirement of a singleton class rather than an immutable one. The other three options are requirements of an immutable class.
12. C. If the variables are `public`, the class is not encapsulated because callers have direct access to them. This rules out Options A and B. Having `private` methods doesn't allow the callers to use the data, making Option D an undesirable answer. Option C is correct and the classic definition of encapsulation where the data is not exposed directly.
13. A. While both objects are instances of `Laptop`, we are not calling methods in this example. Virtual method invocation only works for methods, not instance variables. For instance variables, Java actually looks at the type of the reference and calls the appropriate variable. This makes each reference call a different class's instance variable in this example, and Option A is correct.
14. B. An immutable class must not allow the state to change. In the `Flower` class, the caller has a reference to the `List` being passed in and can change the size or elements in it. Similarly, any class with a reference to the object can get the `List` by calling `get()` and make these changes. The `Flower` class is not immutable. The `Plant` class shows how to fix these problems and is immutable. Option B is correct.
15. C. An instance method can access both instance variables and `static` variables. Both methods compile and Option C is correct.
16. B. A `static` method can access `static` variables, but not instance variables. The `getNumRakes()` method does not compile, so Option B is correct.
17. A. You are allowed to use `null` with `instanceof`; it just prints `false`. The `bus` variable is both a `Vehicle` and a `Bus`, so lines 18 and 19 print `true`. Then it gets interesting. We know that `bus` is not an `ArrayList` or `Collection`. However, the compiler only knows that `bus` is not an `ArrayList` because `ArrayList` is a concrete class. Line 20 does not compile. The compiler can't definitively state that `bus` is not a `Collection`. Some future program could create a subclass of `Bus` that does implement `Collection`, so this line compiles. Therefore, only line 20 fails to compile, and Option A is correct.
18. B. `Building` and `House` are both properly declared inner classes. Any `House` object can be stored in a `Building` reference, making the declarations for `p` and `r` compile. The declaration for `s` is also correct. It so happens that `bh` is a `House` object, so the cast works. The declaration of `q` is a problem though. While the cast itself is fine, a `Building` cannot be stored in a `House` reference, which means the assignment fails to compile. Option B is correct and is the only line with a compiler error in this code. Note that if the declaration of `q` was removed, the declaration of `p` would produce a `ClassCastException` at runtime.
19. D. If two instances of a class have the same hash code, they might or might not be equal. The reverse is not true. If two objects are equal, they must have the same hash code in order to comply with the contracts of these methods. However, in this case, the answer is none of the above because the method can't simply return `true` or `false`. Based on the rules of `equals()`, if `null` is passed in, the result must be `false`. If an object identity is passed in, the result must be `true` due to reflexivity. As a result, Option D is correct.
20. D. This class is a good example of encapsulation. It has a `private` instance variable and is accessed by a `public` method. No changes are needed to encapsulate it, and Option D is correct.

21. B. The singleton pattern requires that only one instance of the class exist. The `ExamAnswers` class is close. However, `getExamAnswers()` is not `static`, so you can't retrieve the instance. Option B is the answer because `TestAnswers` is a correct implementation. It has a `static` variable representing the one instance and a `static` method to retrieve it.
22. C. The `static` initializer is only run once. The `static` method is run twice since it is called twice. Therefore, three lines are printed, and Option C is correct.
23. C. Option A is allowed because the `turnOn()` method is `public` and can be called from anywhere. Options B and D are allowed since the method is in the same class, which is always allowed! Option C is not allowed because `wash()` is a package-private method in another package. Option C is the correct answer.
24. B. The `display()` method has `protected` access. This means it can be accessed by instance methods in the same package and any subclasses. There are no subclasses in this example, so we only need to count the classes in the same package. Option B is correct because `Flashlight` and `Phone` are in the package.
25. B. Line 15 calls the method on line 9 since it is a `Watch` object. That returns `watch`, making Option A incorrect. Line 16 calls the method on line 3 since it is a `SmartWatch` object and the method is properly overridden. That returns `smart watch`, so Option B is the answer, and Option C is incorrect.
26. A. Clearly a `Bus` is a `Vehicle` since the `Bus` class implements `Vehicle`. The `Van` class is also a `Vehicle` since it extends `Bus`. This question also confirms you know that arrays can be tested with `instanceof`, which they can. Therefore, Option A is correct.
27. C. There is no `instanceOf` keyword, making Options B and D incorrect. There is an `instanceof` keyword. If an object is the wrong type, the `equals()` method should return `false`, making Option C the answer.
28. D. The `Hammer` class is a subclass of the `Tool` class. Luckily, the `use()` method has a different signature so it is not an override. This means it is fine that the access modifier is stricter, and Option D is correct. Line `r3` is a valid method unrelated to the superclass.
29. B. Lazy instantiation is part of a possible implementation for the singleton pattern. It defers creating the object until the first caller requests it. While this does save memory, it only does so if the object is never requested. This does not save memory when actually creating the object. Option B is correct.
30. D. Notice how the code begins at line 30. This means you have to infer the surrounding code. Here it is reasonable to assume the classes are inner classes. `Building` and `House` are defined correctly. Any `House` or `Building` reference can potentially be a `House`. The compiler does not know which ones work and which don't. This means all three casts compile.
31. C. Encapsulation doesn't allow callers access to the instance variables, which makes it easier to change the code. The instance variables can be any type, which means they can be mutable or immutable. There are no constraints on the implementation of methods. The purpose of encapsulation is to lessen how tightly tied or coupled the classes are. Option C is the opposite of this, making it the answer.

- 32.** A. An immutable class must not allow the state to change. The `Flower` class does this correctly. While the class isn't `final`, the getters are, so subclasses can't change the value returned. The `Plant` class lacks this protection, which makes it mutable. Option A is correct.
- 33.** D. A `static` initializer is not allowed inside of a method. It should go on the class level rather than the method level. Therefore, the code does not compile, and Option D is correct.
- 34.** A. An object is required to have the same value for repeated calls to `hashCode()` if the value has not changed. This makes III and IV incorrect. If two objects are equal, they are required to have the same hash code. Since equality must be reflexive, it cannot return `false` if the same object is passed, and I is incorrect. Since `equals()` must return `false` when `null` is passed in, it cannot be `true` and II is incorrect. Therefore, Option A is the answer.
- 35.** D. By definition, you cannot change the value of an instance variable in an immutable class. There are no setter methods, making Option A incorrect. While Option B would allow you to set the value, the class would no longer be immutable. Option D is correct. If you are an advanced developer, you might know that you can use reflection to change the value. Don't read into questions like this on the exam. Reflection isn't on the exam, so you can pretend it doesn't exist.
- 36.** B. Option A is incorrect because the "is-a" principle is about inheritance. For example, a `String` is an `Object`. Option C is incorrect because singletons require a `static` variable to ensure there is only one instance. While it is common to have instance variables as well, this is not required to implement the pattern. Option B is correct. For an object to be composed of other objects, instance variables are required.
- 37.** B. The `static` initializer only runs once since statics are shared by all instances. The instance initializer runs twice because we call the constructor twice. Therefore, Option B is correct.
- 38.** A. While there is a `default` keyword in Java, it is only allowed in interfaces or in `switch` statements. It is not a visibility modifier. The author of this code probably intended for the method to be package-private, which doesn't use a visibility modifier. The line with `default` doesn't compile, so Option A is correct. If `default` was removed, the code would all compile.
- 39.** A. The reference `b` points to a `Building` object, which cannot be stored in a `House` reference. This means the assignment to `p` compiles but fails at runtime. The other two casts would run without issue if the code got that far.
- 40.** C. The `hashCode()` method in the `Object` class does not have a parameter. Therefore, the `Sticker` class provides an overloaded method rather than an overridden one. Since it is not an overridden method, the contract for the `Object` class' `hashCode()` method does not apply, and any `int` value can be returned. Therefore, Option C is correct.

## Chapter 12: Advanced Java Class Design

- 1.** B. The lambda expression `s -> true` is valid, making Options A, C, and D incorrect. Parentheses `()` are not required on the left-hand side if there is only one variable. Brackets `{}` are not required if the right-hand side is a single expression. Parameter data types are only required if the data type for at least one parameter is specified, otherwise none are

required. The remaining choice, the arrow operator `->`, is required for all lambda expressions, making Option B the correct answer.

2. D. The application contains a compilation error. The `case` statements incorrectly use the enum name as well as the value, such as `DaysOff.ValentinesDay`. Since the type of the enum is determined by the value of the variable in the `switch` statement, the enum name is not allowed and throws a compilation error when used. For this reason, Option D is correct. If the enum name `DaysOff` was removed, the application would output 12, since the lack of any `break` statements causes multiple blocks to be reached, and Option C would have been the correct answer.
3. C. A functional interface must include exactly one `abstract` method, either by inheritance or declared directly. It may also have any number, including zero, of `default` or `static` methods. For this reason, both parts of Option D are incorrect. The first part of Option A is incorrect because more than one `abstract` method disqualifies it as a functional interface. The first part of Option B is incorrect because the method must be `abstract`; that is to say, any method will not suffice. Finally, Option C is the correct answer. The first part of the sentence defines what it means to be a functional interface. The second part refers to the optional `@FunctionalInterface` annotation. It is considered a good practice to add this annotation to any functional interfaces you define because the compiler will report a problem if you define an invalid interface that does not have exactly one `abstract` method.
4. C. While an anonymous inner class can extend another class or implement an interface, it cannot be declared `final` or `abstract` since it has no class definition. For this reason, Option C is correct. The other classes may be declared `final` or `abstract` since they have a class definition.
5. B. Option A is incorrect because the lambda expression is missing a semicolon (`;`) at the end of the `return` statement. Option C is incorrect because the local variable `test` is used without being initialized. Option D is also incorrect. The parentheses are required on the left-hand side of the lambda expression when there is more than one value or a data type is specified. Option B is the correct answer and the only valid lambda expression.
6. B. An enum cannot be marked `abstract`, nor can any of its values, but its methods can be marked `abstract`, making Option B the correct answer. Note that if an enum contains an `abstract` method, then every enum value must include an override of this `abstract` method.
7. B. The code compiles without issue, so Option D is incorrect. The first `print()` statement refers to `value` in the `Deeper` class, so 5 is printed. The second and third `print()` statements actually refer to the same value in the `Deep` class, so 2 is printed twice. The prefix `Matrix.` is unnecessary in the first of the two `print()` statements and does not change the result. For these reasons, Option B is the correct answer.
8. D. A local inner class can access `final` or effectively `final` local variables, making Option D the correct answer. The second statement is invalid because access modifiers like `private` cannot be applied to local variables.
9. C. The type of the variable in the `switch` statement is the enum `Currency`, but the `case` statements use `int` values. While the enum class hierarchy does support an `ordinal()` method, which returns an `int` value, the enum values cannot be compared directly with `int` values. For this reason, the code does not compile, since the `case` statement values are not compatible with the variable type in the `switch` statement, making Option C the correct answer.

- 10.** C. A local variable is effectively final when its primitive value or object reference does not change after it is initialized, making Option C the correct answer. Option D is incorrect. Any change to the variable after it is initialized disqualifies it for being considered effectively final.
- 11.** D. Both the `Drive` and `Hover` interfaces define a `default` method `getSpeed()` with the same signature. In fact, both `getSpeed()` methods return the same value of 5. The class `Car` implements both interfaces, which means it inherits both `default` methods. Since the compiler does not know which one to choose, the code does not compile, and the answer is Option D. Note that if the `Car` class had overridden the `getSpeed()` method, then the code would have compiled without issue and printed 10 at runtime. In particular, the local class `Racecar` defined in the `main()` method compiles without issue, making Option C incorrect.
- 12.** B. An interface can be extended by another interface and a class can be extended by another class, making the second part of Options A, C, and D incorrect. Option B is correct because an enum cannot be extended. Note that Option C is also incorrect for this reason.
- 13.** B. If the program is called with a single input `WEST`, then `WEST` would be printed at runtime. If the program is called with no input, then the `compass` array would be of size zero, and an `ArrayIndexOutOfBoundsException` would be thrown at runtime. Finally, if the program is called with a string that does not match one of the values in `Direction`, then an `IllegalArgumentException` would be thrown at runtime. The only result not possible is `south`, since the enum value is in uppercase, making Option B the correct answer.
- 14.** B. Enumerated types support creating a set of reusable values whose values are fixed and consistent across the entire application. For these reasons, Options A, C, and D are incorrect. Option B is the false statement because enum values are defined at compile time and cannot be changed or added at runtime.
- 15.** D. The program contains three compilation problems. First off, the enum `Color` extends the enum `Light`, but enums cannot extend other enums so the definition is invalid. Second, the enum value list must end with a semicolon (`;`) if the enum definition contains anything other than the enum values. Since it includes a constructor, a semicolon (`;`) is required after `GREEN`. Finally, enum constructors must be `private`, meaning the `protected` constructor for `Color` does not compile. For these three reasons, Option D is the correct answer.
- 16.** D. Both abstract classes and interfaces can include `static` methods, so Options A and C are incorrect. A static nested class can include `static` methods, but it is the only type of inner class in which this is allowed. Local inner classes, anonymous inner classes, and member inner classes do not support `static` methods. For these reasons, Option D is correct, and Option B is incorrect.
- 17.** B. A functional interface must contain exactly one `abstract` method. The `Bend` interface contains two `abstract` methods, `pump()` and `bend()`, since it extends `Pump` and inherits `pump()`. For this reason, the `Bend` method is not a valid functional interface and therefore cannot be used as a lambda expression, making Option B the correct answer. The rest of the code compiles without issue. Note that the usage of an instance variable to call a `static` method, `r.apply()` in the `main()` method, is permitted but discouraged.
- 18.** C. Applying the `@Override` annotation is optional and not required to override a method or implement an interface method, making Options A and B incorrect. While partially helpful as a form of documentation, it is not the best reason to apply the annotation, making Option D

incorrect. The best reason is that the compiler will actually fail to compile if the method that the `@Override` annotation is being applied to is not actually overriding an inherited method. This behavior helps correct typos or changes in superclasses or interfaces that could break the class or lead to unexpected behavior. For this reason, Option C is the best choice.

19. C. The `Bottle` class includes a static nested class `Ship` that must be instantiated in a static manner. Line `w2` uses an instance of `Bottle` to instantiate the `Ship`. While this would be allowed if `Ship` was a member inner class, since it is a static nested class, line `w2` does not compile, and Option C is the correct answer. Note that if `Ship` was changed to be a member inner class, the code would still not compile since a member inner class cannot include static members and enums are inherently static. Therefore, the correct change would be to fix the declaration on line `w2`.
20. A. Option A is the invalid lambda expression because the type is specified for the variable `j`, but not the variable `k`. The rest of the options are valid lambda expressions. To be a valid lambda expression, the type must be specified for all of the variables, as in Option C, or none of them, as in Options B and D.
21. D. This application declares an anonymous inner class that implements the `Edible` interface. Interface methods are `public`, whereas the override in the anonymous inner class uses package-private access. Since this reduces the visibility of the method, the declaration of `eat()` on line 8 does not compile. Next, the declaration of the `apple` object must end with a semicolon (`;`) on line 11, and it does not. For these two reasons, the code does not compile, and Option D is the correct answer. Note that if these two issues were corrected, with the `public` modifier and missing semi-colon (`;`), then the correct answer would be Option A because the code does not actually call the `eat()` method; it just declares it.
22. A. The code compiles without issue and prints 15, making Option A correct and Option D incorrect. The `main()` method defines a local class `Oak` that correctly extends `Tree`, a static nested class, making Option B incorrect. Finally, the method `getWater()` is permitted to read the variable `water`, defined in the `main()` method, since it is effectively final, having a value of 15 when it is defined. For this reason, Option C is also incorrect.
23. C. Interfaces allow Java to support multiple inheritance because a class may implement any number of interfaces. On the other hand, an anonymous inner class may implement at most one interface, since it does not have a class definition to implement any others. For these reasons, Option C is the correct answer.
24. A. The code does not compile because the declaration of `isDanger()` in the class `SeriousDanger` is an invalid method override. An overridden method may not throw a broader checked exception than it inherits. Since `Exception` is a superclass of `Problem`, thrown by the inherited method in the `Danger` class, the override of this checked exception is invalid. For this reason, line `m1` does not compile, and Option A is the correct answer. The rest of the lines of code compile without issue.
25. B. Options A, C, and D are true statements about interfaces and abstract classes. Option B is the correct answer because neither abstract classes nor interfaces can be marked `final`. For Option D, methods and variables can both be marked `private` in abstract classes in some cases. The “some cases” refers to the fact that the `private` modifier cannot be applied to `abstract` methods, since a method cannot be marked both `final` and `abstract`. Since abstract classes can contain concrete methods, which can take the `private` access modifier, the statement is true.

- 26.** A. The code compiles without issue, so Option C is incorrect. Enum ordinal values are indexed starting with zero, so 0 is printed first. The second line compiles and runs without issue, with `flurry` being converted to `FLURRY`, using the `toUpperCase()` method. Since there is a matching enum named `FLURRY`, that value is printed next. For these reasons, Option A is the correct answer.
- 27.** D. Java was updated to include default interface methods in order to support backward compatibility of interfaces. By adding a `default` method to an existing interface, we can create a new version of the interface, which can be used without breaking the functionality of existing classes that implement an older version of the interface. For this reason, Option D is the correct answer. Options A and C are not applicable to default interface methods, whereas Option B could be achieved by using `static` interface methods.
- 28.** C. The `Penguin` class includes a member inner class `Chick`. Member inner classes cannot include `static` methods or variables. Since the variable `volume` is marked `static`, the member inner class `Chick` does not compile, making Option C the correct answer. Note that the variable `volume` referenced in the `chick()` method is one defined in the `Penguin` outer class. If the `static` modifier was removed from the `volume` variable in the `Chick` class, then the rest of the code would compile and run without issue, printing `Honk(1)!` at runtime.
- 29.** D. Member inner classes require an instance of the surrounding class to be instantiated. Option A is incorrect since we are told that the instantiation request is from a `static` method. Note that this call would be valid from a non-`static` method in `Dinosaur`. Option B is incorrect because it lacks the `new` keyword. Option C is incorrect. `Pterodactyl` is a member inner class, not a static nested class. Option D is correct and uses the instance `dino` to create a new `Pterodactyl` object.
- 30.** C. First off, both `CanBurrow` and `HasHardShell` are functional interfaces since they contain exactly one abstract method, although only the latter uses the optional `@FunctionalInterface` annotation. The declarations of these two interfaces, along with the abstract class `Tortoise`, compile without issue, making Options A and B incorrect. The code does not compile, though, so Option D is incorrect. The class `DesertTortoise` inherits two abstract methods, one from the interface `CanBurrow` and the other from the abstract parent class `Tortoise`. Since the class only implements one of them and the class is concrete, the class declaration of `DesertTortoise` fails to compile on line `k3`, making Option C the correct answer.
- 31.** B. First off, the two interface definitions contain identical methods, with the `public` modifiers assumed in all interfaces methods. For the first statement, the `write()` method is marked `default` in both interfaces, which means a class can only implement both interfaces if the class overrides the `default` method with its own implementation of the method. Since the `Twins` method does override `write()`, the method compiles without issue, making the first statement incorrect. Next, the `publish()` method is marked `static` in both interfaces and the `Twins` class. While having a `static` method in all three is allowed, marking a `static` method with the `@Override` annotation is not because only member methods may be overridden. For this reason, the second statement is correct. Finally, the `think()` method is assumed to be `abstract` in both interfaces since it doesn't have a `static` or `default` modifier and does not define a body. The `think()` method is then correctly overridden with a concrete implementation in the

Twins class, making the third statement incorrect. Since only the second statement was true, Option B is the correct answer.

32. D. An enum and static inner class can define `static` methods, making Option D the correct answer. Options A, B, and C are incorrect because the other types of inner classes cannot define `static` methods. Note that interfaces and abstract classes can define `static` methods.
33. C. First off, Option A does not compile since the variables `p` and `q` are reversed, making the return type of the method and usage of operators invalid. The first argument `p` is a `String` and `q` is an `int`, but the lambda expression reverses them, and the code does not compile. Option B also does not compile. The variable `d` is declared twice, first in the lambda argument list and then in the body of the lambda expression. The second declaration in the body of the lambda expression causes the compiler to generate a duplicate local variable message. Note that other than it being used twice, the expression is valid; the ternary operator is functionally equivalent to the `learn()` method in the `BiologyMaterial` class. Option C is the correct answer since it compiles and handles the input in the same way as the `learn()` method in the `BiologyMaterial` class.
34. C. The code does not compile since it contains two compilation errors, making Option A incorrect. First, the enum list is not terminated with a semicolon (`;`). A semicolon (`;`) is required anytime an enum includes anything beyond just the list of values, such as a constructor or method. Second, the access modifier of `TRUE`'s implementation of `getNickName()` is package-private, but the `abstract` method signature has a `protected` modifier. Since package-private is a more restrictive access than `protected`, the override is invalid and the code does not compile. For these two reasons, Option C is the correct answer. Note that the `@Override` annotation is optional in the method signature, therefore `FALSE`'s version of `getNickName()` compiles without issue. Also, note that the `Proposition` constructor does not include a `private` access modifier, but the constructor compiles without issue. Enum constructors are assumed to be `private` if no access modifier is specified, unlike regular classes where package-private is assumed if no access modifier is specified.
35. A. The code compiles and runs without issue, printing 8 at runtime, making Option A correct and Option D incorrect. The `AddNumbers` interface is a valid functional interface. While it includes both `static` and `default` methods, it only includes one `abstract` method, the precise requirement for it to be considered a functional interface, making Option B incorrect. Finally, Option C is incorrect because the lambda expression is valid and used correctly.
36. A. While this code included a large number of `final` modifiers, none of them prevent the code from compiling when a valid expression is placed in the blank, making Option D incorrect. Option B is incorrect since it returns the `size` variable defined in the `Insert` member inner class, not the `Bottle` class, printing 25 at runtime. Option C is incorrect because the expression is invalid and does not compile when inserted into the blank. Finally, Option A is the correct answer because it compiles, properly references the variable `size` in the `Bottle` class, and prints 14 at runtime.
37. C. The `main()` method attempts to define an anonymous inner class instance but fails to provide the class or interface name, or use the `new` keyword. The right-hand side of the assignment to the `seaTurtle` variable should start with `new CanSwim()`. For this reason, Option C is the correct answer. If the code was corrected with the proper declaration, it would output 7, and Option B would be the correct answer.

- 38.** D. The code does not compile, so Options A and B are incorrect. The declarations of the local inner classes `Robot` and `Transformer` compile without issue. The anonymous inner class that extends `Transformer` compiles without issue, since the `public` variable name is inherited, making Option C incorrect. The only compilation problem in this class is the last line of the `main()` method. The variable name is defined inside the local inner class and not accessible outside class declaration without a reference to the local inner class. Due to scope, this last line of the `main()` method does not compile, making Option D the correct answer. Note that the first part of the `print()` statement in the `main()` method, if the code compiled, would print `GiantRobot`.
- 39.** B. The `Dancer` class compiles without issue, making Option A incorrect. The `SwingDancer` class, though, does not compile because `getPartner()` is an invalid method override. In particular, `Leader` and `Follower` are not covariant since `Follower` is not a subclass of `Leader`. Therefore, line `u2` does not compile, making Option B correct and Option D incorrect. Note that the abstract method `getPartner(int)` is not implemented in `SwingDancer`, but this is valid because `SwingDancer` is an abstract class and is not required to implement all of the inherited abstract methods.
- 40.** C. The code does not compile, so Options A and B are incorrect. The problem here is that the `DEFAULT_VALUE` is an instance variable, not a `static` variable; therefore, the static nested class `GetSet` cannot access it without a reference to the class. For this reason, the declaration of the static nested class `GetSet` does not compile, and Option C is the correct answer. The rest of the code compiles without issue. Note that if the `DEFAULT_VALUE` variable was modified to be `static`, then the code would compile without issue, and Option B would be the correct answer.

## Chapter 13: Generics and Collections

1. C. When declaring a class that uses generics, you must specify a name for the formal type parameter. Java uses the standard rules for naming a variable or class. A question mark is not allowed in a variable name, making I incorrect. While it is common practice to use a single uppercase letter for the type parameter, this is not required. It certainly isn't a good idea to use existing class names like the `News` class being declared here or the `Object` class built into java. However, this is allowed, and Option C is correct.
2. B. Option A is incorrect because the `filter()` method is available on `Stream`, but not `List`. Option C is incorrect because the `replace()` method is available on `List`, but not `Stream`. Option D is tricky because there is a `sort()` method on `List` and a `sorted()` method on `Stream`. These are different method names though, so Option D is incorrect. Option B is the answer because both interfaces have a `forEach()` method.
3. A. Notice how there is unnecessary information in this description. The fact that patrons select books by name is irrelevant. The checkout line is a perfect example of a double-ended queue. We need easy access to one end of the queue for patrons to add themselves to the queue. We also need easy access to the other end of the queue for patrons to get off the queue when it is their turn. The book lookup by ISBN is a lookup by key. We need

a map for this. A `HashMap` is probably better here, but it isn't a choice. So the answer is Option A, which does include both a double-ended queue and a map.

4. B. Java talks about the collections framework, but the `Map` interface does not actually implement the `Collection` interface. `TreeMap` has different methods than `ArrayDeque` and `TreeSet`. It cannot fill in the blank, so Option B is correct.
5. B. Options C and D are incorrect because the method signature is incorrect. Unlike the `equals()` method, the method in `Comparator` takes the type being compared as the parameters when using generics. Option A is a valid `Comparator`. However, it sorts in ascending order by length. Option B is correct. If `s1` is three characters and `s2` is one character, it returns `-2`. The negative value says that `s1` should sort first, which is correct, because we want the longest `String` first.
6. D. `TreeMap` and `TreeSet` keep track of sort order when you insert elements. `TreeMap` sorts the keys and `TreeSet` sorts the objects in the set. This makes Option D correct. Note that you have the option of having `JellyBean` implement `Comparable`, or you can pass a `Comparator` to the constructor of `TreeMap` or `TreeSet`.
7. C. Option A is incorrect because a pipeline still runs if the source doesn't generate any items and the rest of the pipeline is correct. Granted some of the operations have nothing to do, but control still passes to the terminal operation. Option B is incorrect because intermediate operations are optional. Option C is the answer. The terminal operation triggers the pipeline to run.
8. B. The `Iterator` interface uses the `hasNext()` and `next()` methods to iterate. Since there is not a `hasMore()` method, it should be changed to `hasNext()`, making Option B the answer. With respect to Option A, the missing generic type gives a warning, but the code still runs. For Option C, iterators can run as many times as you want, as can the `forEach()` method on `list`.
9. A. First the code creates an `ArrayList` of three elements. Then the list is transformed into a `TreeSet`. Since sets are not allowed to have duplicates, the set only has two elements. Remember that a `TreeSet` is sorted, which means that the first element in the `TreeSet` is 3. Therefore, Option A is correct.
10. C. The word *reduction* is used with streams for a terminal operation, so Options A and B are incorrect. Option D describes a valid terminal operation like `anyMatch()`, but is not a reduction. Option C is correct because a reduction has to look at each element in the stream in order to determine the result.
11. A. The `offer()` method adds an element to the back of the queue. After line 7 completes, the queue contains 18 and 5 in that order. The `push()` method adds an element to the front of the queue. How rude! The element 13 pushes past everyone on the line. After line 8 completes, the queue now contains 13, 18, and 5, in that order. Then we get the first two elements from the front, which are 13 and 18, making Option A correct.
12. D. The `Magazine` class doesn't implement `Comparable<Magazine>`. It happens to implement the `compareTo()` method properly, but it is missing actually writing `implements Comparable`. Since `TreeSet` doesn't look to see if the object can be compared until runtime, this code throws a `ClassCastException` when `TreeSet` calls `add()`, so Option D is correct.

- 13.** C. Line 8 does not compile. `String::new` is a constructor reference. A constructor or method reference is equivalent to a lambda. It participates in deferred execution. When it is executed later, it returns a `String`. It does not return a `String` on line 8. It actually returns a `Supplier<String>`, which cannot be stored in `list`. Since the code does not compile, Option C is correct.
- 14.** B. This code adds two elements to a list. It then gets a stream and iterates through the list, printing two lines. The last line does the same thing again. Since a fresh stream is created, we are allowed to iterate through it, and Option B is correct.
- 15.** D. The `Comic` interface declares a formal type parameter. This means that a class implementing it needs to specify this type. The code on line 21 compiles because the lambda reference supplies the necessary context making Option A incorrect. Option B declares a generic class. While this doesn't tell us the type is `Snoopy`, it punts the problem to the caller of the class. The declaration of `c2` on line 22 compiles because it supplies the type, making Option B incorrect. The code on line 23 compiles because the `SnoopyClass` itself supplies the type making Option C incorrect. Option D has a problem. `SnoopyClass` and `SnoopyComic` appear similar. However, `SnoopyComic` refers to `C`. This type parameter exists in the interface. It isn't available in the class because the class has said it is using `Snoopy` as the type. Since the `SnoopyComic` class itself doesn't compile, the line with `c4` can't instantiate it, and Option D is the answer.
- 16.** A. In streams, the `filter()` method filters out any values that do not match. This means the only value to make it to the terminal operator `count()` is `Chicago`, and Option A is correct.
- 17.** C. When implementing `Comparable`, you implement the `compareTo()` method. Since this is an instance method, it already has a reference to itself and only needs the item it is comparing. Only one parameter is specified, and Option C is correct. By contrast, the `Comparator` interface uses the `compare()` method and the method takes two parameters.
- 18.** C. The source and any intermediate operations are chained and eventually passed to the terminal operation. The terminal operation is where a non-stream result is generated, making Option C correct.
- 19.** A. A constructor reference uses the `new` keyword where a method name would normally go in a method reference. It can implicitly take zero or one parameters just like a method reference. In this case, we have one parameter, which gets passed to the constructor. Option A is correct.
- 20.** D. A custom sort order is specified using a `Comparator` to sort in descending order. However, this `Comparator` is not passed when searching. When a different sort order is used for searching and sorting, the result is undefined. Therefore, Option D is correct.
- 21.** D. Java only allows you to operate on a stream once. The final line of code throws an `IllegalStateException` because the stream has already been used up. Option D is the correct answer.
- 22.** D. The `Wash` class takes a formal type parameter named `T`. Option C shows the best way to call it. This option declares a generic reference type that specifies the type is `String`. It also

uses the diamond syntax to avoid redundantly specifying the type on the right-hand side of the assignment. Options A and B show that you can omit the generic type in the reference and still have the code compile. You do get a compiler warning scolding you for having a raw type. But compiler warnings do not prevent compilation. With the raw type, the compiler treats T as if it is of type Object. That is OK in this example because the only method we call is `toString()` implicitly when printing the value. Since `toString()` is defined on the `Object` class, we are safe, and Options A and B work. Since all three can fill in the blank, Option D is the answer.

23. D. The missing generic type gives a warning, but the code still runs, so Option A is incorrect. The `Iterator` interface uses `hasNext()` and `next()` methods to iterate, so Option B is incorrect. Option C applies to calling the same stream twice. One of our calls is to an `Iterator` anyway, so Option C is incorrect. This code is in fact correct, making the answer Option D.
24. B. This is a `static` method reference. It uses `::` to separate the class name and method name. Option B is correct.
25. B. A source and the terminal operation are required parts of a stream pipeline and must occur exactly once. The intermediate operation is optional. It can appear zero or more times. Since more than once falls within zero or more, Option B is correct.
26. B. `ArrayList` allows `null` elements, making Option B correct. `TreeSet` does not allow `nulls` because they need to compare the values. `ArrayDeque` uses `null` for a special meaning, so it doesn't allow it in the data structure either.
27. D. Option A is the only one of the three options to compile. However, it results in no lines being output since none of the three strings are empty. Options B and C do not even compile because a method reference cannot have an operator next to it. Therefore, Option D is correct.
28. A. Unfortunately you do have to memorize two facts about sort order. First, numbers sort before letters. Second, uppercase sorts before lowercase. Since `TreeMap` orders by key, the first key is 3 and the last is `three`, making Option A correct.
29. C. The `?` is an unbounded wildcard. It is used in variable references but is not allowed in declarations. In a `static` method, the type parameter specified inside the `<>` is used in the rest of the variable declaration. Since it needs an actual name, Options A and B are incorrect. We need to specify a type constraint so we can call the `add()` method. Regardless of whether the type is a class or interface, Java uses the `extends` keyword for generics. Therefore, Option D is incorrect, and Option C is the answer.
30. B. On a stream, the `filter()` method only keeps values matching the lambda. The `removeIf()` does the reverse on a `Collection` and keeps the elements that do not match. In this case, that is `Austin` and `Boston` so Option B is correct.
31. D. The code correctly creates an `ArrayDeque` with three elements. The stream pipeline does compile. However, there is no terminal operation, which means the stream is never evaluated and the output is something like `java.util.stream.ReferencePipeline$2@404b9385`. This is definitely not one of the listed choices, so Option D is correct.

- 32.** C. The `forEach()` method that takes one parameter is defined on the `Collection` interface. However, a map is not a `Collection`. There is a version of `forEach()` defined on the `Map` interface, but it uses two parameters. Since two parameters can't be used with a method reference, Option C is the answer.
- 33.** C. This code is almost correct. Calling two different streams is allowed. The code attempts to use a method reference when calling the `forEach()` method. However, it does not use the right syntax for a method reference. A double colon needs to be used. The code would need to be changed to `System.out::println` to work and print two lines for each call. Since it does not compile, Option C is correct.
- 34.** B. This code shows a proper implementation of `Comparable`. It has the correct method signature. It compares the magazine names in alphabetical order. Remember that uppercase letters sort before lowercase letters. Since `Newsweek` is uppercase, Option B is correct.
- 35.** C. The `filter()` method requires a boolean returned from the lambda or method reference. The `getColor()` method returns a `String` and is not compatible. This causes the code to not compile and Option C to be correct.
- 36.** A. Option A is correct as the source and terminal operation are mandatory parts of a stream pipeline. Option B is incorrect because a `Stream` must return non-primitives. Specialized interfaces like `IntStream` are needed to return primitives. Option C is incorrect because `Stream` has methods such as `of()` and `iterate()` that return a `Stream`. Option D is incorrect because infinite streams are possible.
- 37.** B. The stream pipeline is correct and filters all values out that are 10 characters or smaller. Only `San Francisco` is long enough, so `c` is 1. The `stream()` call creates a new object, so stream operations do not affect the original list. Since the original list is still 3 elements, Option B is correct.
- 38.** B. Options A and C are incorrect because a generic type cannot be assigned to another direct type unless you are using upper or lower bounds in that statement. Now, we just have to decide whether a lower or upper bound is correct for the `T` formal type parameter in `Wash`. The clue is that the method calls `size()`. This method is available on `Collection` and all classes that extend/implement it. Therefore, Option B is correct.
- 39.** C. A `Comparator` takes two parameters, so Options A and B are incorrect. Option D doesn't compile. When using brackets, a `return` keyword and semicolon are required. Option C is a correct implementation.
- 40.** B. Option D is incorrect because there is a `charAt()` instance method. While Option C is correct that the method takes in an `int` parameter, autoboxing would take care of conversion for us if there were no other problems. So Option C is not the answer. Option A is not true because there are constructor and instance method references. Option B is the answer. With method references, only one item can be supplied at runtime. Here, we need either a `String` instance with no parameters in the method or a static method with a single parameter. The `charAt()` method is an instance method with a single parameter so does not meet this requirement.

# Chapter 14: Lambda Built-in Functional Interfaces

1. C. The `Supplier` functional interface does not take any inputs, while the `Consumer` functional interface does not return any data. This behavior extends to the primitive versions of the functional interfaces, making Option C the correct answer. Option A is incorrect because `IntConsumer` takes a value, while `LongSupplier` returns a value. Options B and D are incorrect because `Function` and `UnaryOperator` both take an input and produce a value.
2. A. The `LongSupplier` interface does not take any input, making Option D incorrect. It also uses the method name `getAsLong()`. The rest of the functional interfaces all take a `long` value but vary on the name of the abstract method they use. `LongFunction` contains `apply()` and `LongPredicate` contains `test()`, making Options B and C, respectively, incorrect. That leaves us with `LongConsumer`, which contains `accept()`, making Option A the correct answer.
3. A. The code compiles without issue, so Options C and D are incorrect. The value for distance is 2, which based on the lambda for the `Predicate` will result in a `true` expression, and `Saved` will be printed, making Option A correct.
4. C. Both are functional interfaces in the `java.util.function` package, making Option A true. The major difference between the two is that `Supplier<Double>` takes the generic type `Double`, while the other does not take any generic type and instead uses the primitive `double`. For this reason, Options B and D are true statements. For `Supplier<Double>` in Option B, remember that the returned `double` value can be implicitly cast to `Double`. Option C is the correct answer. Lambdas for `Supplier<Double>` can return a `null` value since `Double` is an object type, while lambdas for `DoubleSupplier` cannot; they can only return primitive `double` values.
5. B. The lambda `(s,p) -> s+p` takes two arguments and returns a value. For this reason, Option A is incorrect because `BiConsumer` does not return any values. Option D is also incorrect, since `Function` only takes one argument and returns a value. This leaves us with Options B and C, which both use `BiFunction`, which takes two generic arguments and returns a generic value. Option C is incorrect because the datatype of the unboxed sum `s+q` is `int` and `int` cannot be autoboxed or implicitly cast to `Double`. Option B is correct. The sum `s+p` is of type `double`, and `double` can be autoboxed to `Double`.
6. C. To begin with, `ToDoubleBiFunction<T,U>` takes two generic inputs and returns a `double` value. Option A is compatible because it takes an `Integer` and `Double` and returns a `Double` value that can be implicitly unboxed to `double`. Option B is compatible because `long` can be implicitly cast to `double`. While we don't know the data types for the input arguments, we know that some values, such as using `Integer` for both, will work. Option C cannot be assigned and is the correct answer because the variable `v` is of type `Object` and `Object` does not have a `length()` method. Finally, Option D is compatible. The variable `y` could be declared `double` in the generic argument to the functional interface, making `y/z` a `double` return value.

7. C. The BiPredicate interface takes two generic arguments and returns a boolean value. Next, DoubleUnaryOperator takes a double argument and returns a double value. Last, ToLongFunction takes a generic argument and returns a long value. That leaves Option C, which is the correct answer. While there is an ObjDoubleConsumer functional interface, which takes a generic argument and a double value and does not return any data, there is no such thing as ObjectDoubleConsumer. Remember that Object is abbreviated to Obj in all functional interfaces in `java.util.function`.
8. C. The code does not compile, so Options A and D are incorrect. The IntUnaryOperator functional interface is not generic, so the argument IntUnaryOperator<Integer> in the `takeTicket()` does not compile, making Option C the correct answer. The lambda expression compiles without issue, making Option B incorrect. If the generic argument <Integer> was dropped from the argument declaration, the class would compile without issue and output 51 at runtime, making Option A the correct answer.
9. A. Option A is the correct answer because BiPredicate takes two generic types and returns a primitive boolean value. Option B is incorrect, since CharSupplier does not exist in `java.util.function`. Option C is also incorrect, since LongFunction takes a primitive long value and returns a generic type. Remember, Java only includes primitive functional interfaces that operate on double, int, or long. Finally, Option D is incorrect. UnaryOperator takes a generic type and returns a generic value.
10. D. First off, the `forEach()` method requires a Consumer instance. Option C can be immediately discarded because Supplier<Double> does not inherit Consumer. For this same reason, Option B is also incorrect. DoubleConsumer does not inherit from Consumer. In this manner, primitive functional interfaces cannot be used in the `forEach()` method. Option A seems correct, since `forEach()` does take a Consumer instance, but it is missing a generic argument. Without the generic argument, the lambda expression does not compile because the expression `p<5` cannot be applied to an Object. The correct functional interface is Consumer<Double>, and since that is not available, Option D is the correct answer.
11. C. BiFunction<Double,Double,Double> and BinaryOperator<Double> both take two Double input arguments and return a Double value, making them equivalent to one another. On the other hand, DoubleFunction<Double> takes a single double value and returns a Double value. For this reason, it is different from the other two, making Option C correct and Option D incorrect.
12. B. BinaryOperator<Long> takes two Long arguments and returns a Long value. For this reason, Option A, which takes one argument, and Option D, which takes two Integer values that do not inherit from Long, are both incorrect. Option C is incorrect because the local variable c is re-declared inside the lambda expression, causing the expression to fail to compile. The correct answer is Option B because `intValue()` can be called on a Long object. The result can then be cast to long, which is autoboxed to Long.
13. C. The program does not compile, so Option A is incorrect. The Supplier functional interface normally takes a generic argument, although generic types are not strictly required since they are removed by the compiler. Therefore, line d1 compiles while triggering a compiler warning, and Options B and D are incorrect. On the other hand, line d2 does cause a compiler error, because the lambda expression does not return a value. Therefore, it is not compatible with Supplier, making Option C the correct answer.

14. A. The input type of a unary function must be compatible with the return type. By compatible, we mean identical or able to be implicitly cast. For this reason, Option A is the correct answer. Option B is incorrect since all of the `UnaryOperator` functional interfaces, generic or primitive, take exactly one value. Option C is incorrect because the primitive functional interfaces do not take a generic argument. Finally, Option D is incorrect. For example, the generic `UnaryOperator<T>` returns an `Object` that matches the generic type.
15. C. Remember that all `Supplier` interfaces take zero parameters. For this reason, the third value in the table is `0`, making Options A and B incorrect. Next, `DoubleConsumer` and `IntFunction` each take one value, `double` and `int`, respectively. On the other hand, `ObjDoubleConsumer` takes two values, a generic value and a `double`, and returns `void`. For this reason, Option C is correct, and Option D is incorrect.
16. D. All `Consumer` functional interfaces have a `void` return type. For this reason, the first and last values in the table are both `void`, making Options A and B incorrect. `IntFunction` takes an `int` and returns a generic value, while `LongSupplier` does not take any values and returns a `long` value. For this reason, Option C is incorrect, and Option D is correct.
17. B. The `removeIf()` method requires a `Predicate` since it operates on a boolean result, making Option A incorrect. The `forEach()` method takes a `Consumer` and does not return any data, making Option B correct, and Options C and D incorrect.
18. C. The code does not compile, so Option A is incorrect. The lambda expression compiles without issue, making Option B incorrect. The task variable is of type `UnaryOperator<Doll>`, with the abstract method `apply()`. There is no `accept()` method defined on that interface, therefore the code does not compile, and Option C is the correct answer. If the code was corrected to use the `apply()` method, the rest of it would compile without issue. At runtime, it would then produce an infinite loop. On each iteration of the loop, a new `Doll` instance would be created with 5, since the post-decrement (--) operator returns the original value of the variable, and that would make Option D the correct answer.
19. C. To begin with, `Consumer` uses `accept()`, making Option A incorrect. Next, `Function` and `UnaryOperator` use `apply()`, making Options B and D, respectively, incorrect. Finally, `Supplier` uses `get()`, making Option C the correct answer.
20. D. First off, Options A and B are incorrect because the second functions for both return a `double` or `Double` value, respectively. Neither of these values can be sent to a `UnaryOperator<Integer>` without an explicit cast. Next, Option C is incorrect. The first functional interface `Function<Double, Integer>` takes only one input, but the diagram shows two inputs for the first functional interface. That leaves us with Option D. The first functional interface `BiFunction<Integer, Double, Integer>` takes an `int`, which can be implicitly autoboxed to `Integer`, and a `Double` and returns an `Integer`. The next functional interface, `BinaryOperator<Integer>`, takes two `Integer` values and returns an `Integer` value. Finally, this `Integer` value can be implicitly unboxed and sent to `IntUnaryOperator`, returning an `int`. Since these behaviors match our diagram, Option D is the correct answer.

- 21.** D. Options A, B, and C are true statements about functional interfaces. A lambda may be compatible with multiple functional interfaces, but it must be assigned to a functional interface when it is declared or passed as a method argument. Also, a method can be created with the return type that matches a functional interface, allowing a lambda expression to be returned. Option D is the correct answer. Deferred execution means the lambda expression is not evaluated until runtime, but it is compiled. Compiler errors in the lambda expression will prevent the code from compiling.
- 22.** B. Option A is incorrect because the `String "3"` is not compatible with the return type `int` required for `IntSupplier`. Option B is the correct answer. Although this will result in a divide by zero issue at runtime, the lambda is valid and compatible with `IntSupplier`. Option C is incorrect because the lambda expression is invalid. The `return` statement is only allowed inside a set of brackets `{}`. Finally, Option D is incorrect. The method reference is used for `Supplier`, not `Consumer`, since it takes a value and does not return anything.
- 23.** C. The lambda expression is invalid because the input argument is of type `Boss`, and `Boss` does not define an `equalsIgnoreCase()` method, making Option C the correct answer. If the lambda was corrected to use `s.getName()` instead of `s`, the code would compile and run without issue, printing `[JENNY, GRACE]` at runtime and making Option A the correct answer.
- 24.** D. First of all, `Consumer<Object>` takes a single `Object` argument and does not return any data. The classes `ArrayList` and `String` do not contain constructors that take an `Object`, so neither of the first two statements are correct. The third statement does support an `Object` variable, since the `System.out.println(Object)` method exists. For these reasons, Option D is the correct answer.
- 25.** B. The `java.util.function` package does not include any functional interfaces that operate on the primitive `float`, making Option A incorrect. Remember, Java only includes primitive functional interfaces that operate on `double`, `int`, or `long`. Option B is correct because it is a valid functional interface. Option C is incorrect because there is no `UnaryIntOperator` functional interface. Note that there is one called `IntUnaryOperator`. Option D is incorrect. The `java.util.function` package does not include any tri- operators, although many are easy to write.
- 26.** D. A lambda expression can match multiple functional interfaces. It matches `DoubleUnaryOperator`, which takes a `double` value and returns a `double` value. Note that the data type of `s+1` is `double` because one of the operands, in this case `s`, is `double`. It also matches `Function<String, String>` since the `(+)` operator can be used for `String` concatenation. Finally, it matches `IntToLongFunction` since the `int` value `s+1` can be implicitly cast to `long`. On the other hand, the lambda expression is not compatible with `UnaryOperator` without a generic type. When `UnaryOperator` is used without a generic argument, the type is assumed to be `Object`. Since the `(+)` operator is not defined on `Object`, the code does not compile due to the lambda expression body, making Option D the correct answer. Note that if the lambda expression did not rely on the `(+)` operator, such as `s -> s`, then `UnaryOperator` would be allowed by the compiler, even without a generic type.
- 27.** B. The `BiFunction` interface takes two different generic values and returns a generic value, taking a total of three generic arguments. Next, `ToDoubleFunction` takes exactly one

generic value and returns a double value, requiring one generic argument. The `ToIntBiFunction` interface takes two generic values and returns an `int` value, for a total of two generic arguments. For these reasons, Options A, C, and D are incorrect. The correct answer is Option B. `DoubleFunction` takes a double value and returns a generic result, taking exactly one generic argument, not two.

28. D. While lambda expressions can use primitive types as arguments, the functional interface in this class uses the wrapper classes, which are not compatible. For this reason, Option A is incorrect. Option B is also incorrect, since the number of arguments and return type does not match the functional interface. Furthermore, the method reference `System.out::print` on the right-hand side of the lambda expression is invalid here, since it returns a method reference, not a double value. Option C is incorrect because `2*w` is of type `double`, which cannot be returned as an `Integer` without an explicit cast. Option D is the correct answer. It takes exactly two arguments because the return value `int` can be implicitly autoboxed to `Integer`.
29. A. `BooleanSupplier` is the only functional interface that does not involve `double`, `int`, or `long`, making Option A the correct answer. The rest of the functional interfaces are not found in `java.util.function`. Java does not have built-in support for primitive functional interfaces that include `char`, `float`, or `short`.
30. D. The code does not compile because the lambda expression `p -> p*100` is not compatible with the `DoubleToIntFunction` functional interface. The input to the functional interface is `double`, meaning `p*100` is also `double`. The functional interface requires a return value of `int`, and since `double` cannot be implicitly cast to `int`, the code does not compile, making Option D the correct answer. If the correct cast was applied to make `(p*100)` an `int`, then the rest of the class would compile and 250 would be printed at runtime, making Option B correct.
31. B. The `ToDoubleFunction` interface takes a generic value, not a `double` value, making Option D incorrect. It also uses the method name `accept()`. The rest of the functional interfaces all take a `double` value. `DoubleConsumer` contains the `accept()` method, making Option A incorrect. `DoublePredicate` contains the `test()` method, making Option B the correct answer. Finally, `DoubleUnaryOperator` contains the `applyAsDouble()` method, making Option C incorrect.
32. D. To start with, line 5 does not compile because `Function` takes two generic arguments, not one. Second, the assignment statement on line 7 does not end with a semicolon (`;`), so it also does not compile. Finally, the `forEach()` method on line 10 requires a `Consumer`, not a `Function`, so this line does not compile. For these three reasons, Option D is the correct answer.
33. D. The `DoubleToLongFunction` interface takes a `double` argument and returns a `long` value. Option A is compatible since the `int` value `1` can be implicitly cast to `long`, and `2L` is already a `long`. Option B is also compatible, since the `double` value `10.0*e` is explicitly cast to `int` then implicitly cast to `long`. Next, Option C is compatible because an explicit cast of the `double` to a `long` value is used. Option D cannot be assigned and is the correct answer. Although the `Double` class does have a `longValue()` method, the left-hand side of the lambda expression must use the primitive `double`, not the wrapper `Double`. This lambda expression violates the signature of the functional interface, since it allows `Double` values to be sent to the interface, including those that could be `null`.

- 34.** C. The DoublePredicate interface takes a double value and returns a boolean value. LongUnaryOperator takes a long value and returns a long value.ToIntBiFunction takes two generic values and returns an int value. The only choice that is not an existing functional interface is ShortSupplier. Recall that Java only includes primitive functional interfaces that operate on double, int, or long. For this reason, Option C is the correct answer.
- 35.** A. The method reference System.out::println takes a single input and does not return any data. Consumer<Sheep> is compatible with this behavior, making Option A the correct answer and Option D incorrect. Option B is incorrect because void cannot be used as a generic argument. Option C is incorrect since System.out::println() does not return any data and UnaryOperator requires a return value.
- 36.** C. The code does not compile, making Options A and B incorrect. The local variable MAX\_LENGTH is neither final nor effectively final, meaning it cannot be used inside the lambda expression. The MAX\_LENGTH variable starts off with an initial value of 2, but then is modified with the increment assignment (+=) operator to a value of 5, disqualifying its ability to be considered effectively final by the compiler. Since the lambda does not compile, Option C is the correct answer. If the code was rewritten so that the MAX\_LENGTH variable was marked final and assigned a value of 5 from the start, then it would output 2, and Option A would be correct.
- 37.** B. To begin with, all of the functional interfaces in the list of choices take two values. The difference is in the name of the method they use. BiConsumer uses accept(), making Option A incorrect. Option B is correct because BiFunction includes the apply() method. Option C is incorrect, since BiPredicate uses the test() method. DoubleBinaryOperator is almost correct but the name of the method is applyAsDouble(), not apply(), making Option D incorrect. For the exam, you should be aware of which primitive functional interfaces use a different method name than the generic ones.
- 38.** B. To start with, IntFunction<Integer> takes an int value and returns an Integer. The first statement uses Integer instead of int as the input argument and is therefore not compatible. The second statement is compatible, since the return type null can be used as an Integer return type. The last statement is also valid. An int can be autoboxed to Integer. For these reasons, Option B is the correct answer.
- 39.** C. The primitive Supplier functional interfaces, such as BooleanSupplier and LongSupplier, do not have a get() method. Instead, they have methods such as getAsBoolean() and getAsLong(), respectively. For this reason, the first line of the checkInventory() method does not compile, making Option C the correct answer. If the method call was changed to getAsBoolean(), then the rest of the code would compile without issue, print Plenty! at runtime, and Option A would be the correct answer.
- 40.** B. Java only supports a single return data type or void. Therefore, it is not possible to define a functional interface that returns two data types, making Option A incorrect. Although Java does not include built-in support for primitive functional interfaces that include float, char, or short, there is nothing to prevent a developer from creating them in their own project, making Option B the true statement and the correct answer. Option C is incorrect because a functional interface that takes no values and returns void is possible. In fact, Runnable is one such example. Option D is also incorrect, since IntFunction<R> takes a primitive argument as input and a generic argument for the return type.

# Chapter 15: Java Stream API

1. D. Option A is incorrect because it doesn't print out one line. The `peek()` method is an intermediate operation. Since there is no terminal operation, the stream pipeline is not executed, so the `peek()` method is never executed. Options B and C are incorrect because they correctly output one line using a method reference and lambda, respectively, and don't use any bad practices. Option D is the answer. It does output one line. However, it is bad practice to have a `peek()` method that has side effects like modifying a variable.
2. A. This code generates an infinite stream of integers: 1, 2, 3, 4, 5, 6, 7, etc. The `Predicate` checks if the element is greater than 5. With `anyMatch()`, the stream pipeline ends once element 6 is hit and the code prints `true`. For both the `allMatch()` and `noneMatch()` operators, they see that the first element in the stream does not match and the code prints `false`. Therefore, Option A is correct.
3. B. Only the `average()` method returns an `OptionalDouble`. This reflects that it doesn't make sense to calculate an average when you don't have any numbers. By contrast, counting without any numbers gives the `long` number `0` and summing gives the `double` number `0.0`. Since only one method matches the return type, Option B is correct.
4. C. The `map()` method can fill in the blank. The lambda converts a `String` to an `int` and Java uses autoboxing to turn that into an `Integer`. The `mapToInt()` method can also fill in the blank and Java doesn't even need to autobox. There isn't a `mapToObj()` in the stream API. Note there is a similarly named `mapToObj()` method on `IntStream`. Since both `map()` and `mapToInt()` work here, Option C is correct.
5. D. The `average()` method returns an `OptionalDouble`. This interface has a `getAsDouble()` method rather than a `get()` method, so the code does compile. However, the stream is empty, so the optional is also empty. When trying to get the value, the code throws a `NoSuchElementException`, making Option D correct.
6. D. Option A is incorrect because `anyMatch()` returns a `boolean`. Option B is incorrect because `filter()` is an intermediate operation, not a terminal operation, and therefore returns a `Stream`. Both of these methods do take a `Predicate` as a parameter. While `findAny()` does return an `Optional`, it doesn't take any parameters. Therefore, Option C is incorrect, and Option D is the answer.
7. B. This code builds a list with two elements. It then uses that list as a source for the stream, sorts the stream as it goes by, and grabs the first sorted element. This does not change the original list. The first element in the sorted stream is `1.2`, but the first element of `list` remains as `5.4`. This makes Option B correct.
8. B. Primitive streams, like `LongStream`, declare an `average()` method, while summary statistics classes, like `LongSummaryStatistics`, declare a `getAverage()` method, making Options C and D incorrect. The `average()` method returns an `OptionalDouble` object, which declares a `getAsDouble()` method rather than a `get()` method. Therefore, Option A is incorrect, and Option B is correct.

- 9.** B. Since the result of the `collect()` is not stored in a variable or used in any way, all the code needs to do is compile. There is no `Collectors.toArrayList()` method. If you want to specify an `ArrayList`, you can call `Collectors.toCollection(ArrayList::new)`. The `Collectors.toList()` method does in fact exist and compile. While there is a `Collectors.toMap()` method, it requires two parameters to specify the key and value functions, respectively. Since only one can compile, Option B is correct.
- 10.** C. As tempting as it is, you can't actually convert a `Map` into a `Stream` directly, which means you can't call the `map()` method on it either. However, you can build a `Stream` out of the keys or values or key/value pairs. Since this code doesn't compile, Option C is correct.
- 11.** D. I is incorrect because `isPresent()` returns `false` for an empty `Optional`. II is incorrect because `of()` throws a `NullPointerException` if you try to pass a null reference. III doesn't throw an exception as the `ofNullable()` is designed to allow a null reference. However, it returns `false` because no value is present. Since none of the choices are correct, Option D is the answer.
- 12.** A. This code does compile. Remember that imports are implied, including the `static` import for `Collectors`. The collector tries to use the number of characters in each stream element as the key in a map. This works fine for the first two elements, `speak` and `bark`, because they are of length 5 and 4, respectively. When it gets to `meow`, it sees another key of 4. The `merge` function says to use the first one, so it chooses `bark` for the value. Similarly, `growl` is 5 characters, but the first value of `speak` is used. There are only two distinct lengths, so Option A is correct.
- 13.** C. For the primitive stream that contains the `int` primitives, the interface names are incorrect. They should be `IntStream` and `IntSummaryStatistics`, making Option C correct. If this was fixed, Option B would be the answer.
- 14.** B. This code does compile. As an intermediate operation, you are allowed to call `peek()` many times in a stream pipeline. You can even call it multiple times in a row. While it is common to write `System.out::println` directly as a parameter to `peek()`, nothing prevents you from creating a `Consumer` variable. Since the `forEach()` method also takes a `Consumer`, we can reuse it. The three `peek()` intermediate operations and one `forEach()` operation total four lines of output. The `map()` operation could be omitted since it simply passes the input through.
- 15.** B. Character objects are allowed in a `Stream`, so line `z1` compiles, making Option C incorrect. Line `z2` also compiles since `findAny()` returns an `Optional` and `ifPresent()` is declared on `Optional`. Therefore, Option D is also incorrect. Now let's look at the `Stream`. The source has three elements. The intermediate operation sorts the elements and then we request one from `findAny()`. The `findAny()` method is not guaranteed to return a specific element. Since we are not using parallelization, it is highly likely that the code will print `a`. However, you need to know this is not guaranteed, making Option B the answer.
- 16.** A. The `sorted()` method takes an optional `Comparator` as the parameter, which takes two `String` parameters and returns an `int`. Option A is correct because the lambda implements this interface. Option B is incorrect because the method reference doesn't take any parameters, nor does it return an `int`.

17. D. The `Optional` class has an `isPresent()` method that doesn't take any parameters. It returns a `boolean` and is commonly used in `if` statements. There is also an `ifPresent()` method that takes a `Consumer` parameter and runs it only if the `Optional` is non-empty. The methods `isNotNull()` and `forEach()` are not declared in `Optional`. Therefore, Option D is correct.
18. C. The first intermediate operation, `limit(1)`, turns the infinite stream into a stream with one element: `true`. The `partitioningBy()` method returns a map with two keys, `true` and `false`, regardless of whether any elements actually match. If there are no matches, the value is an empty list, making Option C correct.
19. B. The `flatMap()` method is used to turn a stream of streams into a one-dimensional stream. This means it gets rid of the empty list and flattens the other two. Option A is incorrect because this is the output you'd get using the regular `map()` method. Option B is correct because it flattens the elements. Notice how it doesn't matter that all three elements are different types of `Collection` implementations.
20. D. The `sorted()` method allows an optional `Comparator` to be passed as a reference. However, `Comparator.reverseOrder()` does not implement the `Comparator` interface. It takes zero parameters instead of the required two. Since it cannot be used as a method reference, the code does not compile, and Option D is correct.
21. D. Option A is incorrect because the `findAny()` might not return 1. The result could be any of the three numbers. Option B is incorrect because there is no `first()` method available as a terminal operation. Option C is tempting because there is a `min()` method. However, since we are working with a `Stream`, this method requires a `Comparator` as a parameter. Therefore, Option D is the answer.
22. C. `List` doesn't have a `filter()` method, so Option A is incorrect. `Stream` does have `filter()` and `map()` methods. However, `Stream` doesn't have an `ifPresent()` method. This makes IV incorrect, so Options B and D are incorrect. Both `Collection` and `String` have an `isEmpty()` method, so either can be used with the `Optional`, making Option C the answer.
23. D. This code generates an infinite stream of the number 1. The `Predicate` checks if the element is greater than 5. This will never be true. With `allMatch()`, the stream pipeline ends after checking the first element. It doesn't match, so the code prints `false`. Both `anyMatch()` and `noneMatch()` keep checking and don't find any matches. However, they don't know if a future stream element will be different, so the code executes infinitely until the process is terminated. Therefore, Option D is correct.
24. D. Both `Collectors.groupingBy()` and `Collectors.partitioningBy()` are useful for turning a stream into a `Map`. The other two methods do not exist. However, when using a condition, you should use `partitioningBy()` as it automatically groups using a `Boolean` key. Therefore, Option D is correct.
25. B. Option A is incorrect because we are working with primitives rather than objects. Option C compiles but outputs the stream references rather than the contents. Option B is correct because it flattens the `int` primitives into one stream.

- 26.** D. The summary statistics classes provide getters in order to access the data. The `getAverage()` method returns a `double` and not an `OptionalDouble`. Option D is the only option to compile.
- 27.** D. Option A doesn't compile because the `get()` method on `Optional` doesn't take any parameters. Options B and C do compile, but both print `Cupcake` since the `Optional` is not empty. Therefore, Option D is correct.
- 28.** C. The first line generates an infinite stream. The stream pipeline has a filter that lets all these elements through. Since `sorted()` requires all the elements be available to sort, it never completes, making Option C correct.
- 29.** A. The `mapToDouble()` method compiles. However, it converts 9 into 9.0 rather than the single digit 9. The `mapToInt()` method does not compile because a `long` cannot be converted into an `int` without casting. The `mapToLong()` method is not available on `LongStream` so it does not compile. It is available on `DoubleStream`, `IntStream`, and `Stream` implementations. Since none of the options outputs the single digit 9, Option A is correct.
- 30.** A. The `filter()` method either passes along a given element or doesn't, making Option D incorrect. The `flatMap()` method doesn't pass along any elements for empty streams. For non-empty streams, it flattens the elements, allowing it to return zero or more elements. This makes Option B incorrect. Finally, the `map()` method applies a one-to-one function for each element. It has to return exactly one element, so Option A is the correct answer.
- 31.** D. First, we sort the stream. Option B is incorrect because `findFirst()` is guaranteed to return the first element. However, the `findFirst()` method returns an `Optional`. Therefore, the output of this code is `Optional[a]` rather than the letter a, making Option D correct.
- 32.** C. There is not a stream pipeline method called `sort()`. There is one called `sorted()`. Since the code does not compile, Option C is correct. If this was fixed, Option A would be correct since the `Comparator` sorts in ascending order.
- 33.** B. This code compiles. It creates a stream of `Ballot` objects. Then it creates a map with the contestant's name as the key and the sum of the scores as the value. For Mario, this is 10 + 9, or 19, so Option B is correct.
- 34.** D. Both `anyMatch()` and `allMatch()` take a `Predicate` as a parameter. This code does not compile because the parameter is missing.
- 35.** D. The `flatMap()` method works with streams rather than collections. The code does not compile because the `x` is not a stream, making Option D correct. If this was fixed, Option B would be the answer.
- 36.** C. The `groupingBy()` collector always returns a `Map` (or a specific implementation class of `Map`), so III can't be right. The other two are definitely possible. To get I, you can group using a `Function` that returns an `Integer` such as `s.collect(Collectors.groupingBy(String::length))`. To get II, you need to group using a `Function` that returns a `Boolean` and specify the type, such as `s.collect(Collectors.groupingBy(String::isEmpty, Collectors.toSet()))`. Notice that autoboxing is used for both. Therefore, Option C is correct.

37. D. There is no built-in method to map a value to a boolean primitive. Therefore, Options B and C don't even compile, so they are incorrect. Option A does compile as it maps a `Runner` to a `Boolean`. However, it doesn't actually `filter()` the stream to eliminate any values, so the output is not the same. It prints 3 instead of 1. None of these are correct, making Option D the answer.
38. A. Option A is the answer because there is a `getCount()` method that returns a `long` rather than a method named `getCountAsLong()`. Option B is incorrect because there is in fact a `getMax()` method. Option C is incorrect because `toString()` is declared on `Object`, which means it is inherited by all classes.
39. C. The `main()` method has warnings, but it does compile, making Option D incorrect. The warnings are both about not declaring the generic type for `Optional`. Option A does not compile because the `orElse()` method expects an `Exception` as the alternate value passed as a parameter. `IllegalArgumentException::new` is a `Supplier` instead. Options B and C both compile as both methods expect a `Supplier` as the parameter. However, `orElseGet()` simply returns the exception from the method rather than throwing it. Option C actually throws the exception the `Supplier` created and is the correct answer.
40. B. Option A happens to output the same result for both pairs. It outputs a blank line in `withFlatMap()` because empty streams are removed and in `withoutFlatMap()` because the `filter()` method removes the empty list. Option B outputs different results. The `withFlatMap()` method outputs `lastall queued up` since it flattens the streams. By contrast, the `withoutFlatMap()` method outputs `[last, all queued up]` since it leaves the structure intact. Since the output is different. Option B produces different results so it is the answer.

## Chapter 16: Exceptions and Assertions

1. D. If no exception is thrown, then the `catch` block will not be executed. The `try` block is always visited first, followed by the `finally` block, which is guaranteed to execute regardless of whether an exception is thrown. For these reasons, Option D is the correct answer, with the statements in the correct order.
2. C. Unlike a `try-with-resources` statement, in which the `catch` and `finally` blocks are optional, a `try` statement requires a `catch` or `finally` block to be used, or both. For this reason, Option C is the correct answer.
3. D. The code does not compile because the `throw` keyword is incorrectly used in the `toss()` method declaration. The keyword `throws` should have been used instead. For this reason, Option D is the correct answer. Since `LostBallException` inherits `Throwable` and the `main()` method handles `Throwable`, `LostBallException` is handled by the `main()` method, making Option B incorrect. Option C is also incorrect because `ArrayStoreException` is an unchecked exception that extends `RuntimeException` and is not required to be handled or declared. Finally, if `throws` was used instead of `throw`, the entire application would compile without issue and print `Caught!`, making Option A the correct answer.

4. A. The only symbol permitted to separate exception types in a multi-catch statement is a single pipe character (|). For this reason, Option A is correct.
5. D. In Java, `assert` is a keyword, meaning it cannot be used as a variable, class, or method name. For this reason, line 5 does not compile. Line 6 also does not compile because the `assert` statement is not a method and does not support parentheses around both expressions. Because neither of these lines compile, Option D is the correct answer.
6. C. First off, `Error` is an unchecked exception. It is recommended that `Error` not be caught by most application processes, making Option A incorrect. `IllegalStateException` inherits `RuntimeException`, both of which are unchecked, making Options B and D, respectively, incorrect. Option C is correct because `ParseException` must be handled or declared.
7. D. The `Exception` class contains multiple constructors, including one that takes `Throwable`, one that takes `String`, and a no-argument constructor. The first `WhaleSharkException` constructor compiles, using the `Exception` constructor that takes a `String`. The second `WhaleSharkException` constructor also compiles. The two statements, `super()` and `new Exception()`, actually call the same constructor in the `Exception` class, one after another. The last `WhaleSharkException` compiles with the compiler inserting the default no-argument constructor `super()`, because it exists in the `Exception` class. For these reasons, all of the constructors compile, and Option D is the correct answer.
8. B. The `UnsupportedOperationException` class is an unchecked exception, which means it inherits from `RuntimeException`. While `Error` also is an unchecked exception, the diagram indicates that the class does not inherit from `Error`. There is only one class between `Throwable` and `RuntimeException` in the diagram, and since we know `RuntimeException` inherits `Exception`, the other missing class must be `Exception`. For these reasons, Option B is the correct answer.
9. C. The code does not compile because the variable `b` is used twice in the `main()` method, both in the method declaration and in the catch block, making Option C the correct answer. If a different variable name was used in one of the locations, the program would print one line, `complete`, making Option A the correct answer. Note that while an exception is created inside the `turnOn()` method, it is not thrown.
10. D. First off, unless assertions are enabled at runtime, no assertion statement guarantees an assertion will be thrown at runtime, making Option D the correct answer. Next, Option A does not compile because the `assert` statement is not a method and does not take arguments in parentheses. It's also invalid because it requires a boolean expression for the first expression, not a numeric one. An additional value can be specified, but it requires a colon separator (:). Option B would be the correct answer and trigger an error if assertions are enabled, since `0==1` evaluates to `false`. Option C is incorrect. Even if assertions were enabled, it would not trigger an error since `0==0` evaluates to `true`.
11. C. The class does not compile because in line `r2`, brackets `{}` are used instead of parentheses `()` in the `try-with-resources` statement, making Option C the correct answer. If this line

was fixed to use parentheses (), then the rest of the class would compile without issue and print This just in! at runtime, making Option A the correct answer.

12. C. When both a `try` block and `close()` method throw exceptions, the one in the `try` block is the primary, while the one in the `close()` method is suppressed. For this reason, Option A is a true statement. Option B is also a true statement, since a `catch` block is not required when using a `try-with-resources` statement. Option C is the correct answer, since resources are closed in reverse order in which they are created, not the other way around. Option D is a true statement because multiple resources can be declared within a single set of parentheses, each separated by a semicolon (;).
13. A. The program compiles without issue, so Option D is incorrect. The narrower `SpellingException` and `NullPointerException`, which inherit from `Exception`, are correctly presented in the first `catch` block, with the broader `Exception` being in the second `catch` block. The `if-then` statement evaluates to true, and a new `SpellingException` instance is created, but it is not thrown because it is missing the `throw` keyword. For this reason, the `try` block ends without any of the `catch` blocks being executed. The `finally` block is then called, making it the only section of code in the program that prints a line of text. For this reason, Option A is the correct answer.
14. C. First off, the `try` block is capable of throwing two checked exceptions, `MissingMoneyException` and `MissingFoodException`. The `catch` block uses the `Exception` class to handle this, since both have `Exception` as a supertype. It then rethrows the `Exception`. For this reason, `Exception` would be appropriate in the blank, making the first statement correct. The compiler is also smart enough to know that there are only two possible subclasses of `Exception` that can actually be thrown in the `main()` method, so declaring `MissingMoneyException` and `MissingFoodException` together also allows the code to compile, making the third statement correct. The second statement, only inserting `MissingMoneyException`, would not allow the code to compile because the `main()` method could throw a checked `MissingFoodException` that was not handled. For these reasons, Option C is the correct answer.
15. C. First off, `Closeable` extends `AutoCloseable`, making Option A incorrect. The difference between the two is that the `close()` method in `AutoCloseable` throws `Exception`, while the `close()` method in `Closeable` throws `IOException`, making Option D incorrect. Since `IOException` is a subclass of `Exception`, both `close()` methods can throw an `IOException`, making Option B incorrect. On the other hand, `Exception` is not a subclass of `IOException`. For this reason, the `close()` method in a class that implements `Closeable` cannot throw an instance of the `Exception` class, because it is an invalid override using a broader exception type, making Option C the correct answer.
16. B. Option A does not compile because a multi-catch expression uses a single variable, not two variables. Option C does not compile because it is not possible to throw this checked `IOException` in the `try` block. Option D does not compile because multi-catch blocks cannot contain two exceptions in which one is a subclass of the other. If it did, one of the two exceptions would be redundant. Option B is the correct answer and the only expression that allows the class to compile. While both exceptions in the multi-catch block inherit from `Exception`, neither is a subclass of the other.

- 17.** C. First off, the order of exceptions in a multi-catch does not matter, only that they not be subclasses of one another, making Options A and B incorrect. Option C is the correct answer because a multi-catch variable is effectively final. Java forbids reassigning of multi-catch variables since it is unclear what the precise reference type is. Option D is incorrect because a multi-catch with a single exception type is just a regular catch block. A regular catch variable is not required to be effectively final and can be reassigned within the catch block.
- 18.** D. The code does not compile, so Option A is incorrect. The first compilation error is that `Shelf` does not implement `AutoCloseable`, meaning a `try-with-resources` statement cannot be used. Even though `Shelf` does implement `Closing`, an interface that uses the same abstract method signature as `AutoCloseable`, the JVM requires `AutoCloseable` be implemented to use `try-with-resources`. The second compilation problem is that `throws` is used instead of `throw` inside the `try` block. Remember that `throws` is only used in method signatures. The third compilation issue is that the order of exceptions in the two catch blocks are reversed. Since `Exception` will catch all `IllegalArgumentException` instances, the second catch block is unreachable. The final compilation error is that the `shelf` variable is used in the `finally` block, which is out of scope. Remember that the scope of `try-with-resources` variables ends when the `try` statement is complete. For these four reasons, Option D is the correct answer.
- 19.** A. Option A is the correct answer. Any `catch` or `finally` blocks used with a `try-with-resources` statement are executed after the declared resources have been closed, not before. Options B and C are true statements, since `Closeable` extends `AutoCloseable` and the requirement for `try-with-resources` is that they must be of type `AutoCloseable`. Finally, Option D is a true statement and one of the primary motivations for using `try-with-resources` statements.
- 20.** D. The optional second parameter of an `assert` statement, when used, must return a value. The second `assert` statement uses `System.out.print()` as its second parameter, which has a return type of `void`. For this reason, the code does not compile, making Option D the correct answer. Other than this one line, the rest of the class compiles without issue.
- 21.** D. Only one of the classes, `MissingResourceException`, inherits from the unchecked `RuntimeException` class, making Option D the correct answer. In fact, `IOException` and `SQLException` extend the checked `Exception` class directly. The `NotSerializableException` is also checked, since it is a subclass of `IOException` via `ObjectStreamException`.
- 22.** D. The code does not compile, making Options A and B incorrect. The declaration of `weatherTracker` uses an anonymous inner class that correctly overrides the `close()` method. Remember that overridden methods cannot throw any new or broader checked exceptions than the inherited method. Alternatively, they can avoid throwing inherited checked exceptions or declare new unchecked exceptions, such as `RuntimeException`. The compilation error is in the `catch` block of the `main()` method, where the `weatherTracker` variable is out of scope. In `try-with-resources` statements, the resources are only accessible in the `try` block. For this reason, Option D is the correct answer.

23. A. Asserts can be enabled by using the command-line options `-ea` or `-enableassertions` and disabled by using `-da` or `-disableassertions`. Passing `-di` does not enable or disable assertions, making Option A the correct answer.
24. A. The application compiles without issue and prints Hello, making Option A the correct answer. The `ReadSign` and `MakeSign` classes are both correctly implemented, with both overridden versions of `close()` dropping the checked exception. The `try-with-resources` statement is also correctly implemented for two resources and does not cause any compilation errors or runtime exceptions. Note that the semicolon (`;`) after the second resource declaration is optional.
25. B. The code compiles, so Option D is incorrect. The order of evaluation for a `try-with-resources` statement is that the resources are closed before any associated `catch` or `finally` blocks are executed. For this reason, 2 is printed first, followed by 1. The `ArithmaticException` is then caught and 3 is printed. The last value printed is 4, since the `finally` block runs at the end. For these reasons, Option B is the correct answer.
26. B. First off, Option A is an incorrect statement because the `AutoCloseable` interface does not define a default implementation of `close()`. Next, the `close()` method should be idempotent, which means it is able to be run multiple times without triggering any side effects. For this reason, Option B is correct. After being run once, future calls to `close()` should not change any data. Option C is incorrect because the `close()` method is fully capable of throwing exceptions. In fact, the signature of the method in `AutoCloseable` throws a checked `Exception`, although classes that override it may choose to drop the checked exception. Option D is incorrect because the return type of `close()` is `void`, which means no return value can be returned.
27. D. The `play()` method compiles without issue, rethrowing a wrapped exception in the `catch` block. While the `main()` method does declare `RuntimeException`, it does not declare or catch the `Exception` thrown by the calls to `play()`. Even though the `play()` method does not appear to actually throw an instance of `Exception`, because it is declared, the `main()` method must catch or declare it. Since the checked exception is not handled, the `main()` method does not compile, and Option D is the correct answer. If the `main()` method was changed to declare the appropriate checked exception, then the rest of the code would compile, and exactly one exception would be printed, making Option A the correct answer.
28. B. Assertions are often used to check method post conditions, test control flow invariants, and validate class invariants, making Options A, C, and D true statements. Option B is the correct answer. An assertion should never modify any data because it may be disabled at runtime, leading to unintended side effects.
29. B. A multi-catch block cannot contain two exceptions in which one is a subclass of the other, since it is a redundant expression. Since `CarCrash` is a subclass of `RuntimeException` and `RuntimeException` is a subclass of `Exception`, line `w2` contains a compilation error, making Option B the correct answer. The rest of the lines of code do not contain any compilation errors.

- 30.** B. First off, a class must inherit from `RuntimeException` or `Error` to be considered an unchecked exception. `Dopey` and `Grumpy` both are subclasses of `Exception`, but not `RuntimeException`, making them both checked exceptions. Since `IOException` is a checked exception, the subclass `Happy` is also a checked exception. `Sleepy` extends `IllegalStateException`, which is an unchecked exception that extends `RuntimeException`. Finally, `Sneezy` extends `Throwable`, which does not inherit `RuntimeException` or `Error`, making it a checked exception. Therefore, there are a total of four checked exceptions and one unchecked exception within the classes listed here. Since there are no compilation errors in any of the class declarations, Option B is the correct answer, with the first and third statement being true.
- 31.** D. The `close()` method in each of the resources throws an `Exception`, which must be handled or declared in the `main()` method. The catch block supports `TimeException`, but it is too narrow to catch `Exception`. Since there are no other catch blocks present and the `main()` method does not declare `Exception`, the try-with-resources statement does not compile, and Option D is the correct answer. If the catch block was modified to handle `Exception` instead of `TimeException`, the code would compile without issue and print 3215 at runtime, closing the resources in the reverse order in which they were declared and making Option B the correct answer.
- 32.** A. The try-catch block already catches `Exception`, so the correct answer would be the one that is not a subclass of `Exception`. In this case, `Error` extends `Throwable` and is the only choice that allows the code to compile. Because `IllegalStateException` and `RingException` both inherit from `Exception`, Options B and C, respectively, are incorrect. Finally, Option D is incorrect because there is an answer choice that allows the code to compile.
- 33.** B. Option A does not compile because the assignment `(age=2)` has a value of `int`, not `boolean`, which is required for an `assert` statement. Option B compiles without issue and is the correct answer. Even though `Error` and `10` are different data types, `String` and `int` respectively, the second `assert` parameter only needs to be a value, so both are allowed. Option C does not compile because the usage of the lambda expression does not match a functional interface. Option D is incorrect because a `return` statement is not allowed in the second expression of an `assert` statement.
- 34.** D. The `MissedCallException` is a checked exception since it extends `Exception` and does not inherit `RuntimeException`. For this reason, the first catch block fails to compile, since the compiler detects that it is not possible to throw this checked exception inside the `try` block, making Option D the correct answer. Note that if `MissedCallException` was changed to extend the checked `RuntimeException`, then the code would compile and the `RuntimeException` from the `finally` block would replace the `ArrayIndexOutOfBoundsException` from the `try` block in the message reported to the caller, making Option C the correct answer.
- 35.** D. The catch variable `d` is of type `BearException` that cannot be assigned an instance of the superclass `RuntimeException` without an explicit cast. For this reason, the first catch block does not compile in `tellStory()`. The second catch block also does not compile, albeit for a slightly different reason. A catch variable in a multi-catch block must be effectively final because the precise type is not known until runtime. Therefore, the compiler does not allow the variable `e` to be reassigned. For these two reasons, Option D is the correct answer. Note that the first catch block does allow the catch variable `d` to be reassigned, it just must be to a class that inherits `BearException` or is an instance of `BearException`.

36. D. The `play()` method declares two checked exceptions, `OutOfTuneException` and `FileNotFoundException`, which are handled in the `main()` method's `catch` block using the `Exception` type. The `catch` block then rethrows the `Exception`. The compiler is smart enough to know that only two possible checked exceptions can be thrown here, but they both must be handled or declared. Since the `main()` method only declares one of the two checked exceptions, `FileNotFoundException` is not handled, and the code does not compile. For this reason, Option D is the correct answer. Note that the `main()` could have also handled or declared `Exception`, since both checked exceptions inherit it. If the `main()` method had declared `Exception`, then `Song finished!` would have been printed followed by a stack trace, making Option C the correct answer.
37. C. The code compiles without issue, making Option D incorrect. Option A is incorrect because assertions are not enabled by default in Java. Therefore, the code will run without throwing any exceptions. Option B is also incorrect because the command enables assertions everywhere but disables them for the `Falcon` class. Option C is the correct answer, with the command disabling assertions everywhere except in the `Falcon` class, causing an `AssertionError` to be thrown at runtime.
38. C. The `Closeable` interface defines a `close()` method that throws `IOException`. The overridden implementation of `MyDatabase`, which implements `Closeable`, declares a `SQLException`. This is a new checked exception not found in the inherited method signature. Therefore, the method override is invalid and the `close()` method in `MyDatabase` does not compile, making Option C the correct answer.
39. D. The code does not compile because the `close()` method throws an `Exception` that is not handled or declared in the `main()` method, making Option D the correct answer. When a `try-with-resources` statement is used with a `close()` method that throws a checked exception, it must be handled by the method or caught within the `try-with-resources` statement.
40. B. The code compiles without issue, making Option C incorrect. In the `climb()` method, two exceptions are thrown. One is thrown by the `close()` method and the other by the `try` block. The exception thrown in the `try` block is considered the primary exception and reported to the caller on line `e1`, while the exception thrown by the `close()` method is suppressed. For this reason, `java.lang.RuntimeException` is thrown to the `main()` method, and Option B is the correct answer.

## Chapter 17: Use Java SE 8 Date/Time API

1. C. The date and time classes added in Java 8 are in the `java.time` package, making Option C correct. The older date classes are in the `java.util` package.
2. A. The `Duration` class is used to reflect an amount of time using small units like minutes. Since it just uses units of time, it does not involve time zones. The `LocalTime` class contains units of hours, minutes, seconds, and fractional seconds. The `LocalDateTime` class contains all the data in a `LocalTime` and adds on a year, month, and date. Neither of these classes uses time zones. There is a `ZonedDateTime` class when you need to use time zones. Since none of the three classes listed includes a time zone, Option A is correct.

3. A. A `Period` is measured in days, weeks, months, or years. A `Duration` is measured in smaller units like minutes or seconds. Only `Duration` has a `getSeconds()` method, making Option A correct.
4. D. To compare times in different time zones, you can subtract the time zone from the time to convert to GMT. This makes it 02:00 in Berlin because we subtract 1 from 3. Similarly, it is 02:00 in Helsinki due to subtracting 2 from 4. Finally, it is 04:00 in Warsaw because we subtracted 1 from 5. We have a tie because it is the same time in Berlin and Helsinki, so Option D is correct.
5. B. On a normal night, adding three hours to 1 a.m. makes it 4 a.m. However, this date begins daylight savings time. This means we add an extra hour to skip the 2 a.m. hour. This makes `later` contain 05:00 instead of 04:00. Therefore, the code prints 5, and Option B is correct.
6. C. `LocalDate` allows passing the month as an `int` or `Month` enum parameter. However, `Month.MARCH` is an enum. It cannot be assigned to an `int` variable, so the declaration of `month` does not compile, and Option C is correct.
7. C. Both `LocalDate` and `DateTimeFormatter` have a `format()` method. This makes II incorrect. While it is tricky, you do need to know that the `format()` method can be called on either object. Since I and III are correct, Option C is the answer.
8. C. Converting to GMT by subtracting the time zone offset, it is 17:00 for the Phoenix time since 10 minus negative 7 is 17. In GMT, the Vancouver time is 16:00 due to subtracting negative 8 from 8. Remember that subtracting a negative number is the same as adding. Therefore, the Vancouver time is an hour earlier than the Phoenix time, and Option C is correct.
9. C. While there is no 2 a.m. on the clock that night, Java adjusts the time to 3 a.m. automatically and changes the time zone. It does not throw an exception, so Option D is incorrect. Option B is a valid expression, since any value after the time adjustment is just a normal time on the clock. Since both A and B are valid expressions, Option C is the correct answer.
10. B. Line 12 creates a `Period` representing a year, six months, and three days. Adding this to `waffleDay` gives us the year 2018, the month of September, and a day of 28. This new date is stored in `later` on line 13 and represents September 28, 2018. Line 14 has no effect as the return value is ignored. Line 17 checks that you know that `isBefore()` returns `false` if the value is an exact match. Since `thisOne` is an exact match but `thatOne` is a whole day before, the output is `false true`, making Option B correct.
11. D. `Duration` is supposed to be used with objects that contain times. While it has an `ofDays()` method, this is a convenience method to represent a large number of seconds. This means that calling `Duration.ofDays(1)` is fine. However, this code throws an `UnsupportedTemporalTypeException` when you try to pass it the `minus()` method on `LocalDate`, making Option D correct. Note that the question asks about a possible result rather than the definitive result because the format of dates varies by region.

12. C. The `DateTimeFormatter` is created with `ofLocalizedDate()`. It knows how to format date fields but not time fields. Line 18 is fine because a `LocalDate` clearly has date fields. Line 19 is also fine. Since a `LocalDateTime` has both date and time fields, the formatter just looks at the date fields. Line 20 is a problem. A `LocalTime` object does not have any date fields so the formatter throws an `UnsupportedTemporalTypeException`, making Option C the answer.
13. D. This question is tricky. It appears to be about daylight savings time. However, the result of `z.plusHours(1)` is never stored in a variable or used. Since `ZonedDateTime` is immutable, the time remains at 01:00. The code prints out 1, making none of these correct and Option D the answer.
14. D. For dates, a lowercase `m` means minute while an uppercase `M` means month. This eliminates Options A and C. A lowercase `h` means hour. Therefore, Option B is incorrect, and Option D is the answer.
15. D. There are three overloads for `LocalTime.of()`. Options A, B, and C are all valid overloads. Option D is not because Java only allows passing one fractional second parameter. Java does support nanoseconds, but not the further granularity of picoseconds.
16. C. The `LocalDate` class represents a date using year, month, and day fields. There is a `getYear()` method to get the year. The `Period` class holds units of years, months, and days. It has a `getYears()` method. There is not a date/time class called `ZonedDateTime`. There is a class called `ZonedDateTime`, which does have a `getYear()` method. Since only `LocalDate` and `Period` have a method to get the year, Option C is correct.
17. A. `Duration` is used for units of time a day and smaller, making Option B a true statement. `Period` is used for units of time a day and larger, making Option C a true statement. While both represent the same length of time, they output different values when calling `toString()`. The `Duration` object outputs `PT24H`, and the `Period` object outputs `P1D`. This shows that `Duration` is providing the `ofDays()` method as a convenience instead of requiring the programmer to type 24 hours. Option A is the answer.
18. B. The first thing to notice is that this is a `LocalTime` object. Since there is no date component, Options C and D are incorrect. Four parameters are passed to this `LocalTime` method. The first three are the hour, minute, and second. The fourth is nanoseconds, which are fractions of a second. While you aren't expected to do calculations with nanoseconds, you should know that a fraction of a second is at the end of the output. Option A is incorrect because `.4` is 40 percent of a second. That's far larger than 4 nanoseconds. Therefore, Option B is correct.
19. B. `LocalDate` starts counting months from one, so month 2 is February. This rules out Options A and C. The pattern specifies that the date should appear before the month, making Option B correct.
20. A. The `ChronoUnit` enum contains values for various measures of time including `HOURS`, so Option A is correct.

- 21.** B. Adding three hours to 13:00 makes it 16:00. While this date happens to be the start of daylight savings time, the change occurs at 2 a.m. This code uses 13:00, which is 1 p.m. Since the calculation does not cross 2 a.m., the fact that it is the date that starts daylight savings time is irrelevant. Option B is correct because the hour is 16 and the time is 16:00.
- 22.** B. This code correctly subtracts a day from `montyPythonDay`. It then outputs a `LocalDateTime` value. Option A is incorrect because it omits the time. Option B is correct because it represents one day earlier than the original date and includes a time in the output.
- 23.** D. There is a `DateTimeFormatter` class, but not a `DateFormat` class. The `DateTimeFormatter` class is used for formatting dates, times, or both. Since the provided code does not compile, nothing can fill in the blank to make the code print `2017-01-15`, and Option D is the answer.
- 24.** B. There are many overloads for `LocalDateTime.of()`. Option A is a valid overload because it uses date and time objects. Options C and D are also valid overloads, showing you can pass the month as an `int` or `Month` enum. Option B is the answer. Java doesn't allow combining a `LocalDate` object with time fields directly.
- 25.** C. In the first time change of the year, clocks "spring ahead" and skip the 02:00–03:00 hour entirely. This means 1:59 is followed by 03:00 on March 12, 2017. By contrast, July 4 is a normal day and 1:59 is followed by 02:00. In the second time change of the year, clocks "fall back" and repeat an hour, so 1:59 is followed by 01:00. Granted, you can't tell whether this is the first or second 1:59 from the image. If this information is relevant to a question's context, the question will specify this fact. In this case, 03:00, 02:00, 02:00 is not a choice. Option C is the answer.
- 26.** D. February has 28 or 29 days, depending on the year. There is never a February 31. Java throws a `DateTimeException` when you try to create an invalid date, making Option D correct.
- 27.** A. This one is tricky. In order to determine GMT, you need to subtract the time zone offset from the hour. In this case, the time zone offset is negative 10. Since subtracting a negative number is like adding a positive number, this means we are adding 16 and 10. That gives us 26. However, there are only 24 hours in a day. We've crossed a time zone boundary, so we can remove a whole 24-hour day. Subtracting 24 from 26 gives us 2. This means it is 02:00 in GMT, and Option A is correct. It's also a day later in GMT, but the question didn't ask that.
- 28.** D. An `Instant` represents a specific moment in time using GMT. Since there are no time zones included, Options A and C are incorrect. This code correctly adds one day to the `instant`, making Option D correct.
- 29.** D. Make sure to pay attention to date types. This code attempts to add a month to a `LocalTime` value. There is no `plusMonths()` method on `LocalTime`, so Option D is correct.
- 30.** D. The format of the pattern is incorrect. You can't just put literal text in there. Most of the characters of `Holiday:` are not defined formatting symbols. The code throws an `IllegalArgumentException`, so Option D is correct.

31. A. To compare times in different time zones, you can subtract the time zone from the time. This makes it 09:00 in Bangkok because we subtract 7 from 16. Similarly, it is 14:00 in Dubai due to subtracting 4 from 18. Finally, it is 12:00 in Kuala Lumpur because we subtracted 8 from 20. Notice how we used a 24-hour clock to make comparing times easier. The earliest time is in Bangkok, so Option A is correct.
32. C. Line 12 creates a `Period` representing three days. `Period` objects do not chain, so only the last method call, which is `to ofDays(3)`, is used in determining the value. Adding three days sets `later` to March 28, 2017. Line 14 has no effect as the return value is ignored. March 28, 2017, is before both `thisOne` and `thatOne`, so Option C is correct.
33. B. The `TemporalUnit` interface does not define a `DAYS` constant, making II and IV incorrect. The `until()` and `between()` methods have the same behavior. They determine how many units of time are between two dates. One takes both dates as parameter and the other is an instance method on the date. Since I and III are equivalent, Option B is the answer. Note that while we don't have date times in this question, the `until()` and `between()` methods work the same way for them.
34. A. The `DateTimeFormatter` class is used to format all of these objects. The method will throw an exception if called with a `LocalDate` since the formatter only knows about time fields. However, it will still compile, making Option A correct.
35. B. This code begins by correctly creating four objects. It then adds a month to `date`. Since Java 8 date/time classes are immutable, this does not affect the value of `iceCreamDay`. Therefore, `iceCreamDay` remains in July. Since months count from one, Option B is correct.
36. A. Java 8 date and time classes are immutable. They use a `static` factory method to get the object reference rather than a constructor. This makes Options B and D incorrect. Further, there is not a `ZonedDateTime` class. There is a `ZonedDateTime` class. As a result, Option C is incorrect, and Option A is the answer.
37. B. The first line of code correctly creates a `LocalDate` object representing March 3, 2017. The second line adds two days to it, making it March 5. It then subtracts a day, making it March 4. Finally, it subtracts yet another day ending at March 3. The outcome of all this is that we have two dates that have the same value, and Option B is correct.
38. C. An `Instant` represents a specific moment in time using GMT. Since `LocalDateTime` does not have a time zone, it cannot be converted to a specific moment in time. Therefore, the `toInstant()` call does not compile, and Option C is correct.
39. A. While it is traditional to include the year when outputting a date, it is not required. This code correctly prints the month followed by a decimal point. After the decimal point, it prints the day of the month followed by the hours and minutes. Happy Pi Day!
40. C. Normally, adding an hour would result in 02:00 in the same time zone offset of `-05:00`. Since the hour is repeated, it is 01:00 again. However, the time zone offset changes instead. Therefore, Option C is correct.

## Chapter 18: Java I/O Fundamentals

1. B. `Writer` is an abstract class, making Option B the correct answer. Note that `InputStream`, `OutputStream`, and `Reader` are also abstract classes.
2. D. `File` uses `mkdir()` and `mkdirs()` to create a directory, not `createDirectory()`, making Option A incorrect. Note there is a `createDirectory()` method in the NIO.2 `Files` class. The `getLength()` method also does not exist, as the correct method is called `length()`. Next, there is a `listFiles()` method used to read the contents of a directory, but there is no `listFile()` method. That leaves us with `renameTo()`, which does exist and is used to rename file system paths.
3. C. The `skip()` method just reads a single byte and discards the value. The `read()` method can be used for a similar purpose, making Option C the correct answer. Option A is incorrect because there is no `jump()` method defined in `InputStream`. Options B and D are incorrect because they cannot be used to skip data, only to mark a location and return to it later, respectively.
4. D. `Serializable` is a marker or tagging interface, which means it does not contain any methods and is used to provide information about an object at runtime. Therefore, Option D is the correct answer because the interface does not define any abstract methods.
5. C. Given a valid instance of `Console`, `reader()` returns an instance of `Reader`, while `writer()` returns an instance of `PrintWriter`. `Reader` and `PrintWriter` was not an answer choice though, making Option C the next best choice since `PrintWriter` inherits `Writer`. Options A and B are incorrect because `PrintReader` is not defined in the `java.io` library. Option D is incorrect because the type of the instance returned by `reader()` is `Reader`, which does not inherit `StringReader`.
6. D. `BufferedWriter` is a wrapper class that requires an instance of `Writer` to operate on. In the `Smoke` class, a `FileOutputStream` is passed, which does not inherit `Writer`, causing the class not to compile, and making Option D the correct answer. If `FileWriter` was used instead of `FileOutputStream`, the code would compile without issue and print 13, making Option B the correct answer.
7. A. The `File` class is used to read both files and directories within a file system, making Option A the correct answer. The other three classes do not exist. Note there is an NIO.2 interface, `java.nio.file.Path`, used to read both file and path information.
8. C. `FileOutputStream` and `FileReader` are both low-level streams that operate directly on files, making Options A and B incorrect. `ObjectInputStream` is a high-level stream that can only wrap an existing `InputStream`. For this reason, Option C is the correct answer. `PrintWriter` can operate on other streams, but it can also operate on files. Since the question asks which class can only wrap low-level streams, Option D is incorrect.
9. D. The code compiles, so Option C is incorrect. The `FileInputStream` does not support marks, though, leading to an `IOException` at runtime when the `reset()` method is called.

For this reason, Option D is the correct answer. Be suspicious of any code samples that call the `mark()` or `reset()` method without first calling `markSupported()`.

10. C. The absolute path is the full path from the root directory to the file, while the relative path is the path from the current working directory to the file. For this reason, Option C is the correct answer.
11. D. The difference between the two methods is that `writeSecret1()` does not take any steps to ensure the `close()` method is called after the resource is allocated. On the other hand, `writeSecret2()` uses a try-with-resources block, which automatically tries to close the resource after it is used. Without a try-with-resources statement or an equivalent `finally` block, any exception thrown by the `write()` method would cause the resource not to be closed in the `writeSecret1()` method, possibly leading to a resource leak. For this reason, Option D is the correct answer. Option A is incorrect since they are not equivalent to each other. Finally, Options B and C are incorrect because both compile without issue.
12. A. The constructor for `Console` is `private`. Therefore, attempting to call `new Console()` outside the class results in a compilation error, making Option A the correct answer. The correct way to obtain a `Console` instance is to call `System.console()`. Even if the correct way of obtaining a `Console` had been used, and the `Console` was available at runtime, `stuff` is `null` in the `printItinerary()` method. Referencing `stuff.activities` results in a `NullPointerException`, which would make Option B the correct answer.
13. A. While you might not be familiar with `FilterOutputStream`, the diagram shows that the two classes must inherit from `OutputStream`. Options B and C can be eliminated as choices since `PrintOutputStream` and `Stream` are not the name of any `java.io` classes. Option D can also be eliminated because `OutputStream` is already in the diagram, and you cannot have a circular class dependency. That leaves us with the correct answer, Option A, with `BufferedOutputStream` and `PrintStream` both extending `FilterOutputStream`. Note that `ByteArrayOutputStream` and `FileOutputStream` referenced in Options C and D, respectively, do not extend `FilterOutputStream`, although knowing this fact was not required to solve the problem.
14. D. The `Cereal` class does not implement the `Serializable` interface; therefore, attempting to write the instance to disk, or calling `readObject()` using `ObjectInputStream`, will result in a `NotSerializableException` at runtime. For this reason, Option D is the correct answer. If the class did implement `Serializable`, then the value of `name` would be `CornLoops`, since none of the constructor, initializers, or setters methods are used on deserialization, making Option B the correct answer.
15. B. An `OutputStream` is used to write bytes, while a `Writer` is used to write character data. Both can write character data, the `OutputStream` just needs the data converted to bytes first. For this reason, Option A is incorrect. Option B is the correct answer, with `Writer` containing numerous methods for writing character or `String` data. Both interfaces contain a `flush()` method, making Option C incorrect. Finally, because both can be used with a byte array, Option D is incorrect.

- 16.** C. First off, the code compiles without issue. The first method call to `mkdirs()` creates two directories, `/templates` and `/templates/proofs`. The next `mkdir()` call is unnecessary, since `/templates/proofs` already exists. That said, calling it on an existing directory is harmless and does not cause an exception to be thrown at runtime. Next, a file `draft.doc` is created in the `/templates` directory. The final two lines attempt to remove the newly created directories. The first call to `delete()` is successful because `/templates/proofs` is an empty directory. On the other hand, the second call to `delete()` fails to delete the directory `/templates` because it is non-empty, containing the file `draft.doc`. Neither of these calls trigger an exception at runtime, though, with `delete()` just returning a boolean value indicating whether the call was successful. Therefore, our program ends without throwing any exceptions, and Option C is the correct answer.
- 17.** D. To answer the question, you need to identify three of the four ways to call the system-independent file name separator. For example, the file name separator is often a forward-slash (/) in Linux-based systems and a backward-slash (\) in Windows-based systems. Option A is valid because it is the fully qualified name of the property. Option B is also valid because `File.separator` and `File.separatorChar` are equivalent. While accessing a `static` variable using an instance is discouraged, as shown in Option B, it is allowed. Option C is valid and a common way to read the character using the `System` class. Finally, Option D is the correct answer and one call that cannot be used to get the system-dependent name separator character. Note that `System.getProperty("path.separator")` is used to separate sets of paths, not names within a single path.
- 18.** D. The first compilation error is that the `FileReader` constructor call is missing the `new` keyword. The second compilation error is that the `music` variable is marked `final`, but then modified in the `while` loop. The third compilation problem is that the `readMusic()` method fails to declare or handle an `IOException`. Even though the `IOException` thrown by `readLine()` is caught, the one thrown by the implicit call to `close()` via the `try-with-resources` block is not caught. Due to these three compilation errors, Option D is the correct answer.
- 19.** C. Both of the methods do exist, making Option D incorrect. Both methods take the same arguments and do the exact same thing, making Option C the correct answer. The `printf()` was added as a convenience method, since many other languages use `printf()` to accomplish the same task as `format()`.
- 20.** C. `FileWriter` and `BufferedWriter` can be used in conjunction to write large amounts of text data to a file in an efficient manner, making Option C the correct answer. While you can write text data using `FileOutputStream` and `BufferedOutputStream`, they are primarily used for binary data. Since there is a better choice available, Option A is incorrect. Option B is incorrect since `FileOutputWriter` and `FileBufferedWriter` are not classes that exist within the `java.io` API. Option D is incorrect since `ObjectOutputStream` is a high-level binary stream. Also, while it can write `String` data, it writes it in a binary format, not a text format.
- 21.** D. The code compiles and runs without issue, so Options A and B are incorrect. The problem with the implementation is that checking if `ios.readObject() == null` is not the recommended way of iterating over an entire file. For example, the file could have been written with `writeObject(null)` in-between two non-null records. In this case, the reading of the

file would stop on this `null` value, before the end of the file has been reached. For this reason, Option D is the correct answer. Note that the valid way to iterate over all elements of a file using `ObjectInputStream` is to continue to call `readObject()` until an `EOFException` is thrown.

22. D. `BufferedInputStream` is the complement of `BufferedOutputStream`. Likewise, `BufferedReader` and `FileReader` are the complements of `BufferedWriter` and `FileWriter`, respectively. On the other hand, `PrintWriter` does not have an accompanying `PrintReader` class within the `java.io` API, making Option D the correct answer. Remember that this is also true of `PrintStream`, as there is no `PrintInputStream` class.
23. C. The `File getParent()` method returns a `String`, not a `File` object. For this reason, the code does not compile on the last line since there is no `getParent()` method defined in the `String` class, and Option C is correct. If the first method call on the last line was changed to `getParentFile()`, then the code would compile and run without issue, outputting `/ - null` and making Option B the correct answer. The `File` class does not require the location to exist within the file system in order to perform some operations, like `getParent()`, on the path.
24. D. All three statements about the program are correct. If `System.console()` is available, then the program will ask the user a question and then print the response if one is entered. On the other hand, if `System.console()` is not available, then the program will exit with a `NullPointerException`. It is strongly recommended to always check whether or not `System.console()` is `null` after requesting it. Finally, the user may choose not to respond to the program's request for input, resulting in the program hanging indefinitely and making the last statement true.
25. C. The code contains two compilation errors. First, the `File list()` method returns a list of `String` values, not `File` values, so the call to `deleteTree()` with a `String` value does not compile. Either the call would have to be changed to `f.listFiles()` or the lambda expression body would have to be updated to `deleteTree(new File(s))` for the code to work properly. Next, there is no `deleteDirectory()` method in the `File` class. Directories are deleted with the same `delete()` method used for files, once they have been emptied. With those two sets of corrections, the method would compile and be capable of deleting a directory tree. Notice we continually used the phrase "capable of deleting a directory tree." While the corrected code is able to delete a directory tree, it may fail in some cases, such as if the file system is read-only.
26. C. `System.err`, `System.in`, and `System.out` are each valid streams defined in the `System` class. `System.info` is not, making Option C the correct answer.
27. D. The code compiles without issue, making Options B and C incorrect. The `BufferedWriter` uses the existing `FileWriter` object as the low-level stream to write the file to disk. By using the try-with-resources block, though, the `BufferedWriter` calls `close()` before executing any associated catch or finally blocks. Since closing a high-level stream automatically closes the associated low-level stream, the `w` object is already closed by the time the finally block is executed. For this reason, the `flush()` command triggers an `IOException` at runtime, making Option D the correct answer. Note that the call to `w.close()`, if that line was reached, does not trigger an exception, because calling `close()` on already closed streams is innocuous.

- 28.** B. The `Console` class contains a `reader()` method that returns a `Reader` object. The `Reader` class defines a `read()` method, but not a `readLine()` method. For this reason, Option B is the correct answer. Recall that a `BufferedReader` is required to call the `readLine()` method. Options A, C, and D are valid ways to read input from the user.
- 29.** C. The code compiles without issue, since `InputStream` and `OutputStream` both support reading/writing byte arrays, making Option A incorrect. Option D is also incorrect. While it is often recommended that an I/O array be a power of 2 for performance reasons, it is not required, making Option D incorrect. This leaves us with Options B and C. The key here is the `write()` method used does not take a length value, available in the `chirps` variable, when writing the file. The method will always write the entire `data` array, even when only a handful of bytes were read into the `data` array, which may occur during the last iteration of the loop. The result is that files whose bytes are a multiple of 123 will be written correctly, while all other files will be written with extra data appended to the end of the file. For this reason, Option C is the correct answer. If the `write(data)` call was replaced with `write(data, 0, chirps)`, which does take the number of bytes read into consideration, then all files would copy correctly, making Option B the correct answer.
- 30.** C. The class name has three components that tell you what it would do if it was a `java.io` stream. First, `Buffered` tells you it can be used to handle large data sets efficiently. Next, `File` tells you it is involved in reading or writing files. Finally, `Reader` tells you it is used to read character data. Therefore, the class would be useful for reading large files of character data from disk efficiently, making Option C the correct answer. Option A is incorrect because it refers to a small file over a network. Options B and D are incorrect because both involve binary data.
- 31.** A. The code compiles and runs without issue. The first two values of the `ByteArrayInputStream` are read. Next, the `markSupported()` value is tested. Since `-1` is not one of the possible answers, we assume that `ByteArrayInputStream` does support marks. Two values are read and three are skipped, but then `reset()` is called, putting the stream back in the state before `mark()` was called. In other words, everything between `mark()` and `reset()` can be ignored. The last value read is 3, making Option A the correct answer.
- 32.** C. Line 5 creates a `File` object, but that does not create a file in the file system unless `cake.createNewFile()` is called. Line 6 also does not necessarily create a file, although the call to `flush()` will on line 7. Note that this class does not properly close the file resource, potentially leading to a resource leak. Line 8 creates a new `File` object, which is used to create a new directory using the `makedirs()` method. Recall from your studies that `makedirs()` is similar to `mkdir()`, except that it creates any missing directories in the path. Since directories can have periods (.) in their name, such as a directory called `info.txt`, this code compiles and runs without issue. Since two file system objects, a file and a directory, are created, Option C is the correct answer.
- 33.** B. Since the file is stored on disk, `FileInputStream` is an appropriate choice. Next, because the data is quite large, a `BufferedInputStream` would help improve access. Finally, since the data is a set of Java values, `ObjectInputStream` would allow various different formats to be read. The only one that does not help in this process is `BufferedReader`, Option B. `BufferedReader` should be used with text-based Reader streams, not binary `InputStream` objects.

- 34.** B. The `flush()` method is defined on classes that inherit `Writer` and `OutputStream`, not `Reader` and `InputStream`. For this reason, the `r.flush()` in both methods does not compile, making Option B the correct answer and Option C incorrect. The methods are not equivalent even if they did compile, since `getNameSafely()` ensures the resource is closed properly by using a try-with-resources statement, making Option A incorrect for two reasons. Finally, Option D would be correct if the calls to `flush()` were removed.
- 35.** A. First off, the class compiles without issue. Although there are built-in methods to print a `String` and read a line of input, the developer has chosen not to use them for most of the application. The application first prints `Pass`, one character at a time. The `flush()` method does not throw an exception at runtime. In fact, it helps make sure the message is presented to the user. Next, the user enters `badxbad` and presses Enter. The stream stops reading on the `x`, so the value stored in the `StringBuilder` is `bad`. Finally, this value is printed to the user, using the `format()` method along with `Result:` as a prefix. For these reasons, Option A is the correct answer.
- 36.** B. The `readPassword()` returns a `char` array for security reasons. If the data was stored as a `String`, it would enter the shared JVM string pool, potentially allowing a malicious user to access it, especially if there is a memory dump. By using a `char` array, the data can be immediately cleared after it is written and removed from memory. For this reason, Option B is the correct answer. The rest of the statements are not true.
- 37.** A. The `BufferedInputStream` constructor in the `readBook()` method requires an `InputStream` as input. Since `FileReader` does not inherit `InputStream`, the `readBook()` method does not compile, and Option A is the correct answer. If `FileReader` was changed to `FileInputStream`, then the code would compile without issue. Since `read()` is called twice per loop iteration, the program would print every other byte, making Option C correct. Remember that `InputStream read()` returns `-1` when the end of the stream is met. Alternatively, we use `EOFException` with `ObjectInputStream readObject()` to determine when the end of the file has been reached.
- 38.** B. Generally speaking, classes should be marked with the `Serializable` interface if they contain data that we might want to save and retrieve later. Options A, C, and D describe the type of data that we would want to store over a long period of time. Option B, though, defines a class that manages transient or short-lived data. Application processes change quite frequently, and trying to reconstruct a process is often considered a bad idea. For these reasons, Option B is the best answer.
- 39.** D. The `receiveText()` method compiles and runs without issue. The method correctly checks that the `mark()` method is supported before attempting to use it. Based on the implementation with `reset()`, the pointer is in the same location before/after the if-then statement. On the other hand, the `sendText()` method does not compile. The `skip()` method is defined on `InputStream` and `Reader`, not `OutputStream` and `Writer`, making Option D the correct answer. If this line was removed, the rest of the code would compile and run without issue, printing `You up?` at runtime and making Option A the correct answer.

- 40.** B. The class compiles and runs without issue, so Option D is incorrect. The class defines three variables, only one of which is serializable. The first variable, `chambers`, is serializable, with the value 2 being written to disk and then read from disk. Note that constructors and instance initializers are not executed when a class is deserialized. The next variable, `size`, is `transient`. It is discarded when it is written to disk, so it has the default object value of `null` when read from disk. Finally, the variable `color` is `static`, which means it is shared by all instances of the class. Even though the value was `RED` when the instance was serialized, this value was not written to disk, since it was not part of the instance. The constructor call `new Value()` between the two `try-with-resources` blocks sets this value to `BLUE`, which is the value printed later in the application. For these reasons, the class prints `2,null,BLUE`, making Option B the correct answer.

## Chapter 19: Java File I/O (NIO.2)

1. C. A symbolic link is a file that contains a reference to another file or directory within the file system, making Options A and B incorrect. Further, there is no such thing as an irregular file. Option C is the correct answer because a regular file is not a directory and contains content, unlike a symbolic link or resource. Option D is also incorrect because all symbolic links are stored as files, not directories, even when their target is a directory.
2. C. The NIO.2 Path interface contains the methods `getRoot()` and `toRealPath()`. On the other hand, the method `isDirectory()` is found in the NIO.2 `Files` class, while the method `listFiles()` is found in the `java.io.File` class. For these reasons, Option C is the correct answer.
3. A. The code does not compile because there is no NIO.2 class `File` that contains an `isHidden()` method, making Option A the correct answer. There is a `java.io.File` class, but that does not contain an `isHidden()` method either. The correct call is `Files.isHidden()`. Remember to check `File` vs. `Files` as well as `Path` vs. `Paths` on the real exam. If the correct method call was used, the program would print `Found!`, and Option C would be the correct answer.
4. D. A breadth-first traversal is when all elements of the same level, or distance from the starting path, are visited before moving on to the next level. On the other hand, a depth-first traversal is when each element's entire path, from start to finish, is visited before moving onto another path on the same level. Both `walk()` and `find()` use depth-first traversals, so Option D is the correct answer.
5. A. Reading an attribute interface is accomplished in a single trip to the underlying file system. On the other hand, reading multiple file attributes using individual `Files` methods requires a round-trip to the file system for each method call. For these reasons, Option A is the correct answer. Option B is incorrect because nothing guarantees it will perform faster, especially if the `Files` method is only being used to read a single attribute. For multiple calls, it is expected to be faster, but the statement uses the word *guarantees*, which is incorrect. Option C is also incorrect because both have built-in support for symbolic links. Finally, Option D is incorrect because this discussion has nothing to do with memory leaks.

6. B. First off, the class compiles without issue. It is not without problems, though. The `Files.isSameFile()` method call on line `j1` first checks if the `Path` values are equivalent in terms of `equals()`. One is absolute and the other is relative, so this test will fail. The `isSameFile()` method then moves on to verify that the two `Path` values reference the same file system object. Since we know the directory does not exist, the call to `isSameFile()` on line `j1` will produce a `NoSuchFileException` at runtime, making Option B the correct answer.
7. B. A cycle is caused when a path contains a symbolic link that references the path itself, or a parent of the parent, triggering an infinitely deep traversal. That said, `Files.walk()` does not follow symbolic links by default. For this reason, the cycle is never activated, and the code would print a number at runtime, making Option B the correct answer. If the `FOLLOW_LINKS` enum value was used in the call to `Files.walk()`, then it would trigger a cycle resulting in a `FileSystemLoopException` at runtime, and Option A would be the correct answer.
8. B. The methods `length()` and `getLength()` do not exist in the `Files` class, making Options A and C incorrect. Recall that the `java.io.File` method retrieves the size of a file on disk. The NIO.2 `Files` class includes the `Files.size()` method to accomplish this same function. For this reason, Option B is the correct answer.
9. D. The code compiles without issue, making Option C incorrect. Even though `tricks` would be dropped in the normalized path `/bag/of/disappear.txt`, there is no `normalize()` call, so `path.subpath(2, 3)` returns `tricks` on line 5. On line 6, the call to `getName()` throws an `IllegalArgumentException` at runtime. Since `getName()` is zero-indexed and contains only one element, the call on line 6 throws an `IllegalArgumentException`, making Option D the correct answer. If `getName(0)` had been used instead of `getName(1)`, then the program would run without issue and print `/home/tricks`, and Option A would have been the correct answer.
10. A. The NIO.2 `Files` class contains the method `isSameFile()`. The methods `length()` and `mkdir()` are found in `java.io.File`, with the NIO.2 equivalent versions being `Files.size()` and `Files.createDirectory()`, respectively. In addition, the `relativize()` method is found in NIO.2 `Path`, not `Files`. Since only `isSameFile()` is found in NIO.2 `Files`, Option A is the correct answer.
11. B. First off, the code compiles without issue, so Option D is incorrect. The enum value `REPLACE_EXISTING` does not use a type, although this compiles correctly if a `static` import of `StandardCopyOption` is used. The `AtomicMoveNotSupportedException` in Option A is only possible when the `ATOMIC_MOVE` option is passed to the `move()` method. Similarly, the `FileAlreadyExistsException` in Option C is only possible when the `REPLACE_EXISTING` option is not passed to the `move()` method. That leaves us with the correct answer of Option B. A `DirectoryNotEmptyException` can occur regardless of the options passed to the `Files.move()` method.
12. D. The `Path` method `getFileName()` returns a `Path` instance, not a `String`. For this reason, the code does not compile, regardless of which line of code is inserted into the blank, making Option D the correct answer. Statements I and III are two valid ways to create a `Path` instance. If the method was updated to use `Path` as the return type, then Option B would be the correct answer. Statement II would cause the method to not compile, because `Path` is an interface and requires a class to be instantiated.

- 13.** A. The code compiles without issue, but that's about it. The class may throw an exception at runtime, since we have not said whether or not the source file exists nor whether the target file already exists, is a directory, or is write-protected. For these reason, Option B is incorrect. Option C is also incorrect because the implementation is a flawed copy method. On a regular file, the code will copy the contents but the line breaks would be missing in the target file. In order to correctly copy the original file, a line break would have to be written after each time `temp` is written. Since it is the only correct statement, Option A is the correct answer.
- 14.** C. First off, there is no `Files.readLines()` method, making Options B and D immediately incorrect. The `Files.readAllLines()` method returns a `List<String>`, while the `Files.lines()` method returns a `Stream<String>`. For this reason, Option C is the correct answer, and Option A is incorrect.
- 15.** A. The program compiles and runs without issue, making Options C and D incorrect. Like `String` instances, `Path` instances are immutable. For this reason, the `resolve()` operation on line 7 has no impact on the `lessTraveled` variable. Since one `Path` ends with `/spot.txt` and the other does not, they are not equivalent in terms of `equals()`, making Option A the correct answer. If lines 6 and 7 were combined, such that the result of the `resolve()` operation was stored in the `lessTraveled` variable, then `normalize()` would reduce `lessTraveled` to a `Path` value that is equivalent to `oftenTraveled`, making Option B the correct answer.
- 16.** C. Options A, B, and D are each advantages of using NIO.2. As you may remember, using an attribute view to read multiple attributes at once is more efficient than a single attribute call since it involves fewer round trips to the file system. Option C is the correct answer. Neither API provides a single method to delete a directory tree.
- 17.** C. The `Files.delete()` method has a return type of `void`, not `boolean`, resulting in a compilation error and making Option C the correct answer. There is another method, `Files.deleteIfExists()`, which returns `true` if the file is able to be deleted. If it was used here instead, the file would compile and print a list of `true` values, making Option A the correct answer. As stated in the description, the directory tree is fully accessible, so none of the `Files.deleteIfExists()` would return `false`.
- 18.** D. First off, `DosFileAttributes` and `PosixFileAttributes` extend `BasicFileAttributes`, which means they are compatible with the `readAttributes()` method signature. Second, they produce instances that inherit the interface `BasicFileAttributes`, which means they can be assigned to a variable `b` of type `BasicFileAttributes` without an explicit cast. For this reason, all three interfaces are permitted, and Option D is the correct answer.
- 19.** D. The `relativize()` method requires that both path values be absolute or relative. Based on the details provided, `p1` is a relative path, while `p2` is an absolute path. For this reason, the code snippet produces an exception at runtime, making Option D the correct answer. If the first path was modified to be absolute by dropping the leading dot (.) in the path expression, then the output would match the values in Option A.

20. C. First off, p2 is an absolute path, which means that `p1.resolve(p2)` just returns p2. For this reason, Option B is incorrect. Since p1 is a relative path, it is appended onto p2, making Option C correct and Option A incorrect.
21. B. The code does not compile because `Files.list()` returns a `Stream<Path>`, not a `List<Path>`, making Option B the correct answer. Note that `java.io.File` does include a `list()` method that returns an array of `String` values and a `listFiles()` method that returns an array of `File` values, but neither is applicable here.
22. C. For this problem, remember that the path symbols can be applied to simplify the path before needing to apply any symbolic links in the file system. The paths in Options A and B can both be reduced from `/objC/bin/..../backwards/..../forward/Sort.java` and `/objC/bin/..../forward./Sort.java`, respectively, to `/objC/forward/Sort.java` just using the path symbols. Because of the symbolic link, this references the same file as `/java/Sort.java`. For these reasons, Options A and B match our target path. Option C can be reduced from `/objC/bin/..../java./forward/Sort.java` to `/objC/java/forward/Sort.java`, which does not match the desired path for the file. The symbolic link is not followed since it exists in the `/objC` directory, not in the `/objC/java` directory. This causes a stack trace to be printed at runtime since the path does not exist, making Option C the correct answer. Option D can be reduced from `/objC/bin/..../..../java/Sort.java` to `/java/Sort.java`, which matches the target path without using the symbolic link.
23. B. We need to empty the `/objC` directory before we can delete it. First, the `Heap.exe` file would have to be deleted before the `bin` directory could be removed, for a total of two calls to `Files.delete()`. Next, the `Heap.m` file is easily deleted with a single call to `Files.delete()`. Calling `Files.delete()` on the symbolic link `forward` deletes the link itself and leaves the target of the symbolic link intact. With a total of four calls, Option B is the correct answer. Option A is incorrect because Java requires directories to be empty before they can be deleted. Option C is also incorrect. It might make sense if `Files.delete()` traversed symbolic links on a delete, but since this is not the case, it is an incorrect answer. Option D is incorrect because there is no `Files.deleteSymbolicLink()` method defined in the Java NIO.2 API.
24. C. Since `System.out` is a `PrintStream` that inherits `OutputStream` and implements `Closeable`, line `y1` compiles without issue. On the other hand, the `Files.copy()` does not compile because there is no overloaded version of `Files.copy()` that takes an `OutputStream` as the first parameter. For this reason, Option C is the correct answer. If the order of the arguments in the `Files.copy()` call was switched, then the code would compile and print the contents of the file at runtime, making Option D the correct answer.
25. B. To begin with, the `BasicFileAttributeView` class contains methods to read and write file data, while the `BasicFileAttributes` class only contains methods to read file data. The advantage of using a `BasicFileAttributeView` is to also modify file data, so Option D is incorrect. Next, The `BasicFileAttributeView` does not include a method to modify the `hidden` attribute. Instead, a `DosFileAttributeView` is required, making Option A incorrect. Option B is the correct answer because `BasicFileAttributeView` includes a `setTimes()` method to modify the file date values. Finally, Option C is incorrect because both read file information in a single round-trip.

- 26.** A. Trick question! The code does not compile, therefore no Path values are printed, and Option A is the correct answer. The key here is that `toRealPath()` interacts with the file system and therefore throws a checked `IOException`. Since this checked exception is not handled inside the lambda expression, the class does not compile. If the lambda expression was fixed to handle the `IOException`, then the expected number of Path values printed would be six, and Option C would be the correct answer. A `maxDepth` value of 1 causes the `walk()` method to visit two total levels, the original `/flower` and the files it contains.
- 27.** D. The first statement returns a `null` value, since the path `..` does not have a parent. That said, it does not throw an exception at runtime, since it is not operated upon. The second and third statements both return paths representing the root `(/)` at runtime. Remember that calling `getRoot()` on a root path returns the root path. The fourth statement throws a `NullPointerException` at runtime since `getRoot()` on a relative path returns `null`, with the call to `getParent()` triggering the exception. Since the fourth statement is the only one to produce a `NullPointerException` at runtime, Option D is the correct answer.
- 28.** C. The code compiles without issue, so Options A and B are incorrect. While many of the `Files` methods do throw `IOException`, most of the `Path` methods do not throw a checked exception. The lack of indent of the return statement on line 6 is intentional and does not prevent the class from compiling. If the input argument `p` is `null` or not an absolute path, then the if-then clause is skipped, and it is returned to the caller unchanged. Alternatively, if the input argument is an absolute path, then calling `toAbsolutePath()` has no effect. In both cases, the return value of the method matches the input argument, making Option C the correct answer.
- 29.** D. Option A is incorrect because both methods take exactly one `Path` parameter, along with an optional vararg of `FileAttribute` values. Option B is also incorrect because both methods will throw a `FileAlreadyExistsException` if the target exists and is a file. Option C is incorrect since both methods declare a checked `IOException`. The correct answer is Option D. The method `createDirectory()` creates a single directory, while `createDirectories()` may create many directories along the path value.
- 30.** C. The `toAbsolutePath()` combines the current working directory and relative path to form a `/hail/.../jungle/.../rain..` path. The `normalize()` method removes the path symbols and leaves a `/rain..` value. Note that the last double period `(..)` is not removed because it is part of a path name and not interpreted as a path symbol. The result is then appended with `snow.txt` and we are left with `/rain../snow.txt`, making Option C the correct answer.
- 31.** A. The program compiles and runs without issue, so Options C and D are incorrect. The process breaks apart the inputted path value and then attempts to reconstitute it. There is only one problem. The method call `getName(0)` does not include the root element. This results in the `repaired` variable having a value of `tissue/heart/chambers.txt`, which is not equivalent to the original path. The program prints `false`, and Option A is the correct answer.
- 32.** B. Unlike `Files.delete()`, the `Files.deleteIfExists()` method does not throw an exception if the path does not exist, making Option B the correct answer. Options A, C, and D describe situations in which the Java process encounters a path in a state that cannot be deleted. In each of these situations, an exception would be thrown at runtime.

- 33.** D. The code does not compile because `Path` is an interface and does not contain a `get()` method. Since the first line contains a compilation error, Option D is the correct answer. If the code was corrected to use `Paths.get()`, then the output would be `true false true`, and Option B would be the correct answer. The normalized path of both is `/desert/sand.doc`, which means they would be equivalent, in terms of `equals()`, and point to the same path in the file system. On the other hand, the non-normalized values are not equivalent, in terms of `equals()`, since the objects represent distinct path values.
- 34.** C. First off, the `Files.getFileAttributeView()` method requires a reference to a subclass of `FileAttributeView`, such as `BasicFileAttributeView.class`. The parameter must also be compatible with the reference assignment to `vw`. For these two reasons, this line of code does not compile. Next, `BasicFileAttributeView` does not contain a `creationTime()` method, so `vw.creationTime()` results in a compilation error. For the exam, remember that view classes do contain access to attributes, but only through the `readAttributes` method, such as `vw.readAttributes().creationTime()`. Since these are the only two lines that contain compilation errors, Option C is the correct answer. Note that we purposely omitted all `import` statements in this question, since this may happen on the real exam.
- 35.** B. The program compiles and runs without issue, making Options C and D incorrect. The first variable, `halleysComet`, is created with `normalize()` being applied right away, leading to a value of `stars/m1.meteor`. The second variable, `lexellsComet`, starts with a value of `./stars/.../solar/`. The `subpath()` call reduces it to its first two components, `./stars`. The `resolve()` method then appends `m1.meteor`, resulting in a value of `./stars/m1.meteor`. Finally, `normalize()` further reduces the value to `stars/m1.meteor`. Since this matches our first `Path`, the program prints `Same!`, and Option B is the correct answer.
- 36.** D. Both stream statements compile without issue, making Options A and B incorrect. The two statements are equivalent to one another and print the same values at runtime. For this reason, Option C is incorrect, and Option D is correct. There are some subtle differences in the implementation besides one using `walk()` with a `filter()` and the other using `find()`. The `walk()` call does not include a depth limit, but since `Integer.MAX_VALUE` is the default value, the two calls are equivalent. Furthermore, the `walk()` statement prints a stream of absolute paths stored as `String` values, while the `find()` statement prints a stream of `Path` values. If the input `p` was a relative path, then these two calls would have very different results, but since we are told `p` is an absolute path, the application of `toAbsolutePath()` does not change the results.
- 37.** A. The code does not compile because `Files.lines()` and `Files.readAllLines()` throw a checked `IOException`, which must be handled or declared. For the exam, remember that other than a handful of test methods, like `Files.exists()`, most methods in the NIO.2 `Files` class that operate on file system records declare an `IOException`. Now, if the exceptions were properly handled or declared within the class, then `jonReads()` would likely take more time to run. Like all streams, `Files.lines()` loads the contents of the file in a lazy manner, meaning the time it takes for `jenniferReads()` to run is constant regardless of the file size. Note the stream isn't actually traversed since there is no terminal operation. Alternatively, `Files.readAllLines()` reads the entire contents of the file before returning a list of `String` values. The larger the file, the longer it takes `jonReads()` to execute. Since the original question says the file is significantly large, then if the compilation problems were corrected, `jonReads()` would likely take longer to run, and Option C would be the correct answer.

- 38.** C. The first `copy()` method call on line `q1` compiles without issue because it matches the signature of a `copy()` method in `Files`. It also does not throw an exception because the `REPLACE_EXISTING` option is used and we are told the file is fully accessible within the file system. On the other hand, the second `copy()` method on line `q2` does not compile. There is a version of `Files.copy()` that takes an `InputStream`, followed by a `Path` and a list of `copy` options. Because `BufferedReader` does not inherit `InputStream`, though, there is no matching `copy()` method and the code does not compile. For this reason, Option C is the correct answer.
- 39.** C. The `Files.isSameFile()` throws a checked `IOException`. Even though accessing the file system can be skipped in some cases, such as if the `Path` instances are equivalent in terms of `equals()`, the method still declares `IOException` since it may access the file system to determine if the two `Path` instances refer to the same file. For this reason, Option C is the correct answer. The rest of the methods listed do not throw any checked exceptions, even though they do access the file system, instead returning `false` if the file does not exist.
- 40.** B. The program compiles and runs without issue, making Options C and D incorrect. The program uses `Files.list()` to iterate over all files within a single directory. For each file, it then iterates over the lines of the file and counts the sum. For this reason, Option B is the correct answer. If the `count()` method had used `Files.walk()` instead of `Files.lines()`, then the class would still compile and run, and Option A would be the correct answer. Note that we had to wrap `Files.lines()` in a try-catch block because using this method directly within a lambda expression without a try-catch block leads to a compilation error.

## Chapter 20: Java Concurrency

- 1.** A. The `ExecutorService` interface defines the two `submit()` methods shown in Options C and D. Because `ExecutorService` extends `Executor`, it inherits the `execute(Runnable)` method presented in Option B. That leaves us with the correct answer, Option A, because `ExecutorService` does not define nor inherit an overloaded method `execute()` that takes a `Callable` parameter.
- 2.** B. The class compiles and runs without throwing an exception, making the first statement true. The class defines two values that are incremented by multiple threads in parallel. The first `IntStream` statement uses an atomic class to update a variable. Since updating an atomic numeric instance is thread-safe by design, the first number printed is always 10, and the second statement is true. The second `IntStream` statement uses an `int` with the pre-increment operator `(++)`, which is not thread-safe. It is possible two threads could update and set the same value at the same time, a form of race condition, resulting in a value less than 5. For this reason, the third statement is not true. Since only the first two statements are true, Option B is the correct answer.
- 3.** C. Option A is incorrect, although it would be correct if `Executors` was replaced with `ExecutorService` in the sentence. While an instance of `ExecutorService` can be obtained

from the `Executors` class, there is no method in the `Executors` class that performs a task directly. Option B is also incorrect, but it would be correct if `start()` was replaced with `run()` in the sentence. It is recommended that you override the `run()` method, not the `start()` method, to execute a task using a custom Thread class. Option C is correct, and one of the most common ways to define an asynchronous task. Finally, Option D is incorrect because Options A and B are incorrect.

4. D. Trick question! `ExecutorService` does not contain any of these methods. In order to obtain an instance of a thread executor, you need to use the `Executors` factory class. For this reason, Option D is the correct answer. If the question had instead asked which `Executors` method to use, then the correct answer would be Option C. Options A and B do not allow concurrent processes and should not be used with a `CyclicBarrier` expecting to reach a limit of five concurrent threads. Option C, on the other hand, will create threads as needed and is appropriate for use with a `CyclicBarrier`.
5. C. `CopyOnWriteArrayList` makes a copy of the array every time it is modified, preserving the original list of values the iterator is using, even as the array is modified. For this reason, the `for` loop using `copy1` does not throw an exception at runtime. On the other hand, the `for` loops using `copy2` and `copy3` both throw `ConcurrentModificationException` at runtime since neither allows modification while they are being iterated upon. Finally, the `ConcurrentLinkedQueue` used in `copy4` completes without throwing an exception at runtime. For the exam, remember that the `Concurrent` classes order read/write access such that access to the class is consistent across all threads and processes, while the `synchronized` classes do not. Because exactly two of the `for` statements produce exceptions at runtime, Option C is the correct answer.
6. C. Resource starvation is when a single active thread is perpetually unable to gain access to a shared resource. Livelock is a special case of resource starvation, in which two or more active threads are unable to gain access to shared resources, repeating the process over and over again. For these reasons, Option C is the correct answer. Deadlock and livelock are similar, although in a deadlock situation the threads are stuck waiting, rather than being active or performing any work. Finally, a race condition is an undesirable result when two tasks that should be completed sequentially are completed at the same time.
7. B. The class does not compile because the `Future.get()` on line 8 throws a checked `InterruptedException` and `ExecutionException`, neither of which is handled nor declared by the `submitReports()` method. If the `submitReports()` and accompanying `main()` methods were both updated to declare these exceptions, then the application would print `null` at runtime, and Option A would be the correct answer. For the exam, remember that `Future` can be used with `Runnable` lambda expressions that do not have a return value but that the return value is always `null` when completed.
8. A. Options B and C are both proper ways to obtain instances of `ExecutorService`. Remember that `newSingleThreadExecutor()` is equivalent to calling `newFixedThreadPool()` with a value of 1. Option D is the correct way to request a single-threaded `ScheduledExecutorService` instance. The correct answer is Option A. The method `newFixedScheduledThreadPool()` does not exist in the `Executors` class, although there is one called `newScheduledThreadPool()`.

- 9.** A. The code compiles without issue but hangs indefinitely at runtime. The application defines a thread executor with a single thread and 12 submitted tasks. Because only one thread is available to work at a time, the first thread will wait endlessly on the call to `await()`. Since the `CyclicBarrier` requires four threads to release it, the application waits endlessly in a frozen condition. Since the barrier is never reached and the code hangs, the application will never output Ready, making Option A the correct answer. If `newCachedThreadPool()` had been used instead of `newSingleThreadExecutor()`, then the barrier would be reached three times, and Option C would be the correct answer.
- 10.** D. First off, `BlockingDeque` is incorrect since it is an interface, not a class. Next, `ConcurrentLinkedDeque` does support adding elements to both ends of an ordered data structure but does not include methods for waiting a specified amount of time to do so, referred to as blocking. `ConcurrentSkipListSet` is also incorrect, since its elements are sorted and not just ordered, and it does not contain any blocking methods. That leaves the correct answer, Option D. A `LinkedBlockingDeque` includes blocking methods in which elements can be added to the beginning or end of the queue, while waiting at most a specified amount of time.
- 11.** A. The `findAny()` method can return any element of the stream, regardless of whether the stream is serial or parallel. While on serial streams this is likely to be the first element in the stream, on parallel streams the result is less certain. For this reason, Option A is the correct answer. When applied to an ordered stream, the rest of the methods always produce the same results on both serial and parallel streams. For this reason, these operations can be costly on a parallel stream since it has to be forced into a serial process.
- 12.** D. The static method `Array.asList()` returns a `List` instance, which inherits the `Collection` interface. While the `Collection` interface defines a `stream()` and `parallelStream()` method, it does not contain a `parallel()` method. For this reason, the second stream statement does not compile, and Option D is the correct answer. If the code was corrected to use `parallelStream()`, then the arrays would be consistently printed in the same order, and Option C would be the correct answer. Remember that the `forEachOrdered()` method forces parallel streams to run in sequential order.
- 13.** D. To start with, the `ForkJoinTask` is the parent class of `RecursiveAction` and `RecursiveTask` and does not contain a `compute()` method, neither abstract nor concrete, making Options A and C automatically incorrect. The `RecursiveTask` class contains the abstract `compute()` method that utilizes a generic return type, while the `RecursiveAction` class contains the abstract `compute()` method that uses a `void` return type. For this reason, Option D is the correct answer.
- 14.** B. An accumulator in a serial or parallel reduction must be associative and stateless. In a parallel reduction, invalid accumulators tend to produce more visible errors, where the result may be processed in an unexpected order. Option A is not associative, since  $(a-b)-c$  is not the same as  $a-(b-c)$  for all values a, b, and c. For example, using values of 1, 2, and 3 results in two different values, -4 and 2. Option C is not stateless, since a class or instance variable `i` is modified each time the accumulator runs. That leaves us with Option B, which is the correct answer since it is both stateless and associative. Even though it ignores the input parameters, it meets the qualifications for performing a reduction.

15. B. The code does not compile because `Callable` must define a `call()` method, not a `run()` method, so Option B is the correct answer. If the code was fixed to use the correct method name, then it would complete without issue, printing `Done!` at runtime, and Option A would be the correct answer.
16. C. Part of synchronizing access to a variable is ensuring that read/write operations are atomic, or happen without interruption. For example, an increment operation requires reading a value and then immediately writing it. If any thread interrupts this process, then data could be lost. In this regard, Option C shows proper synchronized access. Thread 2 reads a value and then writes it without interruption. Thread 1 then reads the new value and writes it. The rest of the answers are incorrect because one thread writes data to the variable in-between another thread reading and writing to the same variable. Because a thread is writing data to a variable that has already been written to by another thread, it may set invalid data. For example, two increment operations running at the same time could result in one of the increment operations being lost.
17. D. The code compiles and runs without issue. The two methods `hare()` and `tortoise()` are nearly identical, with one calling `invokeAll()` and the other calling `invokeAny()`. The key is to know that both methods operate synchronously, waiting for a result from one or more tasks. Calling the `invokeAll()` method causes the current thread to wait until all tasks are finished. Since each task is one second long and they are being executed in parallel, the `hare()` method will take about one second to complete. The `invokeAny()` method will cause the current thread to wait until at least one task is complete. Although the result of the first finished thread is often returned, it is not guaranteed. Since each task takes one second to complete, though, the shortest amount of time this method will return is after one second. In this regard, the `tortoise()` method will also take about one second to complete. Since both methods take about the same amount of time, either may finish first, causing the output to vary at runtime and making Option D the correct answer. Note that after this program prints the two strings, it does not terminate, since the `ExecutorService` is not shut down.
18. B. `ConcurrentSkipListMap` implements the `SortedMap` interface, in which the keys are kept sorted, making Option B the correct answer. While the other answers define ordered data structures, none are guaranteed to be sorted. Remember, if you see `SkipList` as part of a concurrent class name, it means it is sorted in some way, such as a sorted set or map.
19. D. The `synchronized` block used in the `getQuestion()` method requires an object to synchronize on. Without it, the code does not compile, and Option D is the correct answer. What if the command was fixed to synchronize on the current object, such as using `synchronized(this)?` Each task would obtain a lock for its respective object, then wait a couple of seconds before requesting the lock for the other object. Since the locks are already held, both wait indefinitely, resulting in a deadlock. In this scenario, Option A would be the correct answer since a deadlock is the most likely result. We say most likely because even with corrected code, a deadlock is not guaranteed. It is possible, albeit very unlikely, for the JVM to wait five seconds before starting the second task, allowing enough time for the first task to finish and avoiding the deadlock completely.

- 20.** B. The `ScheduledExecutorService` does not include a `scheduleAtFixedDelay()` method, so Option A is incorrect. The `scheduleAtFixedRate()` method creates a new task for the associated action at a set time interval, even if previous tasks for the same action are still active. In this manner, it is possible multiple threads working on the same action could be executing at the same time, making Option B the correct answer. On the other hand, `scheduleWithFixedDelay()` waits until each task is completed before scheduling the next task, guaranteeing at most one thread working on the action is active in the thread pool.
- 21.** D. The application compiles, so Option B is incorrect. The `stroke` variable is thread-safe in the sense that no write is lost since all writes are wrapped in a `synchronized` method, making Option C incorrect. Even though the method is thread-safe, the value of `stroke` is read while the threads may still be executing. The result is it may output 0, 1000, or anything in-between, making Option D the correct answer. If the `ExecutorService` method `awaitTermination()` is called before the value of `stroke` is printed and enough time elapses, then the result would be 1000, and Option A would be the correct answer.
- 22.** B. A race condition is an undesirable result when two tasks that should be completed sequentially are completed at the same time. The result is often corruption of data in some way. If two threads are both modifying the same `int` variable and there is no synchronization, then a race condition can occur with one of the writes being lost. For this reason, Option B is the correct answer. Option A is the description of resource starvation. Options C and D are describing livelock and deadlock, respectively.
- 23.** A. The code compiles, so Option C is incorrect. The application attempts to count the elements of the `sheep` array, recursively. For example, the first two elements are totaled by one thread and added to the sum of the remainder of the elements in the array, which is calculated by another thread. Unfortunately, the class contains a bug. The `count` value is not marked `static` and not shared by all of the `CountSheep` subtasks. The value of `count` printed in the `main()` menu comes from the first `CountSheep` instance, which does not modify the `count` variable. The application prints 0, and Option A is the correct answer. If `count` was marked `static`, then the application would sum the elements correctly, printing 10, and Option B would be the correct answer.
- 24.** D. First off, certain stream operations, such as `limit()` or `skip()`, force a parallel stream to behave it a serial manner, so Option A is incorrect. Option B is also incorrect. Although some operations could take less time to execute, there is no guarantee any operation will actually be faster. For example, the JVM may only allocate a single thread to a parallel stream. In other words, parallel streams may improve performance but do not guarantee it. Option C is incorrect because parallel stream operations are not synchronized. It is up to the developer to provide synchronization or use a concurrent collection if required. Finally, Option D is the correct answer. The `BaseStream` interface, which all streams inherit, includes a `parallel()` method. Of course, the results of an operation may change in the presence of a parallel stream, such as when a stateful lambda expression is used, but they all can be made parallel.

- 25.** A. The code compiles and runs without issue. The JVM will fall back to a single-threaded process if all of the conditions for performing the parallel reduction are not met. The stream used in the `main()` method is not parallel, but the `groupingByConcurrent()` method can still be applied without throwing an exception at runtime. Although performance will suffer from not using a parallel stream, the application will still process the results correctly. Since the process groups the data by year, Option A is the correct answer.
- 26.** A. The code compiles and runs without issue. The three-argument `reduce()` method returns a generic type, while the one-argument `reduce()` method returns an `Optional`. The `concat1()` method is passed an identity "a", which it applies to each element, resulting in the reduction to `aCatAhat`. The lambda expression in the `concat2()` method reverses the order of its inputs, leading to a value of `HatCat`. Therefore, Option A is the correct answer.
- 27.** A. The code compiles without issue, so Options B and C are incorrect. The `f1` declaration uses the version of `submit()` in `ExecutorService`, which takes a `Runnable` and returns a `Future<?>`. The call `f1.get()` waits until the task is finished and always returns `null`, since `Runnable` expressions have a `void` return type. The `f2` declaration uses an overloaded version of `submit()`, which takes a `Callable` expression and returns a generic `Future` object. Since the `double` value can be autoboxed to a `Double` object, the line compiles without issue with `f2.get()` returning `3.14159`. For these reasons, Option A is the correct answer. Option D is incorrect because no exception is expected to be thrown at runtime.
- 28.** C. The class compiles without issue, making Options A and D incorrect. The class attempts to create a synchronized version of a `List<Integer>`. The `size()` and `addValue()` help synchronize the read/write operations. Unfortunately, the `getValue()` method is not synchronized so the class is not thread-safe, and Option C is the correct answer. It is possible that one thread could add to the data object while another thread is reading from the object, leading to an unexpected result. Note that the synchronization of the `size()` method is valid, but since `ThreadSafeList.class` is a shared object, this will synchronize all instances of the class to the same object. This could result in a substantial performance cost if enough threads are creating `ThreadSafeList` objects.
- 29.** D. The post-decrement operator `(--)` decrements a value but returns the original value. It is equivalent to the atomic `getAndDecrement()` method. The pre-increment operator `(++)` increments a value and then returns the new value. It is equivalent to the `incrementAndGet()` atomic operation. For these reasons, Option D is the correct answer.
- 30.** B. When a `CyclicBarrier` goes over its limit, the barrier count is reset to zero. The application defines a `CyclicBarrier` with a barrier limit of 5 threads. The application then submits 12 tasks to a cached executor service. In this scenario, a cached thread executor will use between 5 and 12 threads, reusing existing threads as they become available. In this manner, there is no worry about running out of available threads. The barrier will then trigger twice, printing five 1s for each of the sets of threads, for a total of ten 1s. For this reason, Option B is the correct answer. The application then hangs indefinitely, as discussed in the next question.

- 31.** D. The application does not terminate successfully nor produce an exception at runtime, making Options A and B incorrect. It hangs at runtime because the `CyclicBarrier` limit is five, while the number of tasks submitted and awaiting activation is 12. This means that 2 of the tasks will be left over, stuck in a deadlocked state waiting for the barrier limit to be reached but with no more tasks available to trigger it. For this reason, Option D is the correct answer. If the number of tasks was a multiple of the barrier limit, such as 10 instead of 12, then the application will still hang because the `ExecutorService` is never shut down. The `isShutdown()` in the application finally block does not trigger a shutdown. Remember that it is important to shut down an `ExecutorService` after you are finished with it, else it can prevent a program from terminating. In this case, Option C would be the correct answer.
- 32.** C. The code does not compile because the blocking methods `offerLast()` and `pollFirst()` each throw a checked `InterruptedException` that are not handled by the lambda expressions, so Option C is the correct answer. If the lambda expressions were wrapped with try-catch blocks, then the process would first add all items to the queue, then remove them all of them, resulting in an output of 0. In this case, Option A would be the correct answer. Even though the tasks are completed in parallel, each stream does not terminate until all tasks are done. Note that 10 seconds is more than enough time under normal circumstances to add/remove elements from the queue.
- 33.** A. First of all, the for loops using `copy1` and `copy4` both throw `ConcurrentModificationException` at runtime since neither allows modification while they are being iterated upon. Next, `CopyOnWriteArrayList` makes a copy of the array every time it is modified, preserving the original list of values the iterator is using, even as the array is modified. For this reason, the for loop using `copy2` completes without throwing an exception or creating an infinite loop. Finally, the `ConcurrentLinkedDeque` used in `copy3` completes without producing an exception or infinite loop. The Concurrent collections order read/write access such that access to the class is consistent across all threads and processes, even iterators. Because the values are inserted at the head of the queue using `push()` and the underlying data structure is ordered, the new values will not be iterated upon and the loop finishes. Since none of the for statements produce an infinite loop at runtime, Option A is the correct answer. If `push()` had been used instead of `offer()` in the third loop, with new values being inserted at the tail of the queue instead of at the head, then the for loop would have entered an infinite loop, and Option B would be the correct answer.
- 34.** B. Options A, C, and D are the precise requirements for Java to perform a concurrent reduction using the `collect()` method, which takes a `Collector` argument. Recall from your studies that a `Collector` is considered concurrent and unordered if it has the `Collector.Characteristics` enum values `CONCURRENT` and `UNORDERED`, respectively. Option B is the correct answer because elements of a stream are not required to implement `Comparable` in order to perform a parallel reduction.
- 35.** D. The class compiles and runs without issue, making Options A and B incorrect. The purpose of the fork/join framework is to use parallel processing to complete subtasks across multiple threads concurrently. Unfortunately, calling the `compute()` method inside of an existing `compute()` does not spawn a new thread. The result is that this task is completed using a single thread, despite a pool of threads being available. For this reason, Option D is the correct answer. In order to properly implement the fork/join framework, the `compute()` method would need to be rewritten. The `f1.compute()` call should be replaced with `f1.fork()` to spawn a separate task, followed by `f2.compute()` to process the data on the

current thread, and ending in `f1.join()` to retrieve the results of the first task completed while `f2.compute()` was being processed. If the code was rewritten as described, then Option C would be the correct answer.

36. D. The `shutdown()` method prevents new tasks from being added but allows existing tasks to finish. In addition to preventing new tasks from being added, the `shutdownNow()` method also attempts to stop all running tasks. Neither of these methods guarantee any task will be stopped, making Option D the correct answer. Option C is incorrect because there is no `halt()` method in `ExecutorService`.
37. B. First off, the class uses a `synchronized` list, which is thread-safe and allows modification from multiple threads, making Option D incorrect. The process generates a list of numbers from 1 to 5 and sends them into a parallel stream where the `map()` is applied, possibly out of order. This results in elements being written to `db` in a random order. The stream then applies the `forEachOrdered()` method to its elements, which will force the parallel stream into a single-threaded state. At runtime, line `p1` will print the results in order every time as 12345. On the other hand, since the elements were added to `db` in a random order, the output of line `p2` is random and cannot be predicted ahead of time. Since the results may sometimes be the same, Option B is the correct answer. Part of the reason that the results are indeterminate is that the question uses a stateful lambda expression, which based on your studies should be avoided in practice!
38. C. The program compiles and does not throw an exception at runtime, making Options B and D incorrect. The class attempts to add and remove values from a single `cookies` variable in a thread-safe manner but fails to do so because the methods `deposit()` and `withdrawal()` synchronize on different objects. The instance method `deposit()` synchronizes on the `bank` object, while the `static` method `withdrawal()` synchronizes on the `static Bank.class` object. Even though method calls of the same type are protected, calls across the two different methods are not. Since the compound assignment operators `(+=)` and `(-=)` are not thread-safe, it is possible for one call to modify the value of `cookies` while the other is already operating on it, resulting in a loss of information. For this reason, the output cannot be predicted, and Option C is the correct answer. If the two sets of calls were properly synchronized on the same object, then the `cookies` variable would be protected from concurrent modifications, and Option A would be the correct answer.
39. A. The code attempts to search for a matching element in an array recursively. While it does not contain any compilation problems, it does contain an error. Despite creating `Thread` instances, it is not a multi-threaded program. Calling `run()` on a `Thread` runs the process as part of the current thread. To be a multi-threaded execution, it would need to instead call the `start()` method. For this reason, the code completes synchronously, waiting for each method call to return before moving on to the next and printing `true` at the end of the execution, making Option A the correct answer. On the other hand, if `start()` had been used, then the application would be multi-threaded but then the result may not be ready by the time the `println()` method is called, resulting in a value that cannot be predicted ahead of time. In this case, Option D would be the correct answer.
40. C. Line 13 does not compile because the `execute()` method has a return type of `void`, not `Future`. Line 15 does not compile because `scheduleAtFixedRate()` requires four arguments that include an initial delay and period value. For these two reasons, Option C is the correct answer.

# Chapter 21: Building Database Applications with JDBC

1. C. `Connection` is an interface for communicating with the database. `Driver` is tricky because you don't write code that references it directly. However, you are still required to know it is a JDBC interface. `DriverManager` is used in JDBC code to get a `Connection`. However, it is a concrete class rather than an interface. Since `Connection` and `Driver` are JDBC interfaces, Option C is correct.
2. D. Database-specific implementation classes are not in the `java.sql` package. The implementation classes are in database drivers and have package names that are specific to the database. Therefore, Option D is correct. The `Driver` interface is in the `java.sql` package. Note that these classes may or may not exist. You are not required to know the names of any database-specific classes, so the creators of the exam are free to make up names.
3. D. All JDBC URLs begin with the protocol `jdbc` followed by a colon as a delimiter. Option D is the only one that does both of these, making it the answer.
4. A. The `Driver` interface is responsible for getting a connection to the database, making Option A the answer. The `Connection` interface is responsible for communication with the database but not making the initial connection. The `Statement` interface knows how to run the SQL query, and the `ResultSet` interface knows what was returned by a `SELECT` query.
5. B. The requirement to include a `java.sql.Driver` file in the driver jar file was introduced in JDBC 4.0. A 3.0 driver is allowed, but not required, to include this file. JDBC 3.0 also requires a call to `Class.forName()`. As a result, Option B best fills in the blanks.
6. C. `Connection` is an interface. Since interfaces do not have constructors, Option D is incorrect. The `Connection` class doesn't have a `static` method to get a `Connection` either, making Option A incorrect. The `Driver` class is also an interface without `static` methods, making Option B incorrect. Option C is the answer because `DriverManager` is the class used in JDBC to get a `Connection`.
7. B. The `DriverManager.getConnection()` method can be called with just a URL. It is also overloaded to take the URL, username, and password, making Option B correct.
8. D. `CallableStatement` and `PreparedStatement` are interfaces that extend the `Statement` interface. You don't need to know that for the exam. You do need to know that a database driver is required to provide the concrete implementation class of `Statement` rather than the JDK. This makes Option D correct. Note that while Derby is provided with Java, it is in a separate jar from the “main” JDK.
9. C. A JDBC URL has three components separated by colons. All three of these URLs meet those criteria. For the data after the component, the database driver specifies the format. Depending on the driver, this might include an IP address and port. Regardless, it needs to include the database name or alias. I and II could both be valid formats because they

mention the database box. However, III only has an IP address and port. It does not have a database name or alias. Therefore III is incorrect and Option C correct.

10. C. The requirement to include a `java.sql.Driver` file in the driver jar was introduced in JDBC 4.0. A call to `Class.forName()` was made optional with JDBC 4.0. As a result, Option C best fills in the blanks.
11. A. `Scroll sensitive` is a result set type parameter, and `updatable` is a concurrency mode. The result set type parameter is passed to `createStatement()` before the concurrency mode. If you request options that the database driver does not support, it downgrades to an option it does support rather than throwing an exception. Statements I and III are correct, making Option A the answer.
12. B. JDBC 4.0 allows, but does not require, a call to the `Class.forName()` method. However, since it is in the code, it needs to be correct. This method is expecting a fully qualified class name of a database driver, not the JDBC URL. As a result, the `Class.forName()` method throws a `ClassNotFoundException`, and Option B is the answer.
13. B. There are two `ResultSet` concurrency modes: `CONCUR_READ_ONLY` and `CONCUR_UPDATABLE`. All database drivers support read-only result sets, but not all support updatable ones. Therefore, Option B is correct.
14. D. This code is missing a call to `rs.next()`. As a result, `rs.getInt(1)` throws a `SQLException` with the message `Invalid cursor state - no current row`. Therefore, Option D is the answer.
15. D. The `execute()` method is allowed to run any type of SQL statements. The `executeUpdate()` method is allowed to run any type of the SQL statement that returns a row count rather than a `ResultSet`. Both `DELETE` AND `UPDATE` SQL statements are allowed to be run with either `execute()` or `executeUpdate()`. They are not allowed to be run with `executeQuery()` because they do not return a `ResultSet`. Therefore, Option D is the answer.
16. C. `Connection` is an interface rather than a concrete class. Therefore, it does not have a constructor and line `s2` does not compile. As a result, Option C is the answer. Option A would be the answer if the code `new Connection()` was changed to `DriverManager.getConnection()`.
17. A. There are three `ResultSet` type options: `TYPE_FORWARD_ONLY`, `TYPE_SCROLL_INSENSITIVE`, and `TYPE_SCROLL_SENSITIVE`. Only one of these is in the list, making Option A correct.
18. B. Unlike arrays, JDBC uses one-based indexes. Since `num_pages` is in the second column, the parameter needs to be 2, ruling out Options A and C. Further, there is not a method named `getInteger()` on the `ResultSet` interface, ruling out Option D. Since the proper method is `getInt()`, Option B is the answer.
19. D. Option A does not compile because you have to pass a column index or column name to the method. Options B and C compile. However, there are not columns named 0 or 1. Since these column names don't exist, the code would throw a `SQLException` at runtime. Option D is correct as it uses the proper column name.

- 20.** B. The parameters to `createStatement()` are backward. However, they still compile because both are of type `int`. This means the code to create the `Statement` does compile, and Option A is incorrect. Next comes the code to create the `ResultSet`. While both `execute()` and `executeQuery()` can run a `SELECT` SQL statement, they have different return types. Only `executeQuery()` can be used in this example. The code does not compile because the `execute()` method returns a `boolean`, and Option B is correct. If this was fixed, Option D would be the answer because `rs.next()` is never called.
- 21.** D. Since this code opens `Statement` using a try-with-resources, `Statement` gets closed automatically at the end of the block. Further, closing a `Statement` automatically closes a `ResultSet` created by it, making Option D the answer. Remember that you should close any resources you open in code you write.
- 22.** C. Option A is incorrect because `Driver` is an interface while `DriverManager` is a concrete class. The inverse isn't true either; `DriverManager` doesn't implement `Driver`. Option B is incorrect because the `Connection` implementation comes from the database driver jar. Option C is correct. You can turn off auto-commit mode, but it defaults to on. Option D is incorrect because you need to call `rs.next()` or an equivalent method to point to the first row.
- 23.** C. The requirement to include a `java.sql.Driver` file in the `META-INF` directory was introduced in JDBC 4.0. Older drivers are not required to provide it, making Option B incorrect. A file named `jdbc.driver` has never been a requirement. Option A is incorrect and is simply here to trick you. All drivers are required to implement the `Connection` interface, making Option C the answer.
- 24.** D. First, `rs.next()` moves the cursor to point to the first row, which contains the number 10. Line `q1` moves the cursor to immediately before the first row. This is the same as the position it was in before calling `rs.next()` in the first place. It is a valid position but isn't a row of data. Line `q2` tries to retrieve the data at this position and throws a `SQLException` because there isn't any data, making Option D the answer.
- 25.** B. This code shows how to properly update a `ResultSet`. Note that it calls `updateRow()` so the changes get applied in the database. This allows the `SELECT` query to see the changes and output 10. Option B is correct. Remember that unlike this code, you should always close a `ResultSet` when you open it in real code.
- 26.** C. There is no `ResultSet` method named `prev()`. Therefore, the code doesn't compile, and Option C is correct. If `prev()` was changed to `previous()`, the answer would be Option B because `updateRow()` is never called. Remember that unlike this code, you should always close a `ResultSet` when you open it in real code.
- 27.** D. While the code turns off automatic committing, there is a `commit()` statement after the first two inserts that explicitly commits those to the database. Then automatic commit is turned back on and the third commit is made, making Option D the answer.
- 28.** A. The `count(*)` function in SQL always returns a number. In this case, it is the number zero. This means line `r1` executes successfully because it positions the cursor at that row. Line `r2` also executes successfully and prints 0, which is the value in the row. Since the code runs successfully, Option A is the answer.

29. B. The cursor starts out at position zero, right before the first row. Line 6 moves the cursor to position five. Line 7 tries to move the cursor ten rows before that position which is row negative five. Since you can't move back before row zero, the cursor is at row zero instead. Then line 8 moves the cursor forward five positions from row zero, leaving it at row five and making Option B the answer.
30. C. JDBC 4.0 allows, but does not require, a call to the `Class.forName()` method. Since the database does not exist, `DriverManager.getConnection()` throws a `SQLException`, and Option C is the answer.
31. D. When running a query on a `Statement`, Java closes any already open `ResultSet` objects. This means that `rs1` is closed on line 8. Therefore, it throws a `SQLException` on line 9 because we are trying to call `next()` on a closed `ResultSet`, and Option D is correct.
32. B. The code turns off automatic committing, so the inserts for red and blue are not immediately made. The `rollback()` statement actually prevents them from being committed. Then automatic commit is turned back on and one insert is made, making Option B the answer.
33. A. This code correctly obtains a `Connection` and `Statement`. It then runs a query, getting back a `ResultSet` without any rows. The `rs.next()` call returns `false`, so nothing is printed, making Option A correct.
34. B. Since the `ResultSet` type allows scrolling, the code does not throw a `SQLException` at runtime. Immediately after getting the `ResultSet`, the cursor is positioned at the end immediately after Scott's row. The next two lines try to move forward one row. This has no effect since the cursor is already at the end. Then `previous()` moves the cursor to point to the last row, which is Scott's row. The second `previous()` call moves the cursor up one more row to point to Elena's row, making Option B the answer.
35. B. When passing a negative number to `absolute()`, Java counts from the end instead of the beginning. The last row is Scott's row, so the first print statement outputs Scott. When passing a positive number to `absolute()`, Java counts from the beginning, so Jeanne is output. Therefore, Option B is correct.
36. D. When creating the `Statement`, the code doesn't specify a result set type. This means it defaults to `TYPE_FORWARD_ONLY`. The `absolute()` method can only be called on scrollable result sets. The code throws a `SQLException`, making Option D the answer.
37. B. This code does not compile because the `ResultSet` options need to be supplied when creating the `Statement` object rather than when executing the query. Since the code does not compile, Option B is correct.
38. B. The code turns off automatic committing, so the inserts for red and blue are not immediately made. The `rollback()` statement says to prevent any changes made from occurring. This gets rid of red and blue. Then automatic commit is turned back on and the one insert for green is made. The final `rollback` has no effect since the commit was automatically made. Since there was one row added, Option B is the answer.

- 39.** D. Line 18 doesn't compile because `beforeFirst()` has a `void` return type. Since the code doesn't compile, it doesn't print `true` at all, and Option D is correct. If line 18 called `rs.beforeFirst()` without trying to print the result, Option B would be the answer. All the other statements are valid and return `true`.
- 40.** B. When manually closing database resources, they should be closed in the reverse order from which they were opened. This means that the `ResultSet` object is closed before the `Statement` object and the `Statement` object is closed before the `Connection` object. This makes Option B the answer.

## Chapter 22: Localization

- 1.** D. Oracle defines a locale as a geographical, political, or cultural region. Time zones often span multiple locales, so Option D is correct.
- 2.** C. Currencies vary in presentation by locale. For example, `9,000` and `9.000` both represent nine thousand, depending on the locale. Similarly, for dates, `01-02-1991` and `02-01-1991` represent January 2, 1991, depending on the locale. This makes Option C the answer.
- 3.** C. The `Locale` object provides `getDefault()` and `setDefault()` methods for working with the default locale, so Option C is correct. There is no `get()` method declared on `Locale`.
- 4.** A. Internationalization means the program is designed so it can be adapted for multiple languages. By extracting the town names, this is exactly what has happened here, making Option A correct. Localization means the program actually supports multiple locales. There's no mention of multiple locales here, so Option B is incorrect. Similarly, there is no mention of multiple languages, making Option D incorrect. Finally, specialization is not a term relevant to properties, making Option C incorrect.
- 5.** A. The `Properties` class is a `Map`, making III correct. `Hashtable` and `HashMap` are concrete classes rather than interfaces, so I and II are incorrect. While a `Properties` object is a `Hashtable`, this is not an interface. Since only III is correct, Option A is the answer.
- 6.** C. Java supports properties file resource bundles and Java class resource bundles. Properties file resource bundles contain `String` keys and `String` values. Java class resource bundles contain `String` keys and any type of classes as values. Since both are valid, Option C is correct.
- 7.** B. Calling `Locale.setDefault()` changes the default locale within the program. It does not change any settings on the computer. The next time you run a Java program, it will have the original default locale rather than the one you changed it to.
- 8.** B. Line 18 prints the value for the property with the key `mystery`, which is `bag`. Line 19 prints a space. Line 20 doesn't find the key `more` so it prints `null`. Therefore, it prints `bag null`, and Option B is correct.

9. C. There is not a built-in class called `JavaResourceBundle`, making Options A and B incorrect. The `ListResourceBundle` class is used to programmatically create a resource bundle. It requires one method to be implemented named `getContents()`, making Option D incorrect and Option C correct. This method returns a 2D array of key/value pairs.
10. A. When both a language and country code are present, the language code comes first. The language code is in all lowercase letters and the country code is in all uppercase.
11. C. Java starts out by looking for a Java file with the most specific match, which is language and country code. Since this is happening at runtime, it is looking for the corresponding file with a `.class` extension. This matches Option C, making it the answer. If this file was not found, Java would then look for a `.properties` file with the name, which is Option D. If neither was found, it would continue dropping components of the name, eventually getting to Options A and B in that order.
12. A. This class correctly creates a Java class resource bundle. It extends `ListResourceBundle` and creates a 2D array as the property contents. Since `count` is an `int`, it is autoboxed into an `Integer`. In the `main()` method, it gets the resource bundle without a locale and requests the `count` key. Since `Integer` is a Java `Object`, it calls `get0bject()` to get the value. The value is not incremented each time because the `getContents()` method is only called once. Therefore, Option A is correct.
13. A. A `Locale` can consist of a language only, making Option A correct. It cannot consist of a country code without a language, so Option B is incorrect. Finally, if both a language and country code are present, the language code is first, making Option C incorrect.
14. A. Java supports properties file resource bundles and Java class resource bundles. Properties file resource bundles require `String` values, making Option B incorrect. Java class resource bundles allow any type of classes as values. Since the question asks about defining values, it is the `.java` source code rather than the `.class` bytecode file, making Option A the answer.
15. C. At least one matching resource bundle must be available at the time of the call to `getBundle()`. While the requested key determines which of the resource bundles is used, at least one must exist simply to get the `ResourceBundle` reference, so Option C is the answer.
16. D. The `Properties` class implements `Map`. While the `get()` method, inherited from the superclass, is available, it returns an `Object`. Since `Object` cannot be cast to `String`, it does not compile, and Option D is the answer.
17. D. Java supports properties file resource bundles and Java class resource bundles. Both require `String` as the key format, so Option D is the answer.
18. A. Java starts out by looking for a Java file with the most specific match, which is language and country code. Since there is no such matching file, it drops the country code and looks for a match by language code. Java looks for bytecode files before properties files. Therefore, Option A is the answer. If it wasn't present, Option B would be the next choice. Options C and D would never be considered, as a locale doesn't just have a country code.

- 19.** D. There is no `get()` method on `Locale`. You need to use a constructor or a predefined `Locale` constant to obtain a `Locale` object. Therefore, Option D is the correct answer. Option B is close in that `Locale.ITALIAN` does reference a `Locale` object. However, it should not be passed to the nonexistent `get()` method.
- 20.** A. This code creates a `Map` with two elements. Then it copies both key/value pairs to a `Properties` object. This works because a `Properties` object is also a `Map` and therefore has a `put()` method that takes `Object` parameters. Finally, the code gets the `String` property values of both keys and prints `hammer nail`. Therefore, Option A is correct.
- 21.** D. The code attempts to create a Java class resource bundle. However, the `Type` class does not extend `ListResourceBundle`. It compiles, but throws an exception on line 9 because no resource bundle is actually defined. Therefore, Option D is correct.
- 22.** D. This code compiles and runs without exception, making Option D the answer. Line 3 uses a predefined `Locale` constant. Line 5 passes a language and country code for English in Australia. Line 7 incorrectly passes capital letters as a language code. However, Java automatically converts it to lowercase without throwing an exception. The three lines printed by the code are `ko, en_US` and `en`.
- 23.** C. Line 18 prints the value for the property with key `mystery`, which is `bag`. Line 19 prints a space. Line 20 doesn't find the key `more` so it uses the second parameter `?` as the default value. The code prints `bag ?`, so Option C is correct.
- 24.** B. The `getBundle()` method matches `Cars_fr_FR.properties` since that is the requested locale. The `country` key is in that properties file directly, so `France` is used as the value. The `engine` key is not, so Java looks higher up in the hierarchy and finds it in the language-specific `Cars_fr.properties` file and uses `moteur` as the value. Therefore, Option B is correct. Note that the default locale isn't used.
- 25.** D. The `getBundle()` method matches `Cars_fr.properties`. Since the requested locale of French Canada is not available, it uses the language-specific locale of French. The `engine` key is in that properties file directly, so `moteur` is retrieved as the value. However, we have a problem getting the `horses` key. It is not in the hierarchy of `Cars_fr.properties`. It is in the English properties file, but Java cannot look at the default locale if it found a match with the requested locale. As a result, the code throws a `MissingResourceException`, making Option D the answer.
- 26.** A. The `getBundle()` method matches `Cars_en.properties`. Since the requested locale of English Canada is not available, it uses the language-specific locale of English. The `engine` key is in that properties file directly, so `engine` is retrieved as the value. The `horses` key is also in that properties file, so `241` is used as the value, and Option A is the answer.
- 27.** B. A `ListResourceBundle` is a Java class that provides key/value pairs. The values can be any Java class type and can be created at runtime, making Options A and C false statements. When you want to provide a language-specific resource bundle, an underscore and the language code are added after the class name. Since Option B does not have an underscore, it is a false statement. Since all three statements are false, Option D is the answer.

- 28.** D. Since a `Locale` is passed when requesting the `ResourceBundle`, that `Locale` is used first when looking for bundles. Since there isn't a bundle called `Colors_zh_CN.properties`, Java goes on to check for the language. Option D provides a match on language. If this was not found, Java would go on to the default locale, eventually matching Option B. Since country is not used without language, Options A and C would not be considered as options.
- 29.** B. This class correctly creates a Java class resource bundle. It extends `ListResourceBundle` and creates a 2D array as the property contents. In the `main()` method, it gets the resource bundle without a locale and requests the `count` key. Since this is a Java `Object`, it calls `get0bject()` to get the value and casts it to the correct type. Then the `getCount()` method is called twice, incrementing each time, making Option B the correct answer. Note that having a mutable object as a property is a bad practice.
- 30.** D. Line 10 is incorrect. It tries to get a bundle named `Type`. However, this code is in a package and named `keyboard.Type`. Therefore, a `MissingResourceException` is thrown, and Option D is correct.
- 31.** B. Since no locale is specified, the code tries to find a bundle matching the default locale. While none of the resource bundles match English United States, two do match the language English. The Java class one is used since it is present. Since the Java resource bundle for English doesn't have a key `wheels`, we go up to the parent resource bundle. The default Java resource bundle does have the key `wheels` with the value 4, so Option B is correct.
- 32.** D. Since no locale is specified, the code tries to find a bundle matching the default locale. Two resource bundles match the language English. The Java class one is used since it is present. However, it does not contain a key `color`, nor does its parent. Java does not allow looking in a properties file resource bundle once it has matched a Java class resource bundle. Therefore, it throws a `MissingResourceBundleException`, and Option D is the answer.
- 33.** B. Since no locale is specified, the code tries to find a bundle matching the default locale. None of the resource bundles match the language, so the default resource bundle is used. The default Java resource bundle is used since it is present and has the key `wheels` with the value 4. Therefore, Option B is correct.
- 34.** A. Since a `Locale` is passed when requesting the `ResourceBundle`, that `Locale` is used first when looking for bundles. Since there isn't a bundle for that `Locale`, Java checks for the language `zh`. There still isn't a match, so Java goes on to check for the default `Locale`. Still no match. Next Java checks the language of the default `Locale` and finally finds a match in Option A. If that match wasn't found, Java would match on Option B, which is the default bundle. Option C would never be checked since the base name is `Red` rather than the requested `Color`.
- 35.** C. A `Properties` object implements `Map`. This makes the `get()` method available. `Properties` also defined a `getProperty()` method. Therefore, both methods can fill in the blank, and Option C is correct.

- 36.** B. This code compiles and runs without issue. It creates a default Java class resource bundle. Lines 5 through 7 show it has one key and one `ArrayList` value. Line 9 gets a reference to the resource bundle. Lines 10 through 16 retrieve the `ArrayList` and add six values to it. Since this is a reference, line 17 gets the same object and prints the size of 6. Therefore, Option B is correct.
- 37.** B. The class on line `p1` should be `Properties` rather than `Property`. As written it is incorrect and does not compile, making Option B the answer.
- 38.** C. `ResourceBundle` is an abstract class and uses a factory to obtain the right subclass. Since a call to the constructor `new ResourceBundle()` does not compile, Option C is the answer. If this was fixed, Option A would be the answer because `getContents()` is only called once.
- 39.** A. This code sets the default locale to English and then tries to get a resource bundle for `container` in the `pod` package. It finds the resource bundle `pod.container_en.properties` as the most specific match. Both keys are found in this file, so Option A is the answer.
- 40.** D. Option C is not a valid match for this resource bundle because the base name is `Red` rather than the requested `Colors`. Options A and B are not valid matches because they contain uppercase letters for the language code while Java requires lowercase. Since none match, Option D is correct, and the code throws an exception at runtime.

## Chapter 23: OCP Practice Exam

- 1.** A, E. Line 12 has no effect. The cursor starts out positioned immediately before the first row and `beforeFirst()` keeps it there. Line 13 moves the cursor to point to row five and prints `true`. Line 14 prints the value in that row, which is 5. Line 15 tries to subtract 10 rows from the current position. That would be at row negative five. However, the cursor can't go back further than the beginning, so it stays at row zero. It also prints `false` since there isn't data at row zero. Note this is the same position the cursor was at on line 12. Now we have a problem. Line 16 tries to print a value on row zero, but there is no data. It instead throws a `SQLException`, making Option E correct. Option A is also correct since `true` was only output once.
- 2.** D. While this code misuses formatting characters, it does compile and run successfully, making Options E and F incorrect. A lowercase `m` represents the minutes, which are 59 in this case, ruling out Options A and B. The rest of the code prints the date, hour, and month. This gives the value `59.140103`, making Option D the answer.
- 3.** A, B, E. First off, Option A is a valid functional interface that matches the `Runnable` functional interface. Option B is also a valid lambda expression that matches `Function<Double,Double>`, among other functional interfaces. Option C is incorrect because the local variable `w` cannot be declared again in the lambda expression body since it is already declared in the lambda expression. Option D is also incorrect. If the data type is specified for one variable in a lambda expression, it must be specified for all variables

within the expression. Next, Option E is correct because this lambda expression matches the `UnaryOperator` functional interface. Lastly, Option F is incorrect. The statement `name.toUpperCase()` is missing a semicolon (`;`) that is required to terminate the statement.

4. D. The code does not compile, so Options A, B, and F are incorrect. The first compilation error is in the declaration of the lambda expression for `second`. It does not use a generic type, which means `t` is of type `Object`. Since `Object`, unlike `String`, does not have a method `equalsIgnoreCase()`, the lambda expression does not compile. The second compilation issue is in the lambda expression in the `main()` method. Notice that `process()` takes an `ApplyFilter` instance, and `ApplyFilter` is a functional interface that takes a `List<String>` object. For this reason, `q` in this lambda expression is treated as an instance of `List<String>`. The `forEach()` method defined in `Collections` requires a `Consumer` instance, not a `Function`, so the call `q.forEach(first)` does not compile. For these two reasons, Option D is the correct answer, since the rest of the code compiles without issue.
5. B. First, remember that you are supposed to assume missing imports are present so you can act as if `java.util` and `java.util.stream` are imported. This code does compile. Line `r1` is a valid lambda definition of a `Comparator`. Line `r2` is valid code to sort a stream in a descending order and print the values. Therefore, Option B is correct.
6. F. The first catch block on line `p1` does not compile because `AddingException` and `DividingException` are checked exceptions, and the compiler detects these exceptions are not capable of being thrown by the associated `try` block. The second catch block on line `p2` also does not compile, although for a different reason. `UnexpectedException` is a subclass of `RuntimeException`, which in turn extends `Exception`. This makes `UnexpectedException` a subclass of `Exception`, causing the compiler to consider `UnexpectedException` redundant. For these two reasons, the code does not compile, and Option F is the correct answer.
7. A. The code compiles and runs without issue, making Option A the correct answer. Enums are usually named like classes and have enum values that are all uppercase. While a format like `Colors.RED` or `Colors.GREEN` is the common convention, alternate formats like `COLORS.blue` do compile. Next, note the enum properly implements the `HasHue` interface even though there's no enum-level method, with each value having its own implementation. Also, line 10 does not end with a semicolon (`;`). Because there are no methods or constructors defined outside the value list, a semicolon (`;`) is not required. The enum class and the rest of the application compile without issue, printing `Painting: #00FF00` at runtime.
8. C. Java 8 date and time classes are immutable. They use a `static` factory method to get the object reference rather than a constructor. This makes II, IV, and VI incorrect. Further, there is not a `ZonedDateTime` class. There is a `ZonedDateTime` class. This additionally makes V incorrect. Both I and III compile, so Option C is correct.
9. C, F. The `Optional` does not contain a value. While there is a `get()` method on `Optional`, it doesn't take any parameters, making Options A and B incorrect. Option C is the simplest way to print the desired result. The `orElse()` method returns the parameter passed if the `Optional` is empty. The `orElseGet()` method runs the `Supplier` passed as a parameter, making Option F correct as well.

- 10.** B. This class is never instantiated, so the instance initializer never outputs 1 and the constructor never outputs 3. This rules out Options A, D, and E. A static initializer only runs once for the class, which rules out Option C. Option B is correct because the static initializer runs once printing 2, followed by the static method `callMe()` printing 4 twice, and ending with the `main()` method printing 5.
- 11.** C. The `Locale` class has a constructor taking a language code and an optional country code. The `Properties` class is a type of `Map` so it also has a constructor. By contrast, a `ResourceBundle` subclass is typically obtained by calling the `ResourceBundle.getBundle()` method. `ResourceBundle` is an abstract class, so a subclass will get returned like `ListResourceBundle`.
- 12.** E. Option C is clearly incorrect because the class has a `public` constructor so cannot be a singleton. Options A, B, and D are also incorrect because the `height` instance variable is not `private`. This means other classes in the same package can read and change the value. Therefore, Option E is correct.
- 13.** E. The `readObject()` method returns an `Object` instance, which must be explicitly cast to `Cruise` in the second `try-with-resources` statement. For this reason, the code does not compile, and Option E is the correct answer. If the explicit cast was added, the code would compile but throw a `NotSerializableException` at runtime, since `Cruise` does not implement the `Serializable` interface. If both of these issues were corrected, the code would run and print `4, null`. The `schedule` variable is marked `transient`, so it defaults to `null` when deserialized, while `numPassengers` is assigned the value it had when it was serialized. Note that on deserialization, the constructors and instance initializers are not executed.
- 14.** B. `Driver`, `Connection`, `Statement`, and `ResultSet` are the four key interfaces you need to know for JDBC. `DriverManager` is a class rather than an interface. `Query` is not used in JDBC. Since only `Driver` and `ResultSet` are interfaces in the list, Option B is the answer.
- 15.** A, C, F. The `IntUnaryOperator` takes an `int` value and returns an `int` value. Options B and E are incorrect because the parameter types, `Integer` and `long`, respectively, are not compatible. Option B is incorrect because while unboxing can be used for expressions, it cannot be used for parameter matching. Option E is incorrect because converting from `long` to `int` requires an explicit cast. Option D is incorrect because dividing an `int` by a `double` value `3.1` results in `q/3.1` being a `double` value, which cannot be converted to `int` without an explicit cast. The rest of the lambda expressions are valid since they correctly take an `int` value and return an `int` value.
- 16.** D, F. To begin with, the `read()` method of both classes returns an `int` value, making Option A incorrect. As you may recall from your studies, neither use `byte` or `char`, so that `-1` can be returned when the end of the stream is reached without using an existing `byte` or `char` value. Option B is incorrect because neither contain a `flush()` method, while Option C is incorrect because they both contain a `skip()` method. Both `InputStream` and `Reader` are abstract classes, not interfaces, making Option D correct and Option E incorrect. That leaves Option F as a correct answer. Both can be used to read character or `String` data, although `Reader` is strongly recommended, given its built-in support for character encodings.

17. B. There are not `minus()` or `plus()` methods on `ChronoUnit` making Options C, D, E and F incorrect. Both Options A and B compile; however, they differ in the output. Option A prints 1 because you can add 1 to get from November to December. Option B prints -1 because the first date is larger, and therefore Option B is the correct answer.
18. D. The code compiles without issue, making the first statement true and eliminating Option B. It is possible that `System.console()` could return `null`, leading to a `NullPointerException` at runtime and making the third statement true. For this reason, Options A and C are also incorrect. That leaves us with two choices. While the process correctly clears the password from the `char` array in memory, it adds the value to the JVM string pool when it is converted to a `String`. The whole point of using a `char` array is to prevent the password from entering the JVM string pool, where it can exist after the method that called it has finished running. For this reason, the second statement is false, making Option D correct and Option E incorrect.
19. B. Options C and D are incorrect because they print the `Runner` object rather than the `int` value it contains since `peek()` is called before mapping the value. The `Runner` object is something like `Runner@6d03e736`. Option A is incorrect because the `map()` method returns a `Stream<Integer>`. While `Stream<Integer>` does have a `max()` method, it requires a `Comparator`. By contrast, Option B uses `mapToInt()`, which returns an `IntStream` and does have a `max()` method that does not take any parameters. Option B is the only one that compiles and outputs the `int` values.
20. E. The code does not compile, so Options A, B, and F are incorrect. The first compilation error is on line 3, which is missing a required semicolon (`;`) at the end of the line. A semicolon (`;`) is required at the end of any enum value list if the enum contains anything after the list, such as a method or constructor. The next compilation error is on line 4 because enum constructors cannot be `public`. The last compilation error is on line 10. The `case` statement must use an enum value, such as `FALL`, not an `int` value. For these three reasons, Option E is the correct answer.
21. A, C, E. To start with, `bustNow()` now takes a `Double` value, while `bustLater()` takes a `double` value. To be compatible, the lambda expression has to be able to handle both data types. Option A is correct, since the method reference `System.out::print` matches overloaded methods that can take `double` or a `Double` (via unboxing). Option B is incorrect, since `intValue()` works for the `Consumer<Double>`, which takes `Double`, but not `DoubleConsumer`, which takes `double`. For a similar reason, Option D is also incorrect because only the primitive `double` is compatible with this expression. Option C is correct and results in just a blank line being printed. Option E is correct since it is just the lambda version of the method reference in Option A. Finally, Option F is incorrect because of incompatible data types. The method reference is being used inside of a lambda expression, which would only be allowed if the functional interface returned another functional interface reference.
22. F. This class correctly creates and retrieves a Java class resource bundle. Since `count` is an `int`, it is autoboxed into an `Integer`. However, `rb.getString()` cannot be called for an `Integer` value. The code throws a `ClassCastException`, so Option F is the answer. If this was fixed, the answer would be Option C because the pre-increment is used and `getContents()` is only called once.

- 23.** C. The Java 8 date/time APIs count months starting with one rather than zero. This means `localDate` is created as October 5, 2017. This is not the day that daylight savings time ends. The loop increments the hour six times, making the final time 07:00. Therefore Option C is the answer.
- 24.** A. The code does not compile because `BasicFileAttributes` is used to read file information, not write it, making Option A the correct answer. If the code was changed to pass `BasicFileAttributeView.class` to the `Files.getFileAttributeView()` method, then the code would compile, and Option B would be the correct answer. Finally, remember that by default, symbolic links are not traversed by the `Files.walk()` method, avoiding a cycle. If `FileVisitOption.FOLLOW_LINKS` was passed, then the class would throw an exception because `Files.walk()` does keep track of the files it visits and throws a `FileSystemLoopException` in the presence of a cycle.
- 25.** B. The code compiles without issue, making Option E incorrect. Option A is incorrect because it disables all assertions, which is the default JVM behavior. Option B is the correct answer. It disables assertions everywhere but enables them within the `Watch` class, triggering an `AssertionError` at runtime within the `checkHour()` method. Option C is incorrect because it enables assertions everywhere but disables them within the `Watch` class. Option D is also incorrect because `-enableassert` is not a valid JVM flag. The only valid flags to enable assertions are `-ea` and `-enableassertions`.
- 26.** F. The code compiles without issue, making Option C incorrect. The `main()` method creates a thread pool of four threads. It then submits 10 tasks to it. At any given time, the `ExecutorService` has up to four threads active, which is the same number of threads required to reach the `CyclicBarrier` limit. Therefore, the barrier limit is reached twice, printing `Jump!` twice, making Option A incorrect. Unfortunately, the program does not terminate, so Option B is also incorrect. While eight of the tasks have been completed, two are left running. Since no more tasks will call the `await()` method, the `CyclicBarrier` limit is never reached, and the two remaining threads' tasks hang indefinitely. For this reason, Option F is the correct answer. Option E is incorrect because making the `IntStream` `parallel` would not change the result. Option D is incorrect because the result is the same no matter how many times the program is executed.
- 27.** C. The code does not compile because `Files.isSameFile()` requires `Path` instances, not `String` values. For this reason, Option C is the correct answer. If `Path` values had been used, then the code would compile and print `Same!`, and Option A would be the correct answer since the `isSameFile()` method does follow symbolic links.
- 28.** D, E. Since `JavaProgrammerCert` is a subclass of `Exam`, it cannot have a more specific visibility modifier. `Exam` uses `protected`, which is broader than package-private in `JavaProgrammerCert`. This rules out Options A, B, and C. The other three options all compile. However, Option F has a problem. Suppose your `JavaProgrammerCert` object has an `Exam` object with `pass` set to `true` for both the `oca` and `ocp` variables. The implementation in Option F doesn't look at either of those variables. It looks at the superclass's `pass` value. This isn't logically correct. Therefore, Options D and E are correct.
- 29.** B, D. First off, Option A is incorrect because a narrower exception can be thrown by an overridden method, just not a broader checked exception. Option B is correct and is the rule for return types of overridden methods. Option C is incorrect because overridden methods

must use the same input arguments. If the input arguments are different, the method is overloaded, not overridden. Option D is correct because modifiers that limit the access of the inherited method are not permitted. Option E is incorrect because `private` and `static` methods cannot be overridden. Remember that `private` methods defined in parent classes are not inherited, aka not visible in the child class, while `static` methods may be visible but cannot be overridden since they do not belong to an instance. Finally, Option F is also incorrect. The `@Override` annotation is optional and recommended but not required.

30. D. The most common approach is III, which works for any `SELECT` statement that has an `int` in the first column. If the `SELECT` statement has a function like `count(*)` or `sum(*)` in the first column, there will always be a row in the `ResultSet`, so II works as well. Therefore, Option D is the answer.
31. E. `DoubleBinaryOperator` takes two `double` values and returns a `double` value. `LongToIntFunction` takes one `long` value and returns an `int` value. `ToLongBiFunction` takes two generic arguments and returns a `long` value. `IntSupplier` does not take any values and returns an `int` value. `ObjLongConsumer` takes one generic and one `long` value and does not return a value. For these reasons, Option E is the correct answer.
32. F. While an `Instant` represents a specific moment in time using GMT, Java only allows adding or removing units of `DAYS` or smaller. This code throws an `UnsupportedTemporalTypeException` because of the attempt to add `YEARS`. Therefore, Option F is correct.
33. B. First we create two `Blankie` objects. One of them has the color `pink` and the other leaves it as the default value of `null`. When the stream intermediate operation runs, it calls the `isPink()` method twice, returning `true` and `false` respectively. Only the first one goes on to the terminal operation and is printed, making Option B correct.
34. F. This class is not immutable. Most obviously, an immutable class can't have a setter method. It also can't have a package-private instance variable. The getter method should be `final` so the class prevents a subclass from overriding the method. Since all three of these changes are needed to make this class immutable, Option F is the answer.
35. B, C, E. First of all, `BufferedOutputStream` does not exist in `java.io`, making Option A incorrect. Also, `OutputStream` is abstract, not concrete, so Option F can also be eliminated. The data being written is stored in memory as a `Student` object, so serializing with `ObjectOutputStream` is appropriate. Since a large set of records are involved, we should use `BufferedOutputStream`. Since the data is being written to a file, we would use `FileOutputStream`. For these reasons, Options B, C, and E are correct. Note that `FileWriter` is not possible since it cannot be chained with the other two `java.io` classes, making Option D incorrect.
36. C, D. Option A is incorrect because the `peek()` method returns the next value or `null` if there isn't one without changing the state of the queue. In this example, both `peek()` calls return `18`. Option B is incorrect because the `poll()` method removes and returns the next value, returning `null` if there isn't one. In this case, `18` and `null` are returned, respectively. Options C and D are correct because both the `pop()` and `remove()` methods throw a `NoSuchElementException` when the queue is empty. This means both return `18` for the first call and throw an exception for the second.

- 37.** F. Line 11 assigns a relative Path value of `./song/.../note` to `x`. The second line assigns an absolute Path value of `/dance/move.txt` to `y`. Line 13 does not modify the value of `x` because Path is immutable and `x` is not reassigned to the new value. On line 14, the `resolve()` method is called using `y` as the input argument. If the parameter passed to the `resolve()` method is absolute, then that value is returned, leading the first `println()` method call to output `/dance/move.txt`. On the other hand, the `relativize()` method on line 15 requires both Path values to be absolute, or both to be relative. Mixing the two leads to an `IllegalArgumentException` on line 15 at runtime and makes Option F the correct answer.
- 38.** B. The code does not compile, so Option E is incorrect. The first compilation error is in the `try-with-resources` declaration. There are two resources being declared, which is allowed, but they are separated by a comma (,) instead of a semicolon (;). The second compilation problem is that the order of exceptions in the two `catch` blocks are reversed. Since `Exception` will catch all `StungException` instances, the second `catch` block is unreachable. For these two reasons, Option B is the correct answer.
- 39.** C. The class compiles and runs without issue, so Options D and E are incorrect. The result of `findSlow()` is deterministic and always 1. The `findFirst()` method returns the first element in an ordered stream, whether it be serial or parallel. This makes it a costly operation for a parallel stream since the stream has to be accessed in a serial manner. On the other hand, the result of `findFast()` is unknown until runtime. The `findAny()` method may return the first element or any element in the stream, even on serial streams. Since both 1 1 and 3 1 are possible outputs of this program, the answer cannot be determined until runtime, and Option C is the correct answer.
- 40.** D. There is not a method called `getDefaultValue()` on the `Properties` class. Since the code does not compile, Option D is the answer. The `getProperty()` method on `Properties` is overloaded to allow passing a default value as the second parameter. If this code was changed to use `getProperty()`, the answer would be Option C.
- 41.** B, D. The `try-with-resources` statement requires resources that implement `AutoCloseable`. While `Closeable` extends `AutoCloseable`, it is certainly possible to have a class that implements `AutoCloseable` and works with `try-with-resources` but does not implement `Closeable`, making Option A incorrect. Option B is correct and a valid statement about how resources are closed in `try-with-resources` statements. Option C is incorrect because the exception in the `try` block is reported to the caller, while the exception in the `close()` method is suppressed. Option D is the other correct answer because neither `catch` nor `finally` are required when `try-with-resources` is used. Lastly, Option E is incorrect. While the `AutoCloseable` does define a `close()` method that throws a checked exception, classes that implement this method are free to drop the checked exception, per the rules of overriding methods.
- 42.** B. The `Roller` class uses a formal type parameter named `E` with a constraint. The key to this question is knowing that with generics, the `extends` keyword means any subclass or the class can be used as the type parameter. This means both `Wheel` and `CartWheel` are OK. The `wheel1` declaration is fine because the same type is used on both sides of the declaration. The `wheel2` declaration does not compile because generics require the exact same type when not using wildcards. The `wheel3` and `wheel4` declarations are both fine because

this time there is an upper bound to specify that the type can be a subclass. By contrast, the `super` keyword means it has to be that class or a superclass. The `wheel6` declaration is OK, but the `wheel5` one is a problem because it uses a subclass. Since `wheel2` and `wheel5` don't compile, the answer is Option B.

43. B. This class is not a singleton because it needs a private constructor. Having a setter method is fine. The state of a singleton's instance variables is allowed to change. The static initializer is fine as it runs at the same line as the declaration on line 2. Therefore, only the constructor addition is needed, and Option B is correct.
44. A. This code tries to update a cell in a `ResultSet`. However, it does not call `updateRow()` to actually apply the changes in the database. This means the `SELECT` query does not see the changes and outputs the original value of 0. Option A is correct.
45. A, B, D. Option A is correct because the `java.io` stream classes implement `Closeable` and can be used with `try-with-resources` statements, while `java.util.stream.Stream` does not implement `Closeable`. Option B is correct since the Reader/Writer classes are used for handling character data. There are primitive stream classes in `java.util.stream`, but none for handling character data, such as `CharStream`. Option C is incorrect because neither requires all data objects to implement `Serializable`. Option D is correct since `flush()` is found in `Writer` and `OutputStream` but not in any of the `java.util.stream` classes. Option E is incorrect since both types of streams contain a `skip()` method in some of their classes. Lastly, Option F is incorrect. There is no `sort` method found in any of the `java.io` classes. While there is a `sorted()` method in `java.util.stream.Stream`, the question is asking about what features are available in a `java.io` stream class and not in a `java.util.stream.Stream` class.
46. E. The code does contain compilation errors, so Option A is incorrect. The first is on line 8. The `readAllLines()` method returns a `List<String>`, not a `Stream`. While `parallelStream()` is allowed on a `Collection`, `parallel()` is not. Next, line 14 does not compile because of an invalid method call. The correct NIO.2 method call is `Files.isRegularFile()`, not `File.isRegularFile()`, since the legacy `File` class does not have such a method. Line 18 contains a similar error. `Path` is an interface, not a class, with the correct call being `Paths.get()`. Lastly, line 19 does not compile because the `read()` method throws `Exception`, which is not caught or handled by the `main()` method. For these four reasons, Option D is the correct answer.
47. D. The code compiles, making Option E incorrect. The key here is that the `AtomicInteger` variable is thread-safe regardless of the synchronization methods used to access it. Therefore, synchronizing on an instance object, as in `increment1()` or `increment3()`, or on the class object, as in `increment2()`, is unnecessary because the `AtomicInteger` class is already thread-safe. For this reason, Option D is the correct answer.
48. B, D. The `Files.find()` method requires a starting `Path` value, an `int` maximum depth, and a `BiPredicate<Path, BasicFileAttributes>` matcher instance. For these reasons, Options A, C, and E are incorrect. A `FileVisitOption` vararg is allowed but not required, making Option B correct. The other correct answer is Option D because the method does not take a `long` value.

- 49.** C, D. Option A is incorrect because `Comparable` is implemented on the class being compared. To be useful, such a class must have instance variables to compare, ruling out a lambda. By contrast, a `Comparator` is often implemented with a lambda. Option B is incorrect because `compare()` is found in a `Comparator`. Option C is correct because these methods have different parameters but the same return type. Option D is correct because a `Comparator` doesn't need to be implemented by the class being compared. Option E is incorrect because multiple comparators can use different orders for comparison, which do not need to match the definition of equality.
- 50.** C. The code compiles, so Option E is incorrect. The first boolean expression returns `false` because the two `Path` expressions have different values and are therefore not equivalent. On the other hand, the second boolean expression returns `true`. If we normalize `t1`, it becomes `/stars.exe`, which is equivalent to the `t2` variable in terms of `equals()`. The third boolean expression also returns `true`, even though the file does not exist. The `isSameFile()` method will avoid checking the file system if the two `Path` expressions are equivalent in terms of `equals()`, which from the second boolean expression we know that they are. That leaves the fourth boolean expression, which returns `true`. Passing an absolute `Path` to `resolve()` just returns it, so `t2` and `t3` are equivalent values. For these reasons, Option C is the correct answer. Note that if the `Path` values had not been equivalent in terms of `equals()` for either of the last two boolean expressions, then the file system would have been accessed, and since none of the files exist, an exception would have been thrown at runtime.
- 51.** B, E. The method does not call the `markSupported()` prior to calling `mark()` and `reset()`. This is considered a poor practice. The `input` variable could be a subclass of `Reader` that does not support this functionality. In this situation, the method would ignore the `mark()` call and throw an `IOException` on the `reset()` method, making Option A incorrect and Option B correct. On the other hand, if marking the stream is supported, then no exception would be thrown. First, line 24 skips two values, 1 and 2. On line 25, the `mark()` method is called with a value of 5, which is the number of characters that can be read and stored before the marked point is invalidated. Next, line 26 would skip another value but is undone by the `reset()` on line 27. The next value to be read would be the third value, 3. The `read(char[])` call would then read up to five values, since that is the size of the array. Since only four are left (4, 5, 6, 7) the method would return a value of 4, corresponding to the number of characters read from the stream. For these reasons, the output is 3-4, making Option E the correct answer. Options C and D can be eliminated because `read()` returns an `int` value, not a `char`.
- 52.** C. The `Function` interface uses `apply()`, while the `Consumer` interface uses `accept()`, making Option C the correct answer. For reference, `get()` is the name of the method used by `Supplier`, while `test()` is the name of the method used by `Predicate`.
- 53.** D. The `Teacher` class, including all five `assert` statements, compiles without issue, making Option F incorrect. The first three `assert` statements on lines 4, 5, and 6 evaluate to `true`, not triggering any exceptions, with `choices` updated to 11 after the first assertion is executed. Lines 4 and 7 demonstrate the very bad practice of modifying data in an assertion statement, which can trigger side effects. Regardless of whether an assertion error is thrown, turning on/off assertions potentially changes the value returned by `checkClasswork()`. At line 7, the

assertion statement `12==11` evaluates to `false`, triggering an `AssertionError` and making Option D the correct answer. The `main()` method catches and rethrows the `AssertionError`. Like writing assertion statements that include side effects, catching `Error` is also considered a bad practice. Note that line 8 also would trigger an `AssertionError`, but it is never reached due to the exception on line 7.

54. A, C, E. `BooleanSupplier`, `DoubleUnaryOperator`, and `ToLongBiFunction` are all valid functional interfaces in `java.util.function`, making Options A, C, and E correct. Remember that `BooleanSupplier` is the only primitive functional interface in the API that does not use `double`, `int`, or `long`. For this reason, Option B is incorrect since `char` is not a supported primitive. Option D is incorrect because the functional interfaces that use `Object` are abbreviated to `Obj`. The correct name for this functional interface is `ObjIntConsumer`. That leaves Option F, which is incorrect. There is no built-in `Predicate` interface that takes three values.
55. B, F. The `LackOfInformationException` class does not compile, making Option A incorrect. The compiler inserts the default no-argument constructor into `InformationException` since the class does not explicitly define any. Since `LackOfInformationException` extends `InformationException`, the only constructor available in the parent class is the no-argument call to `super()`. For this reason, the constructor defined at line `t1` does not compile because it calls a nonexistent parent constructor that takes a `String` value, and Option B is one of the correct answers. The other two constructors at lines `t2` and `t3` compile without issue, making Options C and D incorrect. Option E is also incorrect. The `getMessage()` method is inherited, so applying the `@Override` annotation is allowed by the compiler. Option F is the other correct answer. The `LackOfInformationException` is a checked exception because it inherits `Exception` but not `RuntimeException`.
56. C. First, a method reference uses two colons, so it should be `Ready::getNumber`. Second, you can't use generics with a primitive, so it should be `Supplier<Double>`. The rest of the code is correct, so Option C is correct.
57. E. The class compiles, so Options A, B, and C are incorrect. It also does not produce an exception at runtime, so Option F is incorrect. The question reduces to whether or not the `compute()` method properly implements the fork/join framework in a multi-threaded manner. The `compute()` method returns "1" in the base case. In the recursive case, it creates two `PassButter` tasks. In order to use multiple concurrent threads, the first task should be started asynchronously with `fork()`. While that is processing, the second task should be executed synchronously with `compute()` with the results combined using the `join()` method. That's not what happens in this `compute()` method though. The first task is forked and then joined before the second task has even started. The result is that the current thread waits for the first task to completely finish before starting and completing the second task synchronously. At runtime, this would result in single-threaded-like behavior. Since this is a poor implementation of the fork/join framework, Option E is the correct answer.
58. C. The `filter()` method passes two of the three elements of the stream through to the terminal operation. This is redundant since the terminal operation checks the same `Predicate`. There are two matches with the same value, so both `anyMatch()` and `allMatch()` return `true`, and Option C is correct.

- 59.** E. First, Java looks for the requested resource bundle, which is `AB_fr.class` and then `AB_fr.properties`. This rules out Options A, B, and C. Next, Java looks for the default locale's resource bundle, which is `AB_en.properties`. This rules out Option D. Java looks for the default resource bundle. First, Java checks for a Java class file resource bundle and then moves on to the property file. Therefore, Option F is incorrect, and Option E is the answer.
- 60.** D. This code attempts to use two terminal operations, `forEach()` and `count()`. Only one terminal operation is allowed, so the code does not compile, and Option D is correct. The author of this code probably intended to use `peek()` instead of `forEach()`. With this change, the answer would be Option A.
- 61.** F. This code sets the default locale to Japanese and then tries to get a resource bundle for `container` in the `pod` package. Since there is not a Japanese resource bundle available, it uses the default resource bundle `pod.container.properties`. Line `r1` successfully gets the value `generic` for the key name. Line `r2` throws a `MissingResourceException` because there is not a key type in the default resource bundle. The English resource bundle has this key, but it is not in the resource bundle hierarchy.
- 62.** D. Line `c1` correctly creates a stream containing two streams. Line `c2` does not compile since `x` is a stream, which does not have an `isEmpty()` method. Therefore, Option D is correct. If the `filter()` call was removed, `flatMap()` would correctly turn the stream into one with four `Integer` elements and `max()` would correctly find the largest one. The `Optional` returned would contain `33`, so Option B would be the answer in that case.
- 63.** A, C. A `Locale` uses lowercase letters for language codes and uppercase letters for country codes. It can consist of only a language, making Option A correct. If both a language and country code are present, the language code is first, making Option C correct.
- 64.** D, E. While this code does not close the `Statement` and `Connection`, it does compile, making Option A incorrect. Java defaults to auto-commit, which means the update happens right away, making Option C incorrect. Option B is incorrect because either `execute()` or `executeUpdate()` is allowed for UPDATE SQL. The difference is the return type. The `execute()` method returns a `boolean` while the `executeUpdate()` method returns an `int`. The code also runs without error, making Options D and E the answer. And remember to always close your resources in real code to avoid a resource leak.
- 65.** F. This is tricky. The `equals()` method in the `Object` class has a parameter of type `Object`. An overridden version is required to have the same type. The `equals()` method in `Sticker` is an overload rather than an override. Since there is an `@Override` annotation, the code does not compile.
- 66.** C. The code does not compile, so Options A and B are incorrect. The code does not compile because the `Sweater` class is an inner class defined on the instance, which means it is only accessible to be extended and used inside an instance method or by a `static` method that has access to an instance of `Wardrobe`. Since `dress()` is a `static` method, the declaration of local inner class `Jacket` does not compile on line `v1`, making Option C the correct answer. The rest of the code, including the abstract class `TShirt` and anonymous inner class defined inside `dress()`, compile without issue. If the `dress()` method was updated to remove the `static`

modifier, then the code would compile and print Starting...Insulation:20 at runtime, making Option A the correct answer.

67. E. The first statement is not true because `Function` takes two generic arguments and one input argument. If `BiFunction` was used instead of `Function` on line 7, then the code would compile correctly. The second statement is also not true because `IntSupplier` does not take any generic arguments. The third statement is not true as well, since `Armor` is an inner instance class. Without an instance of `Sword` in the `static main()` method, the call `new Armor()` on line 11 does not compile. For these reasons, all three statements are not true, making Option E the correct answer.
68. B, F. A deadlock and livelock both result in threads that cannot complete a task, but only in a livelock do the threads appear active, making Option A incorrect and Option B correct. Options C and D are incorrect because they do not apply to thread liveness. A race condition is an unexpected result when two threads, which should be run sequentially, are run at the same time, leading to an unexpected result, making Option E incorrect. Last but not least, starvation is caused by a single active thread that is perpetually denied access to a shared resource or lock, making Option F the other correct answer.
69. A, E. The first step is to convert both to GMT. We subtract the time zone offset to do this. In GMT, the Nairobi time is 14:00, which we get by subtracting 3 from 17. The Panama time is 15:00 because subtracting negative five from 10 gives us 15. Remember that subtracting a negative number is like adding a positive number. Since the Nairobi time is an hour before the Panama time, Options A and E are correct.
70. D. This code actually does compile. Line `c1` is fine because the method uses the `?` wildcard, which allows any collection. Line `c2` is a standard method declaration. Line `c3` looks odd, but it does work. The lambda takes one parameter and does nothing with it. Since there is no output, Option D is correct.
71. A, C. JDBC 3.0 drivers require a `Class.forName()` call. Since this is missing, Option A is correct, and Option B is incorrect. The `Connection` and `Statement` creation are correct, making Options E and F incorrect. Since the call to `stmt.close()` should be before the call to `conn.close()`, Option C is correct, and Option D is incorrect.
72. B. The code does not compile, so Options D and E are incorrect. The first compilation error is in the `Finder` interface declaration. Since all interfaces are implicitly `abstract`, they cannot be marked `final`. The second compilation error is the declaration of the `find()` method in the `Waldo` class. Since `find()` is inherited from the `Finder` interface, it is implicitly `public`. This makes the override of the method in `Finder` invalid because the lack of access modifier indicates package-level access. Since package-level access is more restrictive than the inherited method's access modifier, the overridden method does not compile in the `Waldo` class. For these two reasons, Option B is the correct answer.
73. F. When summing `int` primitives, the return type is also an `int`. Since a `long` is larger, you can assign the result to it, so line 7 is correct. All the primitive stream types use `long` as the return type for `count()`. Therefore, the code compiles, and Option E is incorrect. When actually running the code, line 8 throws an `IllegalStateException` because the stream has already been used. Both `sum()` and `count()` are terminal operations and only one terminal operation is allowed on the same stream. Therefore, Option F is the answer.

- 74.** D. Line 34 does not compile because of an assignment and value mismatch. The `r1` variable is a `Runnable` expression. While there is an `ExecutorService.submit()` that takes a `Runnable` expression, it returns `Future<?>` since the return type is `void`. This type is incompatible with the `Future<Stream>` assignment without an explicit cast, leading to a compiler error. Next, line 39 does not compile. The `parallelStream()` method is found in the `Collection` interface, not the `Stream` interface. Due to these two compilation errors, Option D is the correct answer.
- 75.** F. Internationalization means the program is designed so it can be adapted for multiple languages. Localization means the program actually supports multiple locales. Since a localized application must first be internationalized, Option F is the answer. Extracted is not a word commonly used with respect to handling multiple languages.
- 76.** C. The class compiles, making Option E incorrect. If `/woods/forest` exists, then the first branch of the if-then statement executes, printing `true` at runtime. On the other hand, if `/woods/forest` does not exist, the program will print `true` if `/woods` exists and `false` if `/woods` does not exist. Unlike `makedirs()`, which if used would always return `true` in this case, `mkdir()` will return `false` if part of the parent path is missing. For this reason, Option C is correct, and Options A and B are incorrect. Finally, Option D is incorrect. This code is not expected to throw an exception at runtime. If the path could not be created, the `mkdir()` method just returns `false`.
- 77.** D. Line 15 calls the method on line 9 since it is a `Watch` object. Line 16 is a `SmartWatch` object. However, the `getName()` method is not overridden in `SmartWatch` since the method signature is different. Therefore, the method on line 9 gets called again. That method calls `getType()`. Since this is a `private` method, it is not overridden and `watch` is printed twice. Option D is correct.
- 78.** B. This code runs the loop six times, adding an hour to `z` each time. However, the first time is the repeated hour from daylight savings time. The time zone offset changes, but not the hour. This means the hour only increments five times. Adding that to 01:00 gives us 06:00 and makes Option B correct.
- 79.** C. Both `schedule` method calls do not compile because these methods are only available in the `ScheduledExecutorService` interface, not the `ExecutorService` interface. Even if the correct reference type for `service` was used, along with a compatible `Executors` factory method, the `scheduleWithFixedDelay()` call would still not compile because it only contains a single numeric value. This method requires two `long` values, one for the initial delay and one for the period. The `execute()` method call compiles without issue because this method is available in the `ExecutorService` interface. For these reasons, only the third statement is true, making Option C the correct answer.
- 80.** A, D, E. Any class that inherits from `RuntimeException` or `Error` is unchecked, while any class that does not is checked and must be declared or handled. `AssertionError` inherits from `Error`, while `IllegalArgumentException` and `MissingResourceException` inherit from `RuntimeException`. The remaining classes—`NotSerializableException`, `SQLException`, and `ParseException`—each inherit `Exception` but not `RuntimeException`, making Options A, D, and E the correct answers.

- 81.** C, D. To be a valid functional interface, an interface must declare exactly one abstract method. Option A is incorrect, because `CanClimb` does not contain any abstract methods. Next, all interface methods not marked `default` or `static` are assumed to be abstract, and abstract methods cannot have a body. For this reason, `CanDance` does not compile, making Option B incorrect. Options C and D are correct answers because each contains exactly one abstract method. Option E is incorrect because it contains two abstract methods, since `test()` is assumed to be abstract.
- 82.** A. A JDBC URL has three components separated by colons. None of these options uses the correct colon delimiter, making Option A the correct answer. If all the semicolons were changed to colons, Option D would be correct. I and V would still be incorrect because they don't begin with the JDBC protocol and magic driver name as the first two components.
- 83.** F. The collector tries to use the number of characters in each stream element as the key in a map. This works fine for the first two elements, `speak` and `bark`, because they are of length 5 and 4, respectively. When it gets to `meow`, we have a problem because the length 4 is already used. Java requires a merge function be passed to `toMap()` as a third parameter if there are duplicate keys so it knows what to do. Since this is not supplied, the code throws an `IllegalStateException` due to the duplicate key, and Option F is correct.
- 84.** D. The application does not compile, so Options A, B, and C are incorrect. The one and only compilation issue is that the `ElectricBass` class implements two interfaces that declare `default` methods with the same `getVolume()` signature. A class can inherit two `default` methods with the same signature but only if the class overrides the methods with its own version. Since `ElectricBass` does not override the method, the `ElectricBass` class does not compile. Since this is the only compilation issue, Option D is the correct answer. If the `ElectricBass` class did correctly override the `getVolume()` method, the rest of the code would compile without issue. In this case, there would be nothing printed at runtime. The `main()` method just declares a local inner class but does not create an instance of it, nor does it call any method on it, making Option C the correct answer.
- 85.** A, B, C. `Files.find()` and `Files.list()` return a `Stream<Path>`, while `Files.lines()` returns a `Stream<String>`. For these reasons, Options A, B, and C are the correct answers. The NIO.2 `Files` class does not contain a `listFiles()` method, making Option D incorrect. There is a method named `listFiles()` in the `java.io.File` class, but it returns a `File` array. Option E is also incorrect because the `Files.readAllLines()` method returns a `List<String>`. Lastly, Option F is incorrect because `Files.walkFileTree()` uses a `FileVisitor` and returns a `Path`. If you were not familiar with the `walkFileTree()` method, then you could have ruled it out by knowing the signatures for Options A, B, and C.



# Index

---

## A

absolute() method, 367–368  
abstract classes, 206  
    default methods, 110  
    extending, 104, 209  
    inheritance, 214  
    instantiation, 103–104  
    methods, 103  
    Writer, 302  
abstract methods, 174  
    enum, 207  
    inclusion restrictions, 109  
    interfaces, 160, 206, 211, 215–216, 425  
    protected modifier, 106  
    public modifier, 98  
accept() method, 411  
access modifiers  
    examples, 172–174, 400  
    interface methods, 112  
    interface variables, 106  
    private, 83, 87, 91, 98, 192  
    protected, 80, 95, 106  
    public, 5, 98, 168, 192  
accessibility, object, 21–22  
accumulators in parallel reduction, 342  
add() method, 137–138  
adding  
    ArrayList, 137–138  
    database rows, 364, 366, 368–369  
    dates, 402  
    days, 292, 296–297, 299  
    hours, 300  
    months, 297, 299  
    primitives, 421  
AddingException class, 388  
addition operator (+)  
    evaluation order, 39  
    precedence, 48  
allMatch() method, 266, 414

and operator (`&&`)  
    boolean expressions, 35–36  
    conditional and nonconditional, 42–44  
    truth tables, 37–38  
    zero-length arrays, 47  
anonymous inner classes, 206, 212–213  
anyMatch() method, 256, 266, 414  
applications, compiling, 7  
apply() method, 245–246  
    BiFunction, 252  
    Function, 411  
args array, 181  
arguments, 10  
    arrays, 61–62  
    classes, 57–58  
    command-line, 181  
    constructors, 83, 91, 94, 113  
ArithmetricException class, 125–126, 172, 281  
ArrayDeque class, 224  
ArrayIndexOutOfBoundsException class, 53, 56, 58, 124, 170, 177  
ArrayList class, 136  
    elements, 137, 139  
    empty, 148–149  
    iteration, 65  
    java.util package, 140  
    null values, 232  
    order, 137  
    remove(), 146  
    size, 141, 147  
arrays  
    arguments, 61–62  
    copying, 339  
    data types, 60  
    declarations, 50, 52, 54, 58, 62  
    dimensions, 55  
        multi-dimensional, 53  
        two-dimensional, 51, 59  
element retrieval, 58  
empty, 60–61  
expanding, 52

indexes, 51–52, 54–59, 61, 178  
 iteration, 65  
 length, 51, 54–55, 59, 61  
 Lists from, 167  
 parameters vs. varargs, 50–51  
 vs. primitives, 50  
 printing, 341  
 reading and writing, 311–312  
 searching, 51–52, 62, 176, 181, 354–355  
 size, 165  
 sorting, 51–52  
 String elements, 51  
`toString()`, 55  
 zero based, 50  
**ArrayStoreException** class, 56  
**assert** keyword, 273, 279  
**AssertionError** class, 275, 286, 397–398  
**assertions**  
 disabling, 280, 286  
 enabling, 280  
 examples, 284, 411–412  
 guaranteed, 275  
 purpose, 282  
**assignments**  
 vs. comparison, 46  
 mismatches, 421–422  
 variables, 25, 154  
**asterisks (\*)**  
 comments, 166  
 precedence, 48, 152  
**asynchronous tasks**, 339  
**AtomicInteger** class, 409  
**attributes**  
 classes, 321  
 objects and references, 113  
**AutoCloseable** interface, 277, 281, 287–288  
**automatic committing**, 363, 366  
**average()** method  
`LongStream`, 258  
`OptionalDouble`, 256–257

---

## B

balanced parentheses, 39  
**BasicFileAttributes** class, 326, 328, 397  
**BasicFileAttributeView** class, 328, 331–332  
**beforeFirst()** method, 369  
**between()** method, 298, 393

**BiFunction** interface, 241  
`apply()`, 252  
 arguments, 248  
 inputs, 246  
**BinaryOperator** interface, 246  
**binarySearch()** method, 51–52, 60, 181  
**BiPredicate** interface, 241–242  
 blocks for if-then statements, 36  
 booleans and boolean expressions  
 if-then statements, 37–38, 41–42  
 mapping to, 267–268  
 truth tables, 37–38  
**BooleanSupplier** interface, 249, 252, 412  
**brackets ({} )**  
 arrays, 50, 54  
 catch blocks, 121  
 finally blocks, 120  
 loops, 68, 168, 182–183  
 return statement, 143  
**break statements**  
 case statements, 37, 41  
 loops, 64, 66, 69, 74, 159  
 switch statements, 35  
**BufferedInputStream** class, 313, 315  
**BufferedOutputStream** class, 305, 403  
**BufferedWriter** class, 302–303, 308, 311  
**ByteArrayInputStream** class, 312–313  
**bytecode files**  
 compiling, 168  
 extension, 4  
 running, 8

---

## C

`call()` method, 342  
**Callable** class, 342  
**CallableStatement** interface, 359  
**calling**  
 constructors, 24–26  
 methods, 85–86, 92–93  
**case statements**  
 break statements, 41  
 switch statements, 37, 47  
 value, 171–172, 206  
 variable types, 38–39, 44–45  
**casting**  
 classes, 107–109  
 objects, 176

catch blocks  
brackets, 121  
checked exceptions, 388  
class handling, 127  
examples, 179–180  
exceptions thrown by, 126  
multi-catch statements, 123, 278, 282, 284–285  
order, 116, 118, 125, 175, 272  
required, 116, 120, 125, 158–159, 272  
try-with-resources, 278, 406  
variables, 122, 130–131, 133, 274  
chained methods, 137, 175  
char data type for wrapper class, 25  
Character objects, 260  
charAt() method, 147, 237  
checked exceptions  
catch blocks, 388  
class examples, 425  
ClassCastException, 117  
compatible, 118–119  
examples, 412  
handling, 122–124  
java.text.ParseException, 273  
main() method, 157–158, 281–282  
method signatures, 124  
overridden methods, 121, 128, 213–214, 279–280  
child classes in no-argument constructors, 113  
ChronoUnit enum  
  HOURS constant, 295  
  MONTHS constant, 393  
.class files, 4, 10, 168  
ClassCastException class, 117, 119, 122–124, 127–131, 227–228  
classes  
  abstract. *See* abstract classes  
  accessible, 13  
  arguments, 57–58  
  attribute, 321  
  casting, 107–109  
  checked exceptions, 425  
  compiling, 6  
  concrete, 109  
  declarations, 12, 23  
  defining, 161  
  element order, 20  
  encapsulation, 83, 85, 87  
examples, 14–16  
extending, 111  
immutable. *See* immutable classes and types  
inheritance, 12, 161, 426  
inner. *See* inner classes  
instantiation, 390–391  
interfaces, 102, 104  
java.util package, 140  
methods, 98  
names, 18, 23  
no-argument constructors, 113  
packages, 89, 161  
references to, 106  
singleton pattern, 196  
subclasses, 199–200, 207  
this keyword, 99  
thread-safe. *See* thread-safe classes  
variable definitions, 7  
variable scope, 9  
wrapper. *See* wrapper classes  
close() method  
  AutoCloseable, 281, 287–288  
  BufferedWriter, 311  
  Closeable, 277, 287  
  ResultSet, 362  
  Statement, 362  
Closeable interface, 277, 287  
closing resources, 420  
collect() method, 352  
Collection interface  
  forEach(), 234  
  removeIf(), 233  
  size(), 236–237  
collections, TreeMap, 224  
Collector argument for parallel reduction, 352  
Collectors interface  
  groupingBy(), 267  
  inputs, 259  
  toMap(), 258  
colons (:)  
  JDBC URL formats, 425  
  method references, 231–232, 413  
command-line arguments, 181  
commas (,) in for loops, 72  
comments, 5, 12, 166  
commit() method, 364  
committing, automatic, 363, 366

- Comparable interface
  - compareTo(), 229–230
  - implementing, 227–228
  - sorting, 234–235
  - TreeMap and TreeSet, 225
- Comparator interface
  - equal objects, 237
  - implementing, 225, 410
  - reverseOrder(), 261–262
  - sort order, 230
  - Stream, 262
- compare() method, 410
- compareTo() method, 229–230, 410
- comparisons
  - vs. assignment, 46
  - String, 34, 41–43, 154–155
  - times, 290, 298, 419
  - variables, 154
- compatibility of default interface methods, 214
- compiling
  - applications, 7
  - classes, 6
  - errors, 8–9, 14
  - javac command, 7, 10, 168
- compound assignment operators, 354
- compute() method, 342
- concat() method, 140
- concrete classes, 109
- concrete methods, 103
- concurrency modes, 360
- concurrent collections, 344
- ConcurrentLinkedDeque class, 352
- ConcurrentSkipListMap collection, 344
- conditional conjunction operator (`&&`)
  - boolean expressions, 35–36
  - conditional and nonconditional, 42–44
  - truth tables, 37–38
  - zero-length arrays, 47
- conditional disjunction (OR) operator (`||`)
  - description, 46
  - truth tables, 47–48
  - Venn diagram, 44
- conditions
  - do-while loops, 65, 70, 74
  - for loops, 71, 171
  - while loops, 65–66, 69–70, 156, 160–161
- Connection interface, 358, 361, 363
- connections, databases, 358
- Console class
  - constructors, 305
  - format() and print(), 308
  - console() method, 305, 310
- constructors
  - calling, 24–26
  - Console, 305
  - description, 93
  - enum, 395–396
  - examples, 89–90, 162–163
  - exceptions, 273
  - invalid, 308
  - lambda expressions, 228, 230
  - missing, 178–179
  - no-argument, 83, 91, 94, 113
  - singleton pattern, 407
  - super(), 80, 86
  - this(), 81
  - valid, 30
  - wrapper classes, 31
- Consumer interface
  - accept(), 411
  - forEach(), 245, 250
  - method references, 248
  - return value, 251
- continue keyword, 64, 73
- conversions
  - to GMT, 291
  - to parallel streams, 347
- copying
  - arrays, 339
  - files, 311–312, 324, 334
  - keys, 376
- CopyOnWriteArrayList class, 339, 352
- core database classes, 362–363
- count() method
  - primitive streams, 421
  - terminal operations, 415
- count(\*) function, 364–365
- country codes
  - getBundle(), 378–379
  - Locale, 374, 377
- covariant return types, 100, 104, 113–114, 130, 220–221, 400–401
- createDirectories() method, 330
- createDirectory() method, 330
- currency in locales, 372

current working directory  
  files, 304  
  paths, 330  
cursor position in ResultSet, 365

---

**D**

data corruption from race conditions, 346  
data types  
  arrays, 60  
  broadening, 169  
  decimals, 22–23  
  declarations, 20  
  extends keyword, 406–407  
  functional interfaces, 252  
  passing to methods, 85, 95  
  ternary operations, 41  
  up-converting, 94  
databases  
  adding rows, 364, 366, 368–369  
  closing resources, 369  
  connections, 358  
  core classes, 362–363  
  driver implementation, 363  
  empty tables, 360  
  examples, 416–417  
  indexes, 361  
  ResultSets. *See* ResultSet interface  
  SELECT statements, 401  
  URL formats, 358–359  
dates  
  adding, 402  
  changing, 328  
  classes, 143–144  
  comparisons, 419  
  formatting, 145, 297–300, 386  
  immutable, 149, 299, 389  
  locales, 372  
  toString(), 294  
DateTimeFormatter class  
  format(), 291  
  hours and minutes, 293  
  ofLocalizedDate(), 292–293  
daylight savings time, 290–293, 295–296,  
  300, 397, 423–424  
days  
  adding, 292, 296–297, 299  
  subtracting, 295

deadlocks  
  description, 339, 419  
  examples, 351  
decimals in data types, 22–23  
declarations  
  arrays, 50–52, 54, 58, 62  
  classes, 12, 23  
  data types, 20  
  interfaces, 420–421  
  lambda expressions, 387  
  main(), 169–170, 175  
  String, 180–181  
  try-with-resources, 404–405  
  variables, 18, 91, 195, 200, 204  
  wildcards, 233  
default instance variable values, 18  
default methods, 110, 114, 160, 214  
default resource bundles, 415  
default statements  
  interfaces, 203  
  switch statements, 35, 47, 163, 183  
deferred execution, 138, 228  
defining classes, 161  
delete() method  
  directories, 306–307, 310  
  Files, 325, 328  
DELETE SQL statements, 360  
deleteIfExists() method, 331  
deleting  
  directories, 310, 325, 328  
  files, 331  
depth-first traversal, 321–322  
design pattern examples, 391–392  
diamond operator ( $<>$ ), 142, 157  
dimensions, array, 55  
  multi-dimensional, 53  
  two-dimensional, 51, 59  
directories  
  creating, 307, 313, 330, 422–423  
  deleting, 310, 325, 328  
  file paths, 304  
  java.io.File, 303  
DirectoryNotEmptyException class,  
  323  
disabling assertions, 280, 286  
DividingException class, 388  
division, remainder, 40  
division operator (/) precedence, 44, 48

do-while loops, 64  
 break statements, 159  
 conditions, 65, 70, 74  
 examples, 67, 70–71, 77, 155  
**DosFileAttributes** class, 326  
**dots (.)**  
 instance variables, 28–29  
 local variables, 23  
 in packages, 15  
 varargs parameter, 50, 56  
**double type**  
 decimals, 22–23  
 suffixes, 20  
 wrapper class, 21  
**DoubleBinaryOperator** interface, 401–402  
**DoubleConsumer** interface  
 parameters, 244  
 return value, 240, 245  
**DoubleFunction** interface, 243  
**DoublePredicate** interface, 251  
**DoubleStream** interface, 256  
**DoubleSupplier** interface, 241  
**DoubleToIntFunction** interface, 249–250  
**DoubleToLongFunction** interface, 251  
**DoubleUnaryOperator** interface, 241, 412  
**Driver** interface, 358–359, 393  
**DriverManager** class, 358–359, 365  
**drivers**  
 database implementation, 363  
**ResultSet**, 360  
**duplicates** in **TreeSet**, 226  
**Duration** class, 292

---

## E

effectively final variables, 209  
**else** statements, 34–35  
 empty arrays, 60–61  
 empty database tables, 360  
 empty strings, 232  
 enabling assertions, 280  
**encapsulation**  
 classes, 83, 85, 87  
 description, 7, 200  
 examples, 188, 195–196  
 object orientation, 180  
 variables, 192

entry point methods, 4, 12, 16  
**enum** and **enumerated types**  
 constructors, 395–396  
 extending, 209–210  
 indexes, 214  
 methods, 207  
 names, 389  
 semicolons, 217–218, 395–396  
 static methods, 216  
 switch statements, 208  
 values, 210  
**EOFExceptions**, 114  
**equal signs (=)**  
 assignment operator vs. comparison, 46  
 comparisons, 154–155  
 precedence, 152  
**String**, 41  
**equality**  
 objects, 195, 202, 237  
**String**, 41  
**equals()** method, 141, 410  
 examples, 189–190  
 overriding, 417  
**String**, 41  
**Error** class, 116–117, 122, 133, 153, 284  
**evaluation order of operators**, 39, 43,  
 45–46  
**Exception** class, 116, 118  
**exceptions**  
 checked. *See checked exceptions*  
 constructors, 273  
 description, 121  
 examples, 276–277  
 purpose, 128  
 single pipe character, 272–273  
 unchecked. *See unchecked exceptions*  
 unhandled, 283, 286, 340  
**exclamation points (!)**  
 applying, 163  
 boolean values, 47  
 description, 46  
**execute()** method  
**ExecutorService**, 338  
 SQL statements, 360, 362, 416–417  
**executeQuery()** method, 360  
**executeUpdate()** method, 360  
 execution deferred, 138, 228  
**Executors** class, 340

ExecutorService class  
  methods, 338–339, 353  
  submit(), 349  
  threads, 399  
expanding arrays, 52  
extending  
  classes, 111  
    abstract, 104, 209  
    inner, 417–418  
  data types, 406–407  
  enumerated types, 209–210  
  interfaces, 102, 106, 111

---

## F

File class, 309  
  directories, 303  
  length(), 322  
  renameTo(), 302  
file system objects, 313  
FileInputStream class, 313  
FileNotFoundException class, 101, 111, 120, 133, 174  
FileOutputStream class, 302–304, 403  
files  
  copying, 311–312, 324, 334  
  deleting, 331  
  moving, 323  
  paths. *See* Path interface and paths  
Files class  
  delete(), 325, 328  
  deleteIfExists(), 331  
  find(), 321, 410  
  getFileAttributeView(), 331–332  
  isHidden(), 320  
  isSameFile(), 321, 323, 334, 400  
  lines(), 324, 333  
  list(), 327, 334–335  
  readAllLines(), 324, 333, 408–409  
  Stream methods, 426  
  walk(), 321–322  
FileWriter class, 308  
filter() method, 264, 414  
  return value, 235–236  
  streams, 229  
filters in stream pipelines, 236–237  
final modifiers  
  classes, 206

effectively, 209  
examples, 218–219  
interfaces, 112, 214  
lambda expressions, 252  
variables, 86, 88, 93  
Finalize class, 329–330  
finalize method, 21, 28  
finally blocks  
  brackets, 120  
  description, 177  
  examples, 179–180  
  exceptions thrown by, 126, 134  
  order, 116, 125, 175, 272  
  required, 116, 120, 125, 158–159, 272  
  try-with-resources, 406  
find() method, 321, 332–333, 410, 420–421, 426  
findAny() method, 260, 341, 405  
findFirst() method, 265  
flatMap() method, 261, 264, 266–267, 416  
flatMapToInt() method, 263  
flush() method, 311, 313–314, 408  
for loops, 64  
  brackets, 168  
  commas, 72  
  conditions, 71, 171  
  elements, 67–68, 70  
  examples, 71–73, 75–77, 177–178  
  execution flow, 76  
  indexes, 65  
for-each loops, 64  
  array iteration, 65  
  description, 181  
  examples, 68  
forEach() method  
  Collection, 234  
  Consumer, 245, 250  
  List and Stream, 224  
  requirements, 242–243  
  terminal operations, 415  
forEachOrdered() method, 353  
fork/join framework, 342  
  examples, 413–414  
  parallel processing, 352–353  
formal type parameters, 228–229, 231  
format() method  
  Console, 308  
  DateTimeFormatter, 291

formatted dates, 145, 147, 291, 297–300, 386  
 forName() method, 358–360, 365, 420  
 Function interface  
   apply(), 411  
   arguments, 250, 418–419  
 functional interfaces  
   description, 247  
   DoubleSupplier, 241  
   examples, 425  
   primitive, 252

getYear() method, 294  
 getYears() method, 294  
 GMT  
   converting to, 291  
   determining, 296  
 greater than signs (>), 45  
   diamond operator, 142  
   lambda expressions, 206–207  
 grouping() method, 263  
 groupingBy() method, 263, 267  
 groupingbyConcurrent() method, 347–348

---

## G

garbage collection  
   elements, 182  
   eligible objects, 29–30, 157, 176–177  
   examples, 30  
   forcing, 25  
   String, 26–27  
 generic types, missing, 231  
 geographical regions, 372  
 get() method  
   ArrayList, 138  
   Optional, 264  
   Paths, 331  
   Properties, 382–383  
   Supplier, 246  
 getAndDecrement() method, 350  
 getAsDouble() method, 257–258  
 getAverage() method, 263  
 getBundle() method, 378–379  
 getColor() method, 235–236  
 getConnection() method, 358–359, 365  
 getContents() method, 373–374  
 getCountAsLong() method, 268  
 getDefault() method, 372  
 getFileAttributeView() method, 331–332  
 getFileName() method, 323  
 getName() method, 322, 330  
 getObject() method, 379–380  
 getParent() method, 309  
 getProperty() method  
   Properties, 382–383, 406  
   System, 307  
 getRoot() method, 320, 329  
 getSeconds() method, 290  
 getString() method, 396–397

---

## H

hanging code, 340–341  
 hasNext() method, 226  
 hiding variables and methods, 108  
 hours  
   adding, 300  
   DateTimeFormatter, 293  
 HOURS constant, 295

---

## I

if-then statements  
   blocks, 36  
   boolean expressions, 37–38, 41–42  
   else, 34–35  
   evaluating, 36–37  
 if-then-else statements  
   examples, 173  
   ternary expressions for, 35  
 ifPresent() method, 260–261  
 IllegalArgumentException class, 129–131, 274, 278  
 IllegalStateException class, 230–231, 277  
 immutability, 190  
 immutable classes and types  
   dates and times, 299, 389–390  
   description, 192  
   examples, 192–193, 200–201, 403  
   instance variables, 202  
   objects, 153  
   String, 142–143, 160  
 import statements and importing  
   class accessibility, 13

packages, 4–5, 13  
  java.lang, 122  
  names in, 153  
redundant, 10  
required, 14  
static members, 95–96, 166  
unused, 6  
wildcards, 9

incrementAndGet() method, 350

indexes  
  ArrayList, 148–149  
  arrays, 51–52, 54–59, 61, 178  
  databases, 361  
  enumerated types, 214  
  for loops, 65  
  months, 144, 397  
  negative, 56–57  
  StringBuilder, 152

IndexOutOfBoundsException class, 148–149

infinite loops, 66, 68, 74

infinite streams, 262, 264

inheritance  
  abstract classes, 214  
  benefits, 12  
  classes, 426  
  description, 99, 164  
  interfaces, 161  
  method visibility, 183  
  multiple, 100, 110, 213

initializers  
  instances, 20–21  
  static, 201–203

initializing variables, 156  
  instance, 24  
  local, 23

inner classes, 195  
  anonymous, 206, 212–213  
  defining, 219  
  extending, 417–418  
  instantiation, 215  
  local, 208, 211  
  variables, 214–215, 220

inputs  
  BiFunction, 246  
  BinaryOperator, 246  
  Collectors, 259  
  IntUnaryOperator, 246  
  Supplier, 240

UnaryOperator, 244  
user, 310–311, 314

InputStream class  
  description, 393  
  read(), 302

insert() method, 147

instance methods  
  examples, 193–194, 197  
  primitives, 18–19

instance variables  
  accessing, 163  
  default values, 18, 183  
  dot notation, 28–29  
  immutable classes, 202  
  initializing, 24  
  object composition, 203  
  referencing, 27  
  static, 90  
  String, 25  
  virtual invocation, 192

instanceof operator, 191, 194

instances  
  initializers, 20–21  
  thread-safe, 338

Instant class, 296–297, 300

instantiation  
  classes, 390–391  
  abstract, 103–104  
  inner, 215  
  lazy, 200  
  objects, 19  
  resource bundles, 391

int data type  
  case statements, 38–39  
  wrapper class, 19, 21, 25

Integer class, 21

interfaces, 247  
  classes, 102, 104  
  declarations, 420–421  
  default statements, 203  
  extending, 102, 106, 111  
  final, 214  
  inheritance, 100, 161, 213  
  methods, 103, 160, 216  
    abstract, 206, 211, 215–216  
    default, 110, 114, 214  
    modifiers, 112  
    restrictions, 110  
    static, 102

program development, 99  
 valid, 218  
 variable modifiers, 106  
 intermediate operations in stream pipelines, 232, 260–261  
 internationalization, 372, 422  
**IntFunction** interface  
 lambda expressions, 252  
 parameters, 244  
 return types, 245  
**IntStream** interface, 259  
**IntSummaryStatistics** interface, 259, 268  
**IntSupplier** interface, 401–402  
**IntUnaryOperator** interface, 242  
 inputs, 246  
 lambda expressions, 393  
**invokeAll()** method, 343–344  
**invokeAny()** method, 343–344  
**IOException** class, 101, 111, 118  
*EOFException*, 114  
 examples, 410–411  
**FileNotFoundException**, 120, 133, 174  
 unchecked exceptions, 172  
**isEmpty()** method, 262, 416  
**isHidden()** method, 320  
**isPresent()** method, 259–261  
**isRegularFile()** method, 408–409  
**isSameFile()** method, 321, 323, 334, 400, 410  
**iteration**  
*ArrayLists*, 65  
 arrays, 65  
 lists, 228  
**Iterator** interface, 226

---

## J

Java, description, 164–165  
**.java** files  
 compiling, 10, 168  
 public classes, 5  
 resource bundles, 372, 375  
 source files, 13  
**java.io.File** class  
 directories, 303  
**length()**, 322  
**renameTo()**, 302  
**java.io.NotSerializableException** class, 425

**java.io** package  
 description, 408  
 stream classes, 308–309, 312  
**java.io.Serializable** class, 302  
**java.lang.Error** class, 122  
**java.lang** package, 5, 122  
**java.lang.RuntimeException** class, 288  
**java.sql** package, 358, 393  
**java.sql.SQLException** class, 425  
**java.text.ParseException** class, 273, 425  
**java.time** package, 290  
**java.util.MissingResourceException** class, 279  
**java.util** package, 140  
**java.util.function** package, 241, 248, 251, 412  
**java.util.stream.Stream** class, 408  
**Java** virtual machine (JVM)  
 entry point methods, 16  
 properties, 9  
**JavaBean** methods, 81, 89  
**javac** command  
 compiling, 7, 10, 168  
 packages, 15  
**jdbc:** characters for database URLs, 358  
**JDBC** URL formats, 425

---

## K

**keys**  
 copying, 376  
**ListResourceBundle**, 379  
 maps, 425–426  
**Properties**, 373  
 resource bundles, 384

---

## L

labels for loops, 73, 76–77  
**lambda** expressions  
 benefits, 138  
 constructor references, 228, 230  
 data types, 396  
 declarations, 387  
 description, 247  
**DoubleToIntFunction** compatibility, 249–250

DoubleToLongFunction compatibility, 251  
examples, 158, 240  
final variables, 252  
invalid, 247  
method equivalent, 217  
method references, 231–232  
parallel reduction, 342  
parameters, 139–140  
passing, 243, 252, 393  
return value, 143, 244  
syntax, 165–166  
ToDoubleBiFunction, 241  
types, 142–143  
UnaryOperator, 248  
valid, 206–207, 212, 386, 388  
wrapper classes, 249  
languages, 373–374, 377, 379  
lazy instantiation, 200  
length() method, 141, 147, 152–153, 322  
length of arrays, 51, 54–55, 59, 61  
less than operator (<)  
    comparisons, 45  
    diamond operator, 142  
lines() method, 324, 333, 426  
LinkedBlockingDeque class, 341  
List class and lists  
    from arrays, 167  
    forEach(), 224  
    iterating, 228  
    streams, 257  
    synchronized, 353  
list() method, 327, 334–335, 426  
ListResourceBundle class  
    getContents(), 373  
    key/value pairs, 379  
literals, 19, 23, 27  
livelocks, 339  
local inner classes  
    local variable access, 208  
    static methods, 211  
local variables  
    dot notation, 23  
    effectively final, 209  
    initializing, 23  
    local inner classes, 208  
    scope, 5, 8, 159–160  
LocalDate class  
    elements, 143  
    examples, 175  
    format(), 291  
    getYear(), 294  
    limitations, 182  
    months, 290, 294  
    package, 140  
LocalDateTime class  
    elements, 143  
    months, 144  
    of(), 296  
    resource bundles, 375  
Locale class and locales  
    country codes, 374, 377  
    currency and dates, 372  
    default, 373, 381–382, 384  
    examples, 372  
    formats, 374, 416  
    getDefault(), 372  
    internationalization, 422  
    languages, 373–374, 377, 379  
    object creation, 376  
    setDefault(), 373  
    localization, 372  
LocalTime class  
    elements, 143, 294  
    of(), 293  
    package, 290  
    units, 147  
logical complement operator (!)  
    applying, 163  
    boolean values, 47  
    description, 46  
long data type  
    case statements, 38–39  
    LongSupplier, 240  
    valid numbers, 155  
    wrapper class, 21  
LongStream interface, 258  
LongSupplier interface  
    long values, 240  
    parameters, 244  
    return types, 245  
LongToIntFunction interface, 401–402  
LongUnaryOperator interface, 251  
loops  
    braces, 68  
    break keyword, 64, 66, 69, 74  
    continue keyword, 64, 73  
    do-while, 64  
    for, 64

for-each, 64  
 infinite, 66, 68, 74  
 labels, 73, 76–77  
 while, 64

## M

main() method  
 args array, 167–168  
 checked exceptions, 157–158, 281–282  
 declarations, 169–170, 175  
 Map interface  
   Properties, 372, 375  
   Stream conversion, 258, 263  
 map() method, 257, 264  
 maps  
   to booleans, 267–268  
   keys, 425–426  
 mapToDouble() method, 264  
 mapToInt() method, 257, 264, 394–395  
 mapToLong() method, 264  
 mark() method, 316, 410–411  
 markSupported() method, 312–313  
 max() method, 394–395  
 memory exceptions, 133  
 method prefixes in JavaBean, 89  
 method references  
   colons, 231–232, 413  
   Consumer, 248  
   lambda expressions, 231–232  
 method signatures, 81, 83, 85, 87–88  
   checked exceptions, 124  
   Predicates, 145  
   valid, 152, 171, 233  
 methods  
   abstract. *See* abstract methods  
   abstract interface, 98  
   calling, 85–86, 92–93  
   chaining, 137  
   classes, 98  
   data type passing, 85  
   default, 110  
   encapsulation, 192  
   entry points, 4, 12  
   enum, 207  
   hiding, 108  
   interfaces, 103, 160, 216

abstract, 206, 211, 215–216  
 default, 110, 114, 214  
 modifiers, 112  
 restrictions, 110  
 static, 102  
 names, 87, 95, 164  
 NullPointerException class, 119  
 overloaded, 11, 81, 90, 107, 110, 112–113, 204  
 @Override annotation, 211  
 overriding, 99–101, 104, 108–109, 111, 113–114, 178, 197–198, 209, 423  
 pass-by-value, 81, 84, 88, 92, 179  
 primitives, 24  
 resource leaks, 304  
 return statement, 86, 90  
 static, 92  
 virtual, 111, 178  
 minus signs (-)  
   lambda expressions, 206–207  
   precedence, 152  
   purpose, 40  
 minusNanos units, 147  
 minutes  
   DateTimeFormatter, 293  
   formatter objects, 147  
 MissingResourceException class, 279, 375, 384  
 mkdir() method, 307, 313, 422–423  
 modifiers. *See* access modifiers  
 modulus operator (%)  
   precedence, 44  
   purpose, 40  
 months  
   adding, 297, 299  
   formatter objects, 147  
   indexes, 144, 397  
   invalid, 291, 296  
   LocalDate, 294  
 moving files, 323  
 multi-catch statements, 278, 282, 284–285  
 multi-dimensional arrays, 53  
 multiple catch blocks, 123  
 multiple inheritance, 100, 110, 213  
 multiplication operator (\*) precedence, 48, 152  
 mutability, 190  
 mutable Strings, 136–138

---

**N**

names  
  classes, 18, 23  
  enum, 389  
  methods, 87, 95, 107, 164  
  parameters, 224  
  primitive types, 25  
  variables, 18, 23, 162, 167  
    wrapper class, 25  
negative indexes, 56–57  
nested static classes, 212–213  
new keyword  
  constructors, 24–25  
  description, 19  
  required, 129  
newFixedThreadPool() method, 340  
newSingleThreadExecutor() method, 340  
newSingleThreadScheduledExecutor()  
  method, 340  
no-argument constructors, 83, 91, 94, 113  
normalize() method, 330, 332  
NotSerializableException class, 425  
null values  
  ArrayList, 232  
  initialized variables, 156  
  instanceof operator, 191, 194  
  primitives, 26  
  String, 25  
NullPointerException class, 41, 119, 124,  
  127, 129–130, 310, 329

---

**O**

ObjDoubleConsumer interface  
  parameters, 244  
  return types, 245  
Object class, 116  
object-oriented programming  
  description, 13  
  diagram, 4  
  encapsulation, 180  
  facets, 5  
ObjectDoubleConsumer interface, 241  
ObjectInputStream class, 303, 306, 313  
ObjectOutputStream class, 403

objects  
  accessible, 21–22  
  attributes, 113  
  casting, 176  
  creating, 53  
  equal, 195, 202, 237  
  instance variables, 203  
  instantiation, 19  
  vs. references, 105  
ObjLongConsumer interface, 401–402  
of() method  
  LocalDateTime, 296  
  LocalTime, 293  
ofDays() method  
  Duration, 292  
  Period, 298  
off-by-one errors, 140–141  
offer() method, 227  
offerLast() method, 351  
ofLocalizedDate() method, 292–293  
operators  
  evaluation order, 39, 43, 45–46  
  precedence, 38, 43–45, 48, 152  
Optional class  
  get(), 264  
  ifPresent(), 260–261  
  isEmpty(), 262  
  isPresent(), 259–261  
OptionalDouble interface  
  average(), 256–257  
  DoubleStream, 256  
OR operator (||)  
  description, 46  
  truth tables, 47–48  
  Venn diagram, 44  
order  
  ArrayList class, 137  
  operators, 39, 43, 45–46  
  sorting, 230, 232–235  
  try blocks, 116, 175, 272  
orElse() method, 390  
orElseGet() method, 390  
orElseThrow() method, 268  
OutputStream class, 306, 328  
overloaded methods, 11, 81, 90, 107, 110,  
  112–113, 204  
overridden methods  
  checked exceptions, 213–214, 279–280

covariant return types, 130, 220–221, 400–401  
 exceptions, 121, 128  
 @Override annotation, 211  
 overriding methods, 99–101, 104, 108–109, 111, 113–114, 178, 198–198, 209, 423

---

## P

package-private access modifier, 80, 82, 89, 173–174  
 packages  
   classes, 89, 161  
   description, 7  
   importing, 4–5, 13, 153  
   separating, 15  
 parallel processing in fork/join framework, 352–353  
 parallel reduction  
   collect(), 352  
   lambda expressions, 342  
 parallel streams, conversions to, 347  
 parameters  
   assert keyword, 279  
   formal types, 224, 231  
   lambda expressions, 139–140  
   overloaded methods, 107  
   Predicates, 138–139  
   types, 228–229  
 parent classes in no-argument constructors, 113  
 parentheses ()  
   balanced, 39  
   precedence, 38, 45  
   ternary expressions, 39, 46–47  
   try blocks, 275  
 parse() methods, 29  
 ParseException class, 273, 425  
 parseInt() method, 169  
 pass-by-value methods, 81, 84, 88, 92, 179  
 passwords, 315, 358–359, 394  
 Path interface and paths  
   breaking up, 330–331  
   creating, 330  
   cycles, 322  
   examples, 404, 410  
   files, 304  
   getFileName(), 323

getRoot() and toRealPath(), 320  
 symbolic links, 327–328  
 Paths class, 331  
 peek() method, 256, 260, 404  
 percent signs (%) precedence, 152  
 Period class, 145–146  
   chained methods, 175  
   getSeconds(), 290  
   getYears(), 294  
   ofDays(), 298  
 periods (.)  
   instance variables, 28–29  
   local variables, 23  
   in packages, 15  
   varargs parameter, 50, 56  
 platform independence, 9  
 plus signs (+)  
   evaluation order, 39  
   precedence, 152  
 plusDays() method, 144  
 plusNanos units, 147  
 political regions, 372  
 poll() method, 404  
 pollFirst() method, 351  
 polymorphism  
   examples, 188–189  
   virtual methods, 178  
 pop() method, 404  
 PosixFileAttributes class, 326  
 postfix operators, 38, 162, 350  
 precedence, operator, 38, 43–45, 48, 152  
 Predicates  
   method signature, 145  
   parameters, 138–139  
   removeIf(), 245  
   types, 142–144, 148  
 prefix operators, 38, 162  
 PreparedStatement interface, 359  
 previous() method, 366–367  
 primitive types  
   adding, 421  
   vs. arrays, 50  
   BiPredicate, 242  
   functional interfaces, 252  
   methods, 18–19, 24  
   names, 25  
   size, 20, 27  
   switch statements, 165  
   wrapper classes, 24, 31

print() method  
  Console, 308  
  String, 148  
PrintStream class, 305  
PrintWriter class, 309  
private access modifiers, 83, 87, 91, 98  
  encapsulation, 192  
  examples, 173–174  
Properties class  
  get(), 382–383  
  getProperty(), 382–383, 406  
  keys, 373  
  Map, 372, 375  
.properties files, 372  
protected access modifiers, 80, 95  
  abstract methods, 106  
  examples, 173–174  
protected methods  
  accessing, 197  
  virtual, 111  
public access modifiers  
  abstract interface methods, 98  
  classes, 5  
  encapsulation, 192  
  examples, 173–174  
  member access, 168

---

## Q

queues, 227

---

## R

race conditions  
  data corruption, 346  
  description, 339, 419  
read() method  
  InputStream, 302  
  user input, 311  
read/write operation order with threads, 343  
readAllLines() method, 324, 333, 408–409  
Reader class  
  description, 393  
  reader(), 302  
  writer(), 302  
reader() method, 302

readLine() method, 311  
readObject() method  
  Object, 392  
  ObjectInputStream, 306, 308–309  
readPassword() method, 311, 315  
recursion  
  arrays searches, 354–355  
  examples, 346–347  
RecursiveAction class, 342  
RecursiveTask class, 342  
reduce() method, 348  
reductions in terminal operations, 226–227  
references  
  attributes, 113  
  to classes, 106  
  constructors, 228, 230  
  lambda expressions, 231–232  
  vs. objects, 105  
  variables, 27  
regular files, 320  
relativize() method, 326, 404  
remainder operator (%)  
  division, 40  
  precedence, 44  
  purpose, 40  
remove() method  
  ArrayDeque, 404  
  ArrayList, 146  
removeIf() method  
  Collection, 233  
  Predicate, 245  
renameTo() method, 302  
REPLACE\_EXISTING value, 323  
replace() method, 148  
reserved words, 23  
reset() method, 316, 410–411  
resolve() method, 325–326, 404, 410  
resource bundles  
  Colors\_en.class, 376  
  Colors\_en\_US.class, 374  
  country codes, 378–379  
  creating, 374, 376–377, 379–380, 383  
  default, 381–382, 415  
  instantiation, 391  
  keys, 384  
  LocalDateTime, 375  
  required, 375  
  searching, 414–415  
  String, 372

- resource leaks, 304
- resource starvation, 339
- resources, closing, 420
- ResultSet interface
  - close(), 362
  - concurrency modes and drivers, 360
  - cursor position, 365
  - empty, 366
  - examples, 386
  - java.sql package, 393
  - scroll sensitive, 359, 363–364, 366–367
  - Statement, 368
  - types, 361
  - updating, 363–364, 407–408
- return statement
  - brackets, 143
  - methods, 86, 90
  - semicolons, 143
- return types and values
  - covariant, 100, 104, 113–114, 130, 220–221, 400–401
  - DoubleConsumer, 240
  - functional interfaces, 245
  - lambda expressions, 244
  - stream pipelines, 257
  - ternary operations, 48
- reverse() method, 139
- reverseOrder() method, 261–262
- rollback() method, 366, 368–369
- root directory, 304
- Runnable class, 339
- RuntimeException class, 116, 128–129, 132, 288
  - ArrayList, 141, 147
  - arrays, 165
  - primitive types, 20, 27
- skip() method, 302, 316
- slashes (/)
  - comments, 12, 166
  - file separators, 307
  - precedence, 44, 48, 152
- sort() method, 51
- sorted concurrent collections, 344
- sorted() method, 260–262, 265
- sorting
  - arrays, 51–52
  - order, 230, 232–235
- source files
  - compiling, 168
  - extension, 13
- sources, Stream, 236
- SQL statements
  - DELETE, 360
  - SQLException, 362
- SQLException class, 362, 365, 368, 425
- stack traces, 131
- StackOverflowError class, 124
- startsWith() method, 141–143
- Statement interface
  - auto-commit mode, 363
  - close(), 362
  - security for passwords, 315, 394
  - SELECT statements, 401
  - semicolons (;)
    - enum, 217–218, 395–396
    - resources, 404–405
    - return statement, 143
    - terminating statements, 8, 250
  - separating packages, 15
  - Serializable class, 302, 315, 317
  - setDefault() method, 373
  - ShortSupplier interface, 251
  - shutdown() method, 353
  - shutdownNow() method, 353
  - signatures, method. *See* method signatures
  - single pipe character (|) for exceptions, 272–273
  - singleton pattern
    - constructors, 407
    - description, 188
    - implementing, 190–191, 196
    - lazy instantiation, 200
  - size
    - ArrayList, 141, 147
    - arrays, 165
    - primitive types, 20, 27
  - skip() method, 302, 316
  - slashes (/)
    - comments, 12, 166
    - file separators, 307
    - precedence, 44, 48, 152
  - sort() method, 51
  - sorted concurrent collections, 344
  - sorted() method, 260–262, 265
  - sorting
    - arrays, 51–52
    - order, 230, 232–235
  - source files
    - compiling, 168
    - extension, 13
  - sources, Stream, 236
  - SQL statements
    - DELETE, 360
    - SQLException, 362
  - SQLException class, 362, 365, 368, 425
  - stack traces, 131
  - StackOverflowError class, 124
  - startsWith() method, 141–143
  - Statement interface
    - auto-commit mode, 363
    - close(), 362
    - security for passwords, 315, 394
    - SELECT statements, 401
    - semicolons (;)
      - enum, 217–218, 395–396
      - resources, 404–405
      - return statement, 143
      - terminating statements, 8, 250
    - separating packages, 15
    - Serializable class, 302, 315, 317
    - setDefault() method, 373
    - ShortSupplier interface, 251
    - shutdown() method, 353
    - shutdownNow() method, 353
    - signatures, method. *See* method signatures
    - single pipe character (|) for exceptions, 272–273
    - singleton pattern
      - constructors, 407
      - description, 188
      - implementing, 190–191, 196
      - lazy instantiation, 200
    - size
      - ArrayList, 141, 147
      - arrays, 165
      - primitive types, 20, 27
    - skip() method, 302, 316
    - slashes (/)
      - comments, 12, 166
      - file separators, 307
      - precedence, 44, 48, 152
    - sort() method, 51
    - sorted concurrent collections, 344
    - sorted() method, 260–262, 265
    - sorting
      - arrays, 51–52
      - order, 230, 232–235
    - source files
      - compiling, 168
      - extension, 13
    - sources, Stream, 236
    - SQL statements
      - DELETE, 360
      - SQLException, 362
    - SQLException class, 362, 365, 368, 425
    - stack traces, 131
    - StackOverflowError class, 124
    - startsWith() method, 141–143
    - Statement interface
      - auto-commit mode, 363
      - close(), 362

## S

- scheduleAtFixedRate() method, 345
- ScheduledExecutorService class, 345, 355
- scheduleWithFixedDelay() method, 424
- scope, 5, 8–9, 159–160
- scroll sensitive ResultSets, 359, 363–364, 366–367
- searching
  - arrays, 51–52, 62, 176, 181, 354–355
  - resource bundles, 414–415
  - unsorted arrays, 60

- implementing, 359
- ResultSets, 368
- statements
  - order, 10
  - terminating, 8, 250
- static classes
  - inner, 216
  - nested, 212–213, 221
- static members
  - access, 168
  - importing, 95–96, 166
  - initializers, 201–203
- static methods, 92
  - dates, 299
  - enum, 216
  - examples, 196–197
  - inner classes, 211, 214–215
  - interfaces, 102, 160
  - variable access, 163, 194
- static variables, 82
  - description, 93, 192
  - hiding, 108
  - inner classes, 214–215
  - instance methods, 90
  - referencing, 27
  - scope, 159–160
- Stream class and streams, 408
  - Character objects, 260
  - Comparator, 262
  - conversion to parallel streams, 347
  - creating, 265–266
  - Files methods, 426
  - filter(), 229
  - forEach(), 224
  - infinite, 262, 264
  - lists, 228, 257
  - Map conversion, 258, 263
  - reusing, 230–231
  - source, 236
  - Writer, 302
- stream pipelines
  - filters, 236
  - intermediate operations, 232, 260–261
  - return value, 257
  - terminal operations, 225–226, 230, 234, 402–403, 415
- String class
  - comparisons, 154–155
  - concat(), 140
  - declarations, 180–181
- element changes, 141
- package, 140
- replace(), 148
- resource bundles, 372
- vs. StringBuilder, 136
- String data type
  - array elements, 51
  - charAt(), 237
  - comparisons, 34, 41–43
  - empty, 232
  - garbage collection, 27
  - immutable, 142–143, 160
  - instance variables, 25
  - mutable, 136–138
  - startsWith(), 141–142
- StringBuilder class, 69–70
  - creating, 136
  - indexes, 152
  - reverse(), 139
  - vs. String, 136
  - working with, 144, 146, 154, 164
- subclasses, 199–200, 207
- submit() method, 338, 349
- subpath() method, 332
- subtracting days, 295
- subtraction operator (-)
  - precedence, 44
  - purpose, 40
- suffixes for double type, 20
- sum() method, 421
- super() command, 80, 86
- superclasses
  - casting, 109
  - references to, 106
- Supplier interface
  - description, 241
  - get(), 246
  - inputs, 240
- switch statements
  - break statements, 35
  - case statements, 37, 47, 171–172
  - default statements, 35, 47, 163, 183
  - variables, 34, 165, 208
- symbolic links
  - description, 320
  - paths, 322, 327–328
- synchronization
  - blocks, 345
  - lists, 353
  - threads, 343

System  
 console(), 310  
 getProperty(), 307  
 System.err stream, 310–311  
 System.in stream, 310–311  
 System.out stream, 328

---

## T

tables, empty, 360  
 TemporalUnit interface, 298  
 terminal operations  
   reductions, 226–227  
   stream pipelines, 225–226, 230, 234,  
     402–403, 415  
 terminating statements, 8, 250  
 ternary operations  
   data types, 41  
   evaluating, 34–35, 42  
   examples, 173  
   if-then-else replacement, 35  
   parentheses (), 39, 46–47  
   returned value, 48  
 this() constructors, 81  
 this keyword, 99  
 Thread class and threads, 339  
   active, 399  
   examples, 349–351  
   hanging code, 340–341  
   parallel processing, 352–353  
   read/write operation order, 343  
 thread-safe classes  
   examples, 345–346, 409  
   instances, 338  
   LinkedBlockingDeque, 341  
   synchronized lists, 353  
 throw keyword, 117, 126, 132–133, 177, 272  
 Throwable class, 117, 119, 127  
 throws keyword, 117, 126, 132–133, 177,  
   272  
 time zones, 290–291, 296, 298, 372  
 times  
   classes, 143–144  
   comparisons, 290, 298, 419  
   immutable, 149, 299, 389–390  
   intervals, 145  
 toAbsolutePath() method, 329–330  
 ToDoubleBiFunction interface, 241, 248

ToDoubleFunction interface, 250  
 ToIntBiFunction interface, 251  
 ToLongBiFunction interface, 401–402, 412  
 ToLongFunction interface, 241  
 toMap() method, 258, 425–426  
 toRealPath() method, 320, 329  
 toString() method, 142  
   arrays, 55  
   availability, 188  
   dates, 294  
   StringBuilder, 136  
 TreeMap  
   collections, 224  
   Comparable, 225  
   description, 224  
 TreeSet  
   Comparable, 225  
   duplicates in, 226  
 trim() method, 145–146  
 truth tables  
   and operators, 37–38  
   disjunctive logical operator, 47–48  
   XOR operator, 43  
 try blocks  
   examples, 179–180  
   order, 116, 175, 272  
   parentheses, 275  
   requirements, 116, 120, 125, 158, 272  
   scope in, 117  
 try-with-resources, 275, 278, 280–281  
   declarations, 404–405  
   java.io, 408  
   requirements, 406  
 two-dimensional arrays, 51, 59  
 TYPE\_FORWARD\_ONLY value, 361

---

## U

UnaryOperator interface  
   inputs, 244  
   lambda expressions, 248  
 unchecked exceptions  
   examples, 172, 283  
   handling, 122–123  
   java.util.MissingResourceException, 279  
 underscores (\_) in literals, 19, 23, 27  
 UnexpectedException class, 388  
 unsorted arrays, searching, 60

UnsupportedTemporalTypeException class,  
147  
until() method, 298  
up-converting data types, 94  
updateRow() method, 363–364, 407–408  
updating ResultSets, 363–364, 407–408  
URL formats for databases, 358–359  
user input, 310–311, 314  
usernames, 358–359

---

## V

valueOf() methods, 29  
varargs parameter  
  vs. array parameters, 50–51  
dots, 56  
element retrieval, 50  
symbol, 50  
variables  
  accessible, 5–6  
  assigning, 25, 154  
  catch blocks, 122, 130–131, 133, 274  
  classes, 7  
  comparisons, 154  
  declarations, 18, 91, 195, 200, 204  
  encapsulation, 192  
  final, 86, 88, 93  
  hiding, 108  
  initializing, 156  
  inner classes, 220  
  local. *See* local variables  
  multi-catch statements, 278  
  names, 18, 23, 162, 167–168  
  nested classes, 221  
  private, 98  
  protected, 95  
  referencing, 27  
  scope, 5, 8–9, 159–160  
  static. *See* static variables  
  switch statements, 34, 165, 208  
Venn diagram for logical OR operator, 44  
virtual methods, 111, 178

---

## W

walk() method, 321  
while loops, 64  
  brackets, 182–183  
  conditions, 65–66, 69–70, 156,  
    160–161  
  examples, 67, 76  
White space, 145–146  
wildcards in import statements, 9  
withFlatMap() method, 268–269  
withoutFlatMap() method, 268–269  
wrapper classes, 21  
  int, 19  
  lambda expressions, 249  
  names, 25  
  primitives, 24, 31  
write() method, 311–312  
Writer class  
  abstract, 302  
  vs. OutputStream, 306  
writer() method, 302

---

## X

XOR operator (^)  
  description, 44  
  truth tables, 43

---

## Y

years, retrieving, 294

---

## Z

zero based arrays, 50  
zero-length arrays, 47  
ZonedDateTime class, 293

# Comprehensive Online Learning Environment

Register on [Sybex.com](http://Sybex.com) to gain access to the comprehensive online interactive learning environment and test bank to help you study for your OCA and OCP Java SE 8 Programmer certifications.

---

The online test bank includes:

- **Practice Test Questions** to reinforce what you learned
- **Bonus Practice Exams** to test your knowledge of the material

Go to <http://www.wiley.com/go/sybextestprep> to register and gain access to this comprehensive study tool package.

## 30% off On-Demand IT Video Training from ITProTV

ITProTV and Sybex have partnered to provide 30% off a Premium annual or monthly membership. ITProTV provides a unique, custom learning environment for IT professionals and students alike, looking to validate their skills through vendor certifications. On-demand courses provide over 1,000 hours of video training with new courses being added every month, while labs and practice exams provide additional hands-on experience. For more information on this offer and to start your membership today, visit <http://itpro.tv/sybex30/>.

