

# Practice Your Java<sup>TM</sup>

Level 1

First Edition

Plurium<sup>TM</sup> Programming Practice Series



## **Practice Your Java Level 1, 1<sup>st</sup> Edition**

Author: Ayodele M. Agboola

Copyright © 2016 Ayodele M. Agboola. All rights reserved.

Published by **Plurium Press**

[www.pluriumpress.com](http://www.pluriumpress.com)

First Edition: January, 2016

Edition 1 Revision 0 2016/01

Kindle Edition

Print Edition ISBNs

**ISBN-13: 978-0-9961338-1-4**

**ISBN-10: 0-9961338-1-X**

### **Downloads & Errata**

Errata and downloads for this book are available at [www.pluriumpress.com](http://www.pluriumpress.com).

To report any errors found in this book, please email the publisher at [practiceyourjava1@pluriumpress.com](mailto:practiceyourjava1@pluriumpress.com).

### **Sales Information**

For bulk purchases or for customized variants, please contact the publisher at [sales@pluriumpress.com](mailto:sales@pluriumpress.com).

### **Trademarks**

Java is a trademark of Oracle Corporation and/or its affiliates.

Microsoft, Windows®, Notepad® and C# are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks remain the property of their respective owners.

### **Disclaimer**

Care has been taken in the preparation of this book; however, no warranty express or implied of any kind is made for the content thereof or for errors or omissions herein. No liability is assumed by the author, publisher or seller for any damages in connection with or arising out of the use of any of the content of this book.

### **Rights**

All rights reserved. No part of this publication may be reproduced, stored or transmitted in any form without the express written permission of the publisher.



# Introduction

Welcome to *Practice Your Java Level 1!* This is a book designed to, through practice, improve the knowledge, skill and confidence of the reader in Java programming. Over 1000 carefully designed exercises along with their solutions are presented in this book to help achieve this end.

On the back cover of the print version of this book, we presented four important objectives that this book is designed to help you achieve. We want to help you *quickly* achieve the following:

- (1) Attain a firm understanding of the foundations of this language
- (2) Have high retention of the knowledge presented in this book,  
thus increasing your (3) skill and (4) confidence in your ability to program in Java at this level and to pursue higher levels of study of this language.

There are two main means by which this book helps achieve these objectives; first by properly ordering the chapters and the contents therein and also, by our unique question structure. We explain these below.

## ***Understanding, Retention, Skill, Confidence by Proper Chapter & Question Order***

I have the following saying regarding flying aeroplanes; “despite the vast number of buttons in the cockpit, an aeroplane is easy to fly if you just learn things in the right order”. When learned and understood in the right order, you will easily remember where each button is and what it does. The same concept holds true for learning Java. Therefore in this book, the presentation of the sub-topics of the language is put in the appropriate order for learning this particular language, thus aiding retention. As the reader proceeds from chapter to chapter, their knowledge and skill in Java is built a methodical fashion.

## ***Understanding, Retention, Skill, Confidence by Optimal Question Structure***

It was stated that one of the objectives of this volume is to help aid understanding and also to help aid retention of the material learnt. Before the ubiquity of the internet, a developer had to retain a high degree of knowledge (not just facts) in his head, referring to texts only when necessary. However, now, due to improper learning of and practice in the language at hand, a developer may spend a lot of time online looking for “point solutions” for problems, rather than taking the time to learn the foundation of the language properly.

So how does this volume aid understanding? How does it aid retention? How does it help speed up learning? It aids understanding, retention and speed in learning by the following structure of the exercises:

### **The reader is often presented one concept/method at a time per exercise**

Like when we learnt the alphabet, we learned it letter by letter. Therfore, in this volume, an appropriate number of programming exercises require the use of only a single Java method as the key to the solution of the exercise. This allows the reader to focus on this single method, thus aiding understanding and memorization.

Of course, questions with multiple concepts are presented as well.

### **The reader is often presented with direct hints for programming questions**

This is a very important feature of the exercises in this book. We explain why.

You see, if an exercise is presented on a topic with which a reader is unfamiliar or peripherally familiar, the reader will search (likely on the internet) for possible solutions. This takes time. This

## Practice Your Java Level 1

wastes time. In fact, for more complex topics, a reader many end up spending so much time reading potential solutions, that when he finally finds a solution he has actually forgotten how he got there and is too fatigued to actually remember the solution! It is better, as is done in this volume, to have “hints” which specify the exact Java methods that feature in the solution to the problem so that the reader can spend his time profitably studying the exact methods/language features that pertain to solving the problem at hand and implementing the solution. This focus also aids retention.

### Inverse question structure

There are some topics which are somewhat recondite. Instead of requiring that the reader come up with solutions to questions on such topics, completed code is presented as the question, and the reader is asked to analyze the presented solution. The intent of such exercises clearly is to teach the reader the involved topic at hand. Again this saves time and lets the user focus on learning the material instead of spending time searching at length for a solution.

The reader will find this question structure to be most helpful.

The astute reader will actually observe that while indeed this is called a practice volume, the structure of many of the solutions is as one actually teaching the Java topics presented.

## Usage Scenarios For This Book

It is envisaged that this book will fulfill the needs of the beginning to early intermediate Java developer and for those seeking a hands-on review of the topics presented herein. The following usage scenarios are foreseen for this book:

- For the individual without previous experience in Java, this book is intended as a companion text to a Java programming volume to help the reader broaden and solidify their understanding, retention, skill and confidence in Java by way of practice exercises.
- For the returning Java programmer at this level, working through the exercises in this book may suffice as a text, given the depth and extent of the exercises and presented solutions.
- The programmer who prefers to learn by practice exercises will find this to be an appropriate volume.
- The classroom instructor, who will be delighted to find that this book is a robust tool for ensuring that their students are well versed in and have proper depth in Java programming.

## Intended Audience

The primary audience for this volume is the Java beginner up to the early intermediate Java programmer.

## Toolset

The coding solutions in this book were implemented in Java 8 using version 4.4 of the Eclipse IDE. There are no IDE specific exercises in this volume. The reader is encouraged to use any IDE of their choosing.

## **Structure of Contents**

This book is structured in chapters, with each chapter covering a major topic (or part thereof). Overlap between topics is covered in the appropriate chapter for such.

### **Chapter Legend/Structure**

Exercises are numbered sequentially. The question numbers are uniformly on the left-hand side of the page. The solution to each exercise appears beside or immediately below the exercise.

An **M** appearing in the question number column indicates an action which must be performed manually; for example, an instruction to insert a USB stick/drive into the computer.

An **E** followed by a number in the question number column indicates that this is an extra unsolved exercise given to you the reader to solve. Where these appear, they are at the end of the chapter.

## **How to use the book**

The individual who is fairly new to Java is advised to go through the chapters in the book sequentially.

A classroom instructor or the individual person who is simply refreshing their Java knowledge can go through the individual chapters in an order that best suits their needs.

## **What You Need To Use This Book**

In order to implement all of the exercises in this book you need to download and install the Java 8 JDK as well as your Java IDE of choice.

The user will also need access to the freely available official Java online documentation supplied by Oracle.

The code for this book is downloadable from the publishers' website.

## **About The Author**

Ayodele “Ayo” Agboola holds a degree in Electrical Engineering from the University of Waterloo in Canada. He has had the pleasure of programming over the course of 30 years, first personally and then professionally. As a professional he has worked globally in many roles, including software development, project management and training, implementing software of various scale, largely for telecom service providers.



## *Acknowledgements*

*Oluwa lóṣe èyí; ó sì jé ohun iyanu lójú wa!*

*And to all the people whose shared knowledge has contributed to this.*



## Table of Contents

Chapter 1. For Starters.....	1
Chapter 2. Console Handling I: Basic Output .....	7
Chapter 3. Number Handling I – Numerical Primitives .....	11
Chapter 4. Boolean operations - The <code>boolean</code> Primitive .....	41
Chapter 5. Strings: Basic Handling .....	47
Chapter 6. Conditional Processing .....	55
Chapter 7. Arrays .....	61
Chapter 8. Iterations/Loops .....	69
Chapter 9. Number Handling II – Numeric Wrapper Classes.....	81
Chapter 10. The <code>Boolean</code> Wrapper Class .....	89
Chapter 11. Enumerations I .....	91
Chapter 12. Number Handling III – <code>BigInteger</code> & <code>BigDecimal</code> .....	95
Chapter 13. Number Handling IV – <code>strictfp</code> & <code>StrictMath</code> .....	109
Chapter 14. Number Handling V – Unsigned Integers .....	111
Chapter 15. Date/Time Handling – Foundation .....	117
Chapter 16. Exceptions .....	137
Chapter 17. Console Handling II: Console Input.....	147
Chapter 18. Console Handling III: Command Line Input.....	155
Chapter 19. The <code>char</code> primitive & the <code>Character</code> Wrapper Class .....	157
Chapter 20. Random Numbers.....	163
Chapter 21. <code>StringBuffer</code> & <code>StringBuilder</code> .....	167
Chapter 22. Classes I .....	175
Chapter 23. Object References & Primitive Type Variables: A Deeper Understanding.....	201
Chapter 24. Classes II: Nested Classes, Local Classes, Anonymous Classes.....	209
Chapter 25. Collections I: Collections with Generics .....	219
Chapter 26. Interfaces .....	231
Chapter 27. Collections II: Collections using Interfaces, along with Iterators .....	241
Chapter 28. Properties, Computer Environment & Computer Information.....	247
Chapter 29. File System I: Path, Directory and File Handling using NIO/NIO.2 .....	253
Chapter 30. File System II: File Input & Output.....	277
Chapter 31. Enumerations II .....	285
Chapter 32. Graphical User Interfaces using Swing.....	289
Chapter 33. Applets – a brief look .....	305

## **Practice Your Java Level 1**

Appendix A. Classes, Objects & Methods: An Introduction .....	307
Appendix B. Project .....	311
Index .....	313

# Chapter 1. For Starters

The intent of this chapter is largely to provide a quick overview of introductory topics with which the reader may already be fully acquainted; these topics pertain to the basic structure of Java applications as well as a cornucopia of basic Java programming topics and concepts. Again, as previously stated, the exercises in this chapter are presented as a quick overview.

**Note:** The reader who needs an introduction to the concepts of classes, objects and methods should first read Appendix A.

---

First, we present a sample program below; we will reference it in some of the exercises in this chapter.

```
package practiceYourJava;  
import java.util.*;  
import java.time.*;  
import java.nio.file.*;  
import java.nio.file.attribute.*;  
  
public class PracticeYourJava {  
  
    public static void main(String[] args) {  
  
    }  
}
```

1.	With respect to the code block shown above, what does the statement <code>package practiceYourJava;</code> mean? This means that we want the program that we are writing to be placed in a “package” which we named <code>practiceYourJava</code> ( <i>the concept of packages is elucidated upon in the next exercise</i> ).
2.	With respect to the code block shown above, give a very brief explanation of what the meaning of the statements <code>import java.util.*; import java.time.*; import java.io.*; import java.nio.file.attribute.*;</code> Each of these statements refers to our intent to <code>import</code> a <u>package</u> ; by “import” we mean that we are informing the compiler that we intend to use part of the contents of the stated package(s) in our program. Our final compiled program will be whatever program we write plus the particular components of the noted packages that use in our program.
3.	What is a “package” in Java? A “package” is a grouping of Java <u>types</u> (by <u>types</u> here we mean classes of various sorts, including enumeration classes, exception classes and interfaces; we will look at each of these in later chapters). It is up to the developer of the package to group whatever types they want into a given package. An example of a package is the <code>java.nio.*</code> package which contains java classes that pertain to certain aspects of file I/O. We have to import this package for use in our program in order to use the classes and methods with which we perform file I/O. So, why do you have to inform the compiler that you want to <code>import</code> particular packages into your program? The reason is so that the compiler knows where to find the previously built types that you want to use in your program for the following two reasons: (1) so that it can ensure that you are referring to the types correctly and (2) so that it can compile those previously existing types along with your program. The package content that you reference via <code>import</code> statements is in effect a part of your program (note though, when you reference standard Java libraries in your program their content is not compiled with your program because those packages are already present in the JVM; at run-time whenever the JVM sees a reference in your code to a component of a standard Java library it simply links to the version/copy of the library code that it already has). Now what does the “*” mean in the package name? It indicates to the compiler that we want every top-level type that appears in the package in question to be made visible for potential use in our program. For example, in the package <code>java.util</code> there are a number of classes; these include the classes <code>Random</code> , <code>LinkedList</code> , <code>HashMap</code> and

## Practice Your Java Level 1

	<p><code>ArrayList</code>, to name a few. Using a “*” (i.e. <code>java.util.*</code>) lets the compiler see all of these types for potential compilation with our application code.</p> <p>You are also free to specify the individual types that you want to use directly in an <code>import</code> statement; for example you can write <code>import java.util.HashMap;</code> instead of <code>import java.util.*;</code>; if you know that the <code>HashMap</code> class is the only item in the package <code>java.util</code> that you want to use in your program. Such precise selection only makes compilation faster, it doesn’t have any impact on the size of your program.</p> <p>You too can create your own packages. In the code example presented above, we are putting the contents of the program in this example into a package named <code>practiceYourJava</code>. In fact, your IDE might prompt you to specify a package name when you try to create a class. If you do not define a package, the IDE might use the “default package”, which doesn’t have a name and thus its contents cannot be referenced from other packages.</p> <p>We also see in our sample code the importation of the packages <code>java.nio.file.attribute.*</code> and <code>java.nio.file.*</code>, where on observation of the names we see that the first package name appears to be a subset of the first. This is not so; the concept of nested or sub-packages does not exist in Java. They are different packages whose names have been made similar just to show the relationship between their contents.</p> <p><i>Default imports</i></p> <p>Java by default automatically imports the following:</p> <ol style="list-style-type: none"><li>(1) the package <code>java.lang.*</code> which contains a lot of useful classes such as the class <code>Math</code></li><li>(2) the other types of the package that your program is code is a part of</li></ol> <p><b>In summary:</b> a package is a grouping of types. You must state in your program that you want to <code>import</code> any package whose contents you use in your application.</p> <p>Note: A good IDE will inform you as to the need to import a given built-in package once it sees you referring in your code to any of the contents of the package in question.</p>
4.	<p>In the code block presented at the beginning of the chapter, we see the following statement:</p> <pre>public class PracticeYourJava {</pre> <p>What does it mean?</p> <p>It means that we are creating our own class that we have named <code>PracticeYourJava</code>.</p> <p>All code in Java applications is always in a class.</p> <p>The keyword <code>public</code> means that this class is accessible by any other class.</p> <p>In this book, we will use a good number of built-in classes and also eventually create our own.</p>
5.	<p>True/False: In Java, should each type (class, interface, enumeration, etc) that you create be in its own file?</p> <p>True. The file name should also be the name of the contained type, with the extension <code>.java</code> applied to it. For example, the class <code>PracticeYourJava</code> should be in a file named <code>PracticeYourJava.java</code>.</p>
6.	<p>What is the concept of a “classfile” in Java?</p> <p>In Java, a classfile is the result of the Java compilation process applied to a <code>.java</code> file. The content of this file is the compiled bytecode that the Java Virtual Machine (JVM) runs. The output file has the same name as the input file, however now with an extension of <code>.class</code> instead of the extension <code>.java</code>.</p> <p>This bytecode is unique to the language Java itself. You can think about it as “Java machine code”.</p>
7.	<p>What is the Java Virtual Machine (JVM)?</p> <p>The Java Virtual Machine (JVM) is the software in which your Java application runs, as Java programs do not run directly on the processor like say a program written in C will. The JVM interprets your classfiles (which contain Java bytecode) and translates their contents to instructions for the particular environment (Windows, UNIX, Linux, etc) that the JVM is running on.</p> <p>The JVM itself runs directly on the processor. The existence of the JVM concept is what facilitates Java’s “write once run anywhere” mantra, as the bytecode in the classfile can be understood as Java’s “machine code” which all JVMs, irrespective of the operating system or microprocessor that they are running on, understand and translate for their own particular environment.</p>
8.	<p>What is the name of the method for which the compiler looks in a classfile to invoke/execute when you direct that your program be run?</p> <p>The method <code>main</code>.</p> <p>Every class that you want to run directly must always have a method named <code>main</code> (method names are case-</p>

	sensitive). Classes which do not have a method <code>main</code> defined in them cannot be run directly; they can only be used by other classes as components.
9.	What are the implications if your <code>.java</code> file does not contain a method <code>main</code> ? <i>Hint: See preceding exercise</i> By not putting a method <code>main</code> into the <code>.java</code> file in question, we are saying that we do not want its resulting classfile to be directly runnable. This is very common, as most <code>.java</code> files simply contain type definitions that other programs use as components.
10.	Write out the different possible method declarations that a given method <code>main</code> that we want to be callable/invokable directly by the JVM can have. The only possible method declaration that fits the noted criteria for method <code>main</code> is as follows: <pre>1. public static void main(String[] args)</pre> The access modifier <code>public</code> gives the method the visibility for it to be seen and thus be invokable by the JVM. The method <code>main</code> cannot return a value to the operating system directly (there are other methods such as <code>System.exit</code> with which it can if it wants to) and therefore it is assigned a return type of <code>void</code> . The method <code>main</code> must always be ready to receive an array of type <code>String</code> as can be seen in the input type. Also, <code>main</code> always has the modifier <code>static</code> applied to it (We discuss this modifier in Chapter 22 entitled <i>Classes I</i> ).
11.	Apart from the method type modifier <code>static</code> , what other method type modifier can be used for the method <code>main</code> ? No other. The method <code>main</code> is <u>always</u> a <code>static</code> method. Method modifiers (such as <code>static</code> ) are covered in Chapter 22, entitled <i>Classes I</i> .
12.	In the code snippet below, we have the keyword <code>return</code> . What does this keyword do? <pre>public static void main(String[] args){     return; }</pre> The keyword <code>return</code> , when executed, causes the immediate termination of the method in which it is invoked/called and returns control to the calling method. In the case of the method <code>main</code> which is called by the operating system, a <code>return</code> statement in <code>main</code> exits the program and returns control to the operating system. In summary, we can say that <code>return</code> means “exit the method that you are running now and return control back to whoever called this method”.
13.	Contrast the use of the keyword <code>return</code> in this program with its use in the preceding exercise. <pre>public int doSomething(){     return(5); }</pre> The return statements in this and the prior exercise have the same basic effect in that they return control to the method that invoked the method that invoked the <code>return</code> statement, however, in this exercise, the <code>return</code> also “returns” a desired value to its calling method (in this example it is returning the value 5). A very important point to note is that the ability of the statement <code>return</code> to return anything is linked to the return type specified in the method definition in which the <code>return</code> statement appears. This particular example says that we are returning a value of type <code>int</code> and that declaration makes it legal for us to return the <code>int</code> value 5.
14.	Explain why the following code is invalid: <pre>public String void main(String[] args){     return("finished"); }</pre> It is invalid because the method <code>main</code> can cannot return anything directly; its return type should always be <code>void</code> .
<b>Comments</b>	
15.	What does the string sequence <code>//</code> mean when it appears on a line in a program? This double forward slash indicates that whatever text comes after it on the same line is a comment, therefore the compiler should ignore it.
16.	What is the output of the following code fragment? <pre>/*</pre>

## Practice Your Java Level 1

	<pre>System.out.println("Hello"); System.out.println(); */</pre> <p>Nothing will be output. The reason is because the whole block of code in the fragment is commented, as indicated by the starting comment delimiter <code>/*</code> and the ending comment delimiter <code>*/</code>. The delimited code in the question is an example of a <i>multi-line comment</i>.</p>										
	<p><b>A brief look at instance members &amp; static members</b></p> <p>The objective of the following questions is to refresh the reader on the difference between instance members and static members of a class. This is useful because we meet a number of these before we actually get to the chapter which discusses classes at length.</p>										
17.	<p>We have the following class definition:</p> <pre>class Man{     public static String gender="male";     public String firstName;     public String lastName; }</pre> <p>Indicate which are the static members of the class and which are the instance members.</p> <p>The only static member of this class is the <code>String</code> field <code>gender</code>. This is discernible by the modifier <code>static</code> in its definition.</p> <p>The <code>String</code> fields <code>firstName</code> and <code>lastName</code> are the instance members of the class.</p>										
18.	<p>Show the instantiation(creation) of an object of class <code>Man</code> named <code>man01</code> in the method <code>main</code> of class <code>Man</code>. After this, set the <code>firstName</code> field of the object <code>man01</code> to “Adam”, then print out the field <code>man01.firstName</code>.</p> <pre>class Man{     public static String gender="male";     public String firstName;     public String lastName;      public static void main(String[] args){         Man man01 = new Man(); // We have now created an object named man01 of the class Man.         man01.firstName = "Adam"; // firstName is a "public" field so we can set it in any method         System.out.println( man01.firstName);     } }</pre>										
19.	<p>What is the difference between a static class member and an instance member?</p> <p>A static class member is a member which “belongs” to the class, whereas an instance member “belongs” to the individual instances of the class.</p> <p>A static class member is invoked relative to the class, whereas an instance member is invoked relative to an instantiated object. We can see this by the following example 4-line code fragment that uses the earlier defined class <code>Man</code>:</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; width: 45%;">Code</th> <th style="text-align: center; width: 45%;">Commentary</th> </tr> </thead> <tbody> <tr> <td><code>Man man01 = new Man();</code></td> <td>instantiation of an object named <code>man01</code> of class <code>Man</code></td> </tr> <tr> <td><code>man01.firstName = "John";</code></td> <td>observe that the instance field <code>firstName</code> is called relative to an instance</td> </tr> <tr> <td><code>man01.lastName = "Doe";</code></td> <td>the same observation as above applies to this instance field too</td> </tr> <tr> <td><code>System.out.println(Man.gender);</code></td> <td>observe how the <code>static String gender</code> is accessed; it is accessed relative to the class name (<code>Man</code> in this case), NOT relative to an object name. It is <u>NOT</u> accessible as <code>man01.gender</code> (you can try it in order to see the result), as static members are always accessed relative to the class.</td> </tr> </tbody> </table>	Code	Commentary	<code>Man man01 = new Man();</code>	instantiation of an object named <code>man01</code> of class <code>Man</code>	<code>man01.firstName = "John";</code>	observe that the instance field <code>firstName</code> is called relative to an instance	<code>man01.lastName = "Doe";</code>	the same observation as above applies to this instance field too	<code>System.out.println(Man.gender);</code>	observe how the <code>static String gender</code> is accessed; it is accessed relative to the class name ( <code>Man</code> in this case), NOT relative to an object name. It is <u>NOT</u> accessible as <code>man01.gender</code> (you can try it in order to see the result), as static members are always accessed relative to the class.
Code	Commentary										
<code>Man man01 = new Man();</code>	instantiation of an object named <code>man01</code> of class <code>Man</code>										
<code>man01.firstName = "John";</code>	observe that the instance field <code>firstName</code> is called relative to an instance										
<code>man01.lastName = "Doe";</code>	the same observation as above applies to this instance field too										
<code>System.out.println(Man.gender);</code>	observe how the <code>static String gender</code> is accessed; it is accessed relative to the class name ( <code>Man</code> in this case), NOT relative to an object name. It is <u>NOT</u> accessible as <code>man01.gender</code> (you can try it in order to see the result), as static members are always accessed relative to the class.										

## Some points on methods

The following set of exercises pertain to the concept of methods.

20.	In Java, what is meant by the term “ <i>method signature</i> ”?
	<u>Method signature</u> in Java means the following: the method name and the types of its input parameters in order.

	<p>For example, for a <b>public</b> method named <b>addIntegers</b> which adds two integers passed to it and returns a type of <b>long</b>, the method signature is:</p> <pre>addIntegers(int, int);</pre>
21.	<p>What is meant by the term “<i>method declaration</i>”?</p> <p><u>Method declaration</u> means the following: the method name, its access scope (whether <b>public</b>, <b>private</b>, etc.) the input types and parameter names in the order in which they are sent to the method and the output type of the method.</p> <p>For example, for a <b>public</b> method named <b>addIntegers</b> which adds two integers <b>x</b> and <b>y</b> passed to it and returns a type of <b>long</b>, the method declaration is:</p> <pre>public long addIntegers(int x, int y);</pre>
22.	<p>What kind of types can be returned by a method using a <b>return</b> statement?</p> <p>Any type can be returned by a method, whether a user defined type or a built-in type (the method <b>main</b> is a special case method which cannot use <b>return</b> to return anything).</p>
23.	<p>What is meant by the term <i>method overloading</i>?</p> <p>This means that for a given method name, there can be different versions of it with different input types and it is up to the Java compiler to call the correct one (the compiler determines which one to invoke by looking at the types of the parameters that you pass to it).</p> <p>For example, if I have the following methods which have the same name but different input parameters:</p> <pre>addIntegers(int, int); addIntegers(int, int, int);</pre> <p>then if another method invokes <b>addIntegers</b> passing three integer values to it, the compiler knows that I mean to call the second version of <b>addIntegers</b> indicated above; if I invoke <b>addIntegers</b> passing only two integers to it, the compiler knows to invoke the first version of <b>addIntegers</b> shown above.</p> <p>Note however, that a method cannot be overloaded only with respect to the return type, for example the following pair cannot jointly appear in the same class:</p> <pre>long    addIntegers(int, int); int     addIntegers(int, int);</pre> <p>The compiler would not know which of the two methods to call since both have the same input parameters. You will come across a number of overloaded methods in the built-in Java classes.</p>
24.	<p>In a class named <b>PracticeYourJava</b>, create a <b>static</b> method named <b>greet</b> which does not return anything and does not take any parameters. All it should do is print out the statement “<i>Hello, how are you?</i>” and then return to its caller. Call this method from the method <b>main</b>.</p> <pre>public class PracticeYourJava{     static void greet(){         System.out.println("Hello, how are you?");     }     public static void main(String[] args) {         greet(); //calling the method hello     } }</pre>
25.	<p>Write a <b>static</b> method named <b>addIntegers</b> with a return type of <b>int</b> that adds two integers and returns the resulting integer value.</p> <p>To test the method, pass two integer values of choice from <b>main</b> to <b>addIntegers</b> and have <b>main</b> print out the result.</p> <pre>public class PracticeYourJava {     static int addIntegers(int a, int b) {         return(a + b);     }     public static void main(String[] args) {         int result = addIntegers(5, 2);         System.out.println(result);     } }</pre>

## Practice Your Java Level 1

### Variable number of Arguments (varargs)

**Note:** While this topic is more advanced than a first chapter topic, it is placed here for consistency. The reader is free to peruse this topic lightly and then come back to it when a need for the feature arises in later chapters.

26.	<p>What is the concept of a variable number of arguments (varargs)?</p> <p>This is a mechanism by which we can pass a variable number of arguments of a given type to a method designed to receive a variable number of arguments. For example, we can have a method which we use to calculate and return the sum of a variable number of integers passed to it; i.e. any method calling such a method can put any number of integers in the parameter list that they send to this method. The called method would simply note the number of items passed to it and add them up using a loop.</p> <p>The variable that receives a variable number of arguments must be the last parameter in the method declaration. Also, there can only be one variable that can receive a variable number of arguments in a method declaration and it must be the last variable in the method declaration.</p> <p>It should be noted that for general intents and purposes the variable that receives a variable number of arguments has the behavior of an array.</p> <p>27. Write a static method, that can add <i>any number of integers</i> passed to it and returns a <b>long</b> as the result. Call this method from <b>main</b>, testing it with the following sets of data:</p> <ol style="list-style-type: none"><li>1. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10</li><li>2. 55, 76, 88, 13</li><li>3. 3, 7</li><li>4. An integer array containing the values 1, 5, 15, 19</li></ol> <pre>public class PracticeYourJava {     static long addIntegers(int... intArray) {         if (intArray.length == 0) return (0);         long total = 0;         for (int i = 0; i &lt; intArray.length; i++)             total += intArray[i];         return (total);     }      public static void main(String[] args) {         long result1 = 0, result2 = 0, result3 = 0, result4 = 0;         result1 = addIntegers(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);         result2 = addIntegers(55, 76, 88, 13);         result3 = addIntegers(3, 7);          int[] intArray01 = { 1, 5, 15, 19 };         result4 = addIntegers(intArray01);     } }</pre>
-----	--

**Note:** Until we get to the chapter on Classes, any method that we create will be a **static** method just as **main** is, that is, methods that are not tied to a particular object but rather to the class (non-static methods will be dealt with later when we deal directly with classes).

Due to the interrelationship between methods and classes, the concept of methods is further enhanced and refined in Chapter 22 which covers Classes.

# Chapter 2. Console Handling I: Basic Output

A significant part of many programs is the ability to present output to the user, via the interface with which the user is interacting with the program. This can be textual output to the console. In this chapter, exercises are presented with which the reader can practice and sharpen their skills in basic and formatted console output, using in particular the methods `System.out.printf`, `System.out.println` and `System.out.print`. Also, the basic use of string, integer and floating point variables appears in this chapter as the presented exercises pertain to outputting data of such types.

Output using the <code>System.out.printf</code> method		
1.	Predict the output of this program. Also explain the code. <pre>public class FormattingExercises{     public static void main(String[] args){         System.out.println("Hello!");     } }</pre>	<b>Output:</b> Hello! <b>Code Explanation:</b> We used the <code>System.out.println</code> method to output the string “Hello” to the console. All we had to do was to pass the string, enclosed in double quotes to the <code>println</code> method.
2.	Predict the output of this program. Also explain the code. <pre>public class FormattingExercises{     public static void main(String[] args){         String var01 = "Hello!";         System.out.printf("%s", var01);     } }</pre>	<b>Output:</b> Hello! <b>Code Explanation:</b> We used the <code>System.out.printf</code> method to output the contents of a variable named <code>var01</code> of type <code>String</code> to the console. Looking at the parameters passed to the <code>printf</code> method, we see that the first parameter is the double-quote bounded string <code>%s</code> and the second parameter is the name of a variable, in this case <code>var01</code> . The relationship between the 1 <sup>st</sup> and subsequent parameters is as follows: The first parameter in a call to <code>printf</code> can contain any text that we want to be output, mixed in with “formatting strings”, such as <code>%s</code> , which stand in for the values of <code>String</code> variable names that we pass along to <code>printf</code> as subsequent parameters to the 1 <sup>st</sup> . In this exercise, we have one formatting string presented in the first parameter and we also pass the name of the variable that it stands in for to <code>printf</code> . The number of formatting strings in the 1 <sup>st</sup> parameter must always be matched by an equal number of variable names passed in as parameters 2...n.
3.	Predict the output of this program. Also explain the code. <pre>public class FormattingExercises{     public static void main(String[] args){         String var01 = "Hello!";         String var02 = "How are you?";         System.out.printf("My friend! %s %s", var01, var02);     } }</pre>	<b>Output:</b> My friend! Hello! How are you? <b>Code Explanation:</b> Looking at the parameters passed to the <code>printf</code> method, we see that the first parameter is <code>“My friend! %s %s”</code> and the second and third parameters are the names of variables, in this case <code>var01</code> and <code>var02</code> . As stated in the solution to the preceding exercise, the first parameter passed to <code>printf</code> can contain any text that we want to be output and if desired we can mix in with that text “formatting strings”, such as <code>%s</code> , which stand in for the values of variable names that we pass along to <code>printf</code> as subsequent parameters to the 1 <sup>st</sup> parameter. In the 1 <sup>st</sup> parameter we have the text <code>“My friend!”</code> and within the same parameter we have two formatting strings, <code>%s</code> , a space and again <code>%s</code> , which indicates to us that our output string will be the string literal

## Practice Your Java Level 1

	<p>“My friend!” followed by whatever variable is substituted for the first %s and then a space (as seen in parameter 1) and then the value of whatever variable is to be substituted for the second %s.</p> <p>It should be explicitly stated that the order of replacement of formatting strings in parameter 1 is the order in which parameters 2...n appear, i.e. the first formatting string is replaced by the value of parameter 2, the 2<sup>nd</sup> formatting string is replaced by parameter 3 the 3<sup>rd</sup> formatting string is replaced by parameter 4 and so on.</p> <p>The resultant output is as noted above.</p>	
4.	<p>Predict the output of this program. Also explain the code.</p> <pre>public class FormattingExercises{     public static void main(String[] args){         String var01 = "Hello!";         System.out.printf("%s %s", var01, var01);     } }</pre> <p><b>Output:</b> Hello! Hello!</p> <p><b>Code Explanation:</b> The explanation of this code is essentially the same as that for the preceding exercise; only in this case we have chosen to have the same variable repeated for each of the two string formatting elements %s and %s in our 1<sup>st</sup> parameter.</p>	
5.	<p>Predict the output of this program. Also explain the code.</p> <pre>public class FormattingExercises{     public static void main(String[] args){         String var01 = "Hello!";         String var02 = "How are you?";         System.out.printf("%s\n%s", var01, var02);     } }</pre> <p><b>Output:</b> Hello! How are you?</p> <p><b>Code Explanation:</b> We see that the output is on two lines. This is due to the effect of the formatting parameter \n which we put between the two formatting placeholders in the first parameter. \n essentially means “put a newline here”. This is the effect that we have seen in our output.</p>	
6.	<p>Predict the output of this program. Also explain the code.</p> <pre>public class FormattingExercises{     public static void main(String[] args){         String var01 = "Hello!";         String var02 = "How are you?";         System.out.printf("%s\t%s", var01, var02);     } }</pre> <p><b>Output:</b> Hello! How are you?</p> <p><b>Code Explanation:</b> We see that the output has a tab between the two strings that have been output. This is due to the effect of the formatting parameter \t which we put between the two formatting placeholders in the exercise. \t means “put a tab here”. This is the effect that we have seen in our output.</p>	
7.	<p>I have two hardcoded <code>String</code> variables defined as follows:</p> <pre>String firstName = "John"; String lastName = "Doe";</pre> <p>Write a program which will use a <code>printf</code> statement to output these two <code>String</code> variables to the console on the same line with a single space in-between them.</p>	<pre>public class FormattingExercises {     public static void main(String[] args) {         String firstName = "John";         String lastName = "Doe";         System.out.printf("%s %s", firstName, lastName);     } }</pre>
8.	<p>Predict the output of this program. Also explain the code.</p> <pre>public class FormattingExercises{</pre>	<p><b>Output:</b> 2</p> <p><b>Code Explanation:</b> We are introduced here to a new formatting parameter, %d. The parameter %d is used for</p>

	<pre>public static void main(String[] args){     int var01=2;     System.out.printf("%d", var01); }</pre>	integer values. In all other aspects, the description of the functioning of the <code>printf</code> method is the same as in the preceding exercises.
9.	Predict the output of this program. Also explain the code.  <pre>public class FormattingExercises{     public static void main(String[] args){         double var01=2.55;         System.out.printf("%f", var01);     } }</pre>	<b>Output:</b> 2.550000 <b>Code Explanation:</b> We see here the formatting parameter, <code>%f</code> . The parameter <code>%f</code> is used for floating point values. In all other aspects, the description of the functioning of the <code>printf</code> method is the same as in the preceding exercises. Observe that the output is written to 6 decimal places. This is the default output format of the <code>%f</code> formatting parameter. We see in a later chapter how to modify the precision of the output.
10.	Predict and explain the output of this program.  <pre>public class FormattingExercises{     public static void main(String[] args){         double var01=2.123456789;         System.out.printf("%f", var01);     } }</pre>	<b>Output:</b> 2.123457 <b>Explanation:</b> While we did indeed enter a number which has 9 decimal places, as previously stated, the <code>printf</code> method by default only prints values out to 6 decimal places, rounding the last digit if necessary. In a later chapter, we see how to modify the precision of the output.

Output using the <code>System.out.print</code> & <code>System.out.println</code> methods		
11.	Write programs which output to the console the text "Hello World" followed by a newline character using <code>System.out.print</code> and <code>System.out.println</code> respectively.  <pre>public class FormattingExercises {     public static void main(String[] args) {         System.out.println("Hello World");     } }</pre>	<pre>public class FormattingExercises {     public static void main(String[] args) {         System.out.print("Hello World\n");     } }</pre> <b>Note:</b> You see that a <code>println</code> automatically adds a line terminator to the output. If you choose to use <code>print</code> , then you yourself have to add the newline character '\n' to the data being output in order to get the same output as <code>println</code> .
12.	Write a program which uses <code>System.out.println</code> to output the text "Hello World", inclusive of the double quotes, to the console.  <i>Hint: use the \ escape sequence.</i>	<pre>public class FormattingExercises {     public static void main(String[] args) {         System.out.println("\"Hello World\"");     } }</pre> Note: See the <i>escaping</i> of each double quote using the \ in each instance.
13.	Write a program which outputs the exact phrase "Hello World\n" inclusive of the "\n".	<pre>public class FormattingExercises {     public static void main(String[] args) {         System.out.println("Hello World\\n");     } }</pre>

Outputting from a variable using <code>System.out.println</code>		
14.	I have two hardcoded <code>String</code> variables defined as follows:  <pre>String firstName = "John"; String lastName = "Doe";</pre> Write a program which will use a <code>println</code> statement to output these two <code>String</code> variables to the console on the same line with a single space	<pre>public class FormattingExercises {     public static void main(String[] args) {         String firstName = "John";         String lastName = "Doe";         System.out.println(firstName + " " + lastName);     } }</pre>

## Practice Your Java Level 1

	in-between them.	
15.	I have two <b>String</b> variables defined as follows: <pre>String firstName = "John"; String lastName  = "Doe";</pre> Concatenate these strings, with a space in-between them, putting the resulting concatenation into a single <b>String</b> variable named <b>fullName</b> and then output the concatenated string to the console.	<pre>public class FormattingExercises {     public static void main(String[] args) {         String firstName = "John";         String lastName = "Doe";         String fullName = firstName + " " + lastName;         System.out.println(fullName);     } }</pre>
E1	The reader should study the class <b>DecimalFormat</b> to see how numerical formatting rules can be applied to the methods <b>System.out.println</b> and <b>System.out.println</b> .	

# Chapter 3. Number Handling I – Numerical Primitives

This chapter offers extensive coverage of the topic of number handling in Java. Your knowledge of numerical types in Java and operations thereon is exercised and enhanced by the wide-ranging set of exercises that are presented in this chapter.

Key topics on which exercises are presented in this chapter include: knowledge of the primitive numerical types, appropriate selection of numerical types, computation exercises with built-in numerical operators as well as with the methods of the class `Math`.

Beyond the topics noted above, exercises on the usage of numerical constants from the numerical wrapper classes, overflow conditions, the `Exact` methods, infinity handling, hexadecimal numbers, number base conversion, logical and bitwise operations, increment, decrement and shorthand numerical operators are also included. A robust set of exercises on the formatting of numerical output is also presented.

The topic of number handling in Java is extensive and thus is not limited to the topics in this chapter; this chapter is the first of five chapters in this book on the topic of number handling in Java.

	Explain the concept of <u>numerical primitives</u> in Java.	
1.	One way to describe a numerical primitive in Java is that a numerical primitive is a number type that is natively represented and handled in the computer CPUs that run Java. Most CPUs in the world today can represent and process the types <code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> and <code>double</code> natively. Also, numerical operations such as <code>+</code> , <code>-</code> , <code>/</code> , <code>*</code> on primitive types are native operations built into CPUs.	
2.	List in ascending order of size, the different <u>integer primitives</u> available in Java.	In order of size: <code>byte</code> , <code>short</code> , <code>int</code> and <code>long</code> .
	Explain the concept of <u>numerical wrapper classes</u> in Java.	
3.	In the first exercise, we explained that numerical primitives and some operations that can be performed on them are directly represented in most computer CPUs. However, not every operation that we would like performed is so represented. To address this, a corresponding class for each primitive exists in Java. These classes serve as an appropriate collection point for all methods and matters regarding the respective primitive that they correspond to. For example, the primitive <code>int</code> has a corresponding wrapper class named <code>Integer</code> , the primitive <code>short</code> has a wrapper class called <code>Short</code> , etc. Beyond serving as a collection point, there is also certain functionality in Java which works only on objects (for example <i>Collections</i> functionality, which we will see exercises on later); objects of these wrapper classes stand in for their equivalent primitive values in such cases. In this chapter we will see the use of the wrapper classes to a degree; they and their interaction with primitives is covered more directly in the chapter entitled <i>Number Handling II – Numeric Wrapper Classes</i> .	
4.	List the names of the corresponding wrapper classes respectively for each of the integer primitives <code>byte</code> , <code>short</code> , <code>int</code> and <code>long</code> .	<code>Byte</code> , <code>Short</code> , <code>Integer</code> , <code>Long</code>
<b>int</b>		
5.	What is the minimum value supported by the Java <code>int</code> type?	-2,147,483,648
6.	What is the maximum value supported by the Java <code>int</code> type?	2,147,483,647
7.	What is the default value assigned to an <code>int</code> variable that you have not yet assigned a value to ( <i>called an uninitialized variable</i> )?	There is no value assigned by the compiler. You yourself have to assign a value to a primitive variable before you can use it.
8.	How many bits/bytes are used to represent an <code>int</code> in Java?	32 bits/4 bytes
9.	Write a line of code to show the declaration and initialization	<code>int firstInteger = 5;</code>

## Practice Your Java Level 1

	of an <code>int</code> variable named <code>firstInteger</code> with the value 5.	
10.	<p>Write a program, using the built-in constants in the class <code>Integer</code>, which writes out the minimum and maximum values supported by the <code>int</code> type; also print out the size in bits that an <code>int</code> occupies.</p> <p><i>Hint: constants <code>Integer.MIN_VALUE</code>, <code>Integer.MAX_VALUE</code>, <code>Integer.SIZE</code></i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Integer.MIN_VALUE);         System.out.println(Integer.MAX_VALUE);         System.out.println(Integer.SIZE);     } }</pre> <p><b>Note:</b> As can be observed in this solution, the constants that refer to the primitive <code>int</code> are stored in its related wrapper class <code>Integer</code>. Recall that in the solution to exercise 3 it was stated that the corresponding wrapper class for each primitive type contains methods and other data (constants for example) that pertain to it and its corresponding primitive type.</p> <p>It should be pointed out that the range of values supported by a primitive and its corresponding class are the same. However, the size of an <code>int</code> (which is represented by the constant <code>Integer.SIZE</code>) is not the size of an <code>Integer</code> object, that constant only refers to the size of an <code>int</code>.</p> <p>The same logic above applies to all of the primitive types and their respective related classes.</p>	

<b>byte</b>		
11.	Describe the <code>byte</code> type. In particular describe the number of bits/bytes required to contain a <code>byte</code> , as well as the range of values that an <code>byte</code> can represent in Java.	A <code>byte</code> in Java is an integer type which occupies a single byte (8 bits) and thus can represent a range of $2^8 = 256$ numbers; the range of values that it represents is -128 to 127.
12.	Write a program which declares and initializes a byte variable named <code>firstByte</code> to the value -115 and then prints the variable to the console, stating “The value of <code>firstByte</code> is < <code>value</code> >”.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         byte firstByte = -115;         System.out.println("The value of firstByte is " + firstByte);     } }</pre>
13.	What is the minimum value supported by the <code>byte</code> type?	-128
14.	What is the maximum value supported by the <code>byte</code> type?	127
15.	Write a program, using the appropriate built-in constants in the class <code>Byte</code> , which writes out the minimum and maximum values supported by the <code>byte</code> type; also print out the size in bits that a <code>byte</code> occupies.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Byte.MIN_VALUE);         System.out.println(Byte.MAX_VALUE);         System.out.println(Byte.SIZE);     } }</pre>

<b>short</b>		
16.	What is the minimum value supported by the short integer ( <code>short</code> ) type?	-32,768
17.	What is the maximum value supported by the <code>short</code> type?	32,767
18.	How many bits/bytes are used to represent a <code>short</code> ?	16 bits/2 bytes
19.	Write a program which declares and initializes a <code>short</code> variable named <code>firstShort</code> to the value 10,000 and then prints the variable to the console, stating “The value of <code>firstShort</code> is < <code>value</code> >”.	<pre>public class PracticeYourJava {     public static void main(String[] args) {</pre>

	<pre> short firstShort = 10000; System.out.println("The value of firstShort is " + firstShort); } } </pre>
20.	<p>Write a program, using the built-in constants in the class <code>Short</code>, which writes out the minimum and maximum values supported by the <code>short</code> type; also, print out the size in bits that a <code>short</code> occupies.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Short.MIN_VALUE);         System.out.println(Short.MAX_VALUE);         System.out.println(Short.SIZE);     } } </pre>
<b>long</b>	
21.	<p>What is the minimum value supported by the Java long integer (<code>long</code>) type?</p> <p>-9,223,372,036,854,775,808 (This is the value <math>-2^{63}</math>).</p>
22.	<p>What is the maximum value supported by the Java <code>long</code> type?</p> <p>9,223,372,036,854,775,807 (This is <math>2^{63} - 1</math>)</p>
23.	<p>How many bits/bytes are used to represent a <code>long</code>?</p> <p>32 bits/4 bytes</p>
24.	<p>Write a program which initializes a <code>long</code> variable named <code>firstLong</code> with the value 7,000,000,000 and then prints the variable to the console, stating “The value of <code>firstLong</code> is &lt;<i>value</i>&gt;”.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         long firstLong = 7000000000L;         System.out.println("The value of firstLong is " + firstLong);     } } </pre>
25.	<p>Explain the reason for the postfix <code>L</code> at the end of the numerical value in the preceding exercise.</p> <p>The reason for the postfix is because by default Java sees integer literals as values of type <code>int</code>; if the integer literal presented is greater than <code>Integer.MAX_VALUE</code> then Java sees it as an invalid value because in a sense it sees it as an integer that is out of bounds. However when it sees the postfix, Java evaluates the value as the type that the postfix implies.</p>
26.	<p>What is the difference between the following declarations of the variable <code>firstLong</code>?</p> <pre> long firstLong = 7000000000L; and long firstLong = 7000_000_000L; and long firstLong = 7_000_000_000L; and long firstLong = 7_000_000__000L; </pre> <p>There is actually no difference between any of the declaration styles. Java allows you to put the underscore character in number literals as adds clarity for human viewers of the numerical value. The number of underscores does not matter either (see the 4<sup>th</sup> example). However, the underscore character cannot be put at the beginning or the end of the number.</p>
27.	<p>Write a program, using the built-in constants in the class <code>Long</code>, which writes out the minimum and maximum values supported by the <code>long</code> type; also print out the size in bits that a <code>long</code> occupies.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Long.MIN_VALUE);         System.out.println(Long.MAX_VALUE);         System.out.println(Long.SIZE);     } } </pre>

## Practice Your Java Level 1

floating point types ( <code>float</code> , <code>double</code> )		
28.	What is the minimum value supported by the <code>float</code> type?	$-3.4028235 \times 10^{-38}$
29.	What is the maximum value supported by the <code>float</code> type?	$-3.4028235 \times 10^{38}$
30.	How many bits/bytes are used to represent a <code>float</code> ?	32 bits/4 bytes
31.	What is the precision of a <code>float</code> ?	<p>6-7 significant digits (that includes the digits both before and behind the decimal point).</p> <p>You know that integer values are discrete values with a deterministic integer distance of 1 between each point on the integer number scale. With floating points numbers it is not so; there are an infinite number of floating point numbers possible, even between two adjacent integers (for example between 0 and 1). Therefore it is impossible to write down all floating point numbers even within a small range, nor is it possible for a computer to represent the infinite number of points of even a small range on the floating point number scale. Therefore it has been decided that the <code>float</code> type will have a given maximum precision that it can represent. Any attempt to use the <code>float</code> type to represent a number with a precision higher than 6-7 digits will be unsuccessful.</p> <p>Even within the specified range of 6-7 digits, there are some numbers which cannot be represented exactly due to the way in which floating point numbers are stored in computers. Any decimal number that requires maintenance of precision (such as currency calculations) should only use the type <code>BigDecimal</code>.</p> <p>The reader is urged to investigate appropriate sources which describe how floating point numbers are represented in computers.</p>
32.	What is the smallest possible positive value that a <code>float</code> can represent?	$\sim 1.4 \times 10^{-45}$ Think of the number as the smallest step away from <code>0</code> that the Java <code>float</code> type can make.
33.	Why is a <code>float</code> also described as a <i>single</i> ?	<code>float</code> really refers to what is called a “ <u>single precision</u> floating point number”, which is a floating point number format that is designed to occupy only 4 bytes. This nomenclature is an IEEE standard.
34.	Write a program, using the built-in constants in the class <code>Float</code> , which writes out the minimum and maximum values supported by the <code>float</code> type. Also print out the size in bits that a <code>float</code> occupies.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Minimum value = %f\n" + (-1 * Float.MAX_VALUE));         //there is no constant for minimum value         System.out.println("Maximum value = " + Float.MAX_VALUE);         System.out.println("Bits occupied = " + Float.SIZE);     } }</pre>
35.	Write a program which prints out the smallest positive <code>float</code> that Java supports.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.printf("Smallest positive float = %f", Float.MIN_VALUE);     } }</pre>
36.	Why does java require that there a suffix/postfix <code>F</code> applied to the value assigned to a variable of type <code>float</code> ?	The reason is because Java presumes that any floating point literal entered is of type <code>double</code> ( <i>double means “double precision floating point number”, a type which we will see shortly</i> ) unless the number is explicitly indicated to be a single precision number as indicated by the postfix <code>F</code> .
37.	Write a program which initializes a single precision floating point variable named <code>firstFloat</code> with the value 7.919876542 and prints its value out to the console using <code>System.out.println</code> . Note and explain the difference between the initialized value and the value that is printed out.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float firstFloat = 7.919876542F; // Note the F at the end         System.out.println(firstFloat);     } }</pre>

	<pre>}</pre> <p><b>Observation/Explanation:</b> The output is 7.9198766 not the 10 digit number 7.919876542 that was input! The reason is that the <code>float</code> type officially only has a precision of 6-7 significant digits and so we should not have attempted to assign it a value of higher precision. Yes indeed there are 8 significant digits in the output, however the last digit was the result of rounding. There are times when the value of the rounded up digit is acceptable and times when it is not.</p> <p>Also note that the value that was input was rounded (not truncated). There are different rounding modes available in Java; exercises on these are presented in Chapter 12, entitled <i>Number Handling III</i>.</p>
38.	<p>Write a program which will assign the value 1768000111.77 to a variable of type <code>float</code>. Print the value out using the <code>%f</code> specifier of the <code>printf</code> method. Note the output and present your observations on it.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1 = 1768000111.77F; // Note the F at the end         System.out.printf("f1 = %f\n", f1);     } }</pre> <p><b>Output:</b> <code>f1 = 1768000128.00000</code></p> <p><b>Observations:</b> The output is quite different from the input! In fact, even if the input variable was manually rounded to the nearest decimal position, the value obtained would be closer to the original number than what the computer has determined the float to be!</p> <p>As in the preceding exercise, the error was that the number entered was beyond the stated precision of a float (which is 6-7 significant digits). Java did indeed retain the integrity of at least the first 6-7 digits.</p> <p><b>Conclusion(s):</b> Do not attempt to apply to a floating point variable a value beyond its stated precision.</p>
<b>double</b>	
39.	What is the minimum value supported by the type <code>double</code> ?
40.	What is the maximum value supported by the type <code>double</code> ?
41.	How many bits/bytes are used to represent a <code>double</code> ?
42.	What is the smallest possible positive value that a <code>double</code> can represent?
43.	What is the precision of a <code>double</code> ?
44.	Write a line of code that shows the declaration and initialization of a double precision floating point variable named <code>firstDouble</code> with the value $5.4857990943 \times 10^{-4}$ .
45.	Write a program, using the built-in constants in the class <code>Double</code> , which writes out the minimum and maximum values supported by the <code>double</code> type. Also print out the size in bits that a <code>double</code> occupies.
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Minimum value = " + (-1 * Double.MAX_VALUE));                     //there is no direct constant for this         System.out.println("Maximum value = " + Double.MAX_VALUE);         System.out.println("Bits occupied = " + Double.SIZE);     } }</pre>
<b>Assessing the Precision of type <code>double</code></b>	
46.	<p>Write a program which will assign the 12 digit number 1768000111.77 to a variable of type <code>double</code> and print it out using the <code>%f</code> specifier of the <code>printf</code> method and also using the method <code>println</code>. Note the output and present your observations on it.</p> <p>(This input value is the same attempted with a <code>float</code> in question 38).</p> <pre>public class PracticeYourJava {</pre>

## Practice Your Java Level 1

	<pre>public static void main(String[] args) {     double d1 = 1768000111.77;     System.out.printf("d1 = %f\n", d1);     System.out.println(d1); }</pre> <p><b>Output:</b> d1 = 1768000111.770000 d1 = 1.76800011177E9</p> <p><b>Observations:</b> The output matches the input, for the precision of the input value is 12 significant digits, which is within the supported range of a <code>double</code>. We observe that <code>println</code> output the number in exponential format.</p>
47.	<p>Write a program which will assign the 16 digit value 17680001118938.77 to a variable of type <code>double</code> and print it out using the <code>%f</code> specifier of the <code>printf</code> method. Note the output and present your observations on it.</p> <p>Solution: modify the code in the preceding exercise to use the number in this exercise.</p> <p><b>Output:</b> d1 = 17680001118938.770000</p> <p><b>Observations:</b> The output matches the input exactly, for the precision of the input value has 15-17 significant digits which is within the supported range of a <code>double</code>.</p>
48.	<p>Write a program which will assign the 17 precision place value 176800011189387.35 to a variable of type <code>double</code> and print it out using the <code>%f</code> specifier of the <code>printf</code> method. Note the output and present your observations on it.</p> <p>Solution: modify the code in the preceding exercise to use the number in this exercise.</p> <p><b>Output:</b> d1 = 176800011189387.340000</p> <p><b>Observations:</b> As can be seen, the last digit of the entered number is distorted. This should be expected as the <code>double</code> supports only at best a 15-17 precision place number.</p> <p><b>Summary:</b> You should carefully consider whether or not you should use the type <code>double</code> for values with precision greater than 15 places.</p>
49.	<p>Write a program which will assign the 18 precision place value 123456789012345678.0 to a variable of type <code>double</code> and print it out using the <code>%f</code> specifier of the <code>printf</code> method. Note the output and present your observations on it.</p> <p>Solution: modify the code in the preceding exercise to use the number in this exercise.</p> <p><b>Output:</b> d1 = 123456789012345680.000000</p> <p><b>Observations:</b> As can be seen, digit 18 is rounded up and that affects the value of digit 17. This should be expected as the type <code>double</code> supports at best only a 15-17 precision place number.</p>
50.	<p><b>IMPORTANT</b></p> <p>Print out the value <code>Float.MIN_VALUE</code>, using <code>printf</code> and then <code>println</code>. Comment on your observations.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double d1 = Double.MIN_VALUE;         System.out.printf("d1 = %f\n", d1);         System.out.println("d1 = " + d1);     } }</pre> <p><b>Output</b></p> <p>d1 = 0.000000 d1 = 4.9E-324</p> <p>Clearly the 1<sup>st</sup> output, using <code>printf</code> is wrong! This occurrence is because of the <i>default</i> way in which <code>printf</code> formats floating point numbers; it will by default print at most 6 digits after the decimal point and will not by default attempt to print the value out in exponential format as <code>println</code> does.</p> <p>Later in this chapter, exercises on formatting of numerical output are presented.</p> <p>For further experimentation along this line, print out the <code>double</code> value -1.0011596521859 using <code>printf</code> and <code>println</code>.</p>
51.	<p><b>IMPORTANT</b></p> <p>Print out the value to at least 15 places of the result of the computation 1/10. Comment on the output.</p> <pre>public class PracticeYourJava {</pre>

```

public static void main(String[] args) {
    double x = 1.0f/10.0f;
    System.out.printf("%.15f\n", x);
}

```

**Output**

`z = 0.100000001490116`

The value is not exactly `0.1!` The computation `1/10` is supposed to yield a very specific rational result. This error occurs in some floating point calculations because of the way that computers handle/store floating point numbers. Clearly, this is not a good thing when we need exact precision; imagine scenarios where we have a financial application which computes values ranging into the billions of currency units; this error will definitely show up in such sensitive computations. Another example where precision is required is in scientific calculations; this imprecision in handling would also show up then. We see in the next section and in the chapter *Number Handling III* how to achieve the desired precision for floating point numbers by using the type `BigDecimal`.

**The numerical types `BigInteger` & `BigDecimal`: A brief note**

**BigInteger:** What do you use to represent an integer value whose value is larger than the type `long` or even longer than an unsigned long (we see the unsigned long type in a later chapter)? The type `BigInteger`. This type can accept and operate on arbitrarily large integer values.

**BigDecimal:** In our exercises with floating point values, you've observed that the exact value of floating point numbers is not always correct, this due to the way that floating numbers are represented in computers. However, Java presents a different type, the type `BigDecimal`, which represents floating point numbers with fidelity. Numbers represented using the type `BigDecimal` are not represented in the same internal format that `float` and `double` are. Due to its fidelity, for any financial computation that involves decimal values, or for any decimal value whose fidelity we utterly must preserve, the only acceptable choice in Java for such numbers is the type `BigDecimal`.

We have presented these two types here briefly as we refer to their use in a few upcoming exercises. They are covered directly in the chapter entitled *Number Handling III – Big Integer, Big Decimal*.

52. I have the integer value 424,312,343,211,431,231,234,312,431,234,122,343,123,131,312,313,123,436. What type is best to represent it in Java?

The type `BigInteger`. This type handles arbitrarily long integer values. We see this type in a later chapter.

**Numerical suffixes**

53. What are the suffixes to apply to numerical values to indicate what type they are? In particular list the suffixes for each of, `long`, `float` and `double`.

<code>long</code>	L
<code>float</code> (also known as single)	F
<code>double</code>	D

**Note:** These suffixes are case insensitive.

**Decide which number type best represents the kind of number indicated in the question.**

54.	Which type would be best to represent the salary of an individual?	The type <code>BigDecimal</code> . The reasons is because monetary values are always best represented by variables of type <code>BigDecimal</code> as <code>BigDecimal</code> represents decimal numbers accurately.
55.	Which type would be best to represent the aggregate salary over 20 years of the salaries of a company which has 50,000 employees?	The type <code>BigDecimal</code> would be best because we are representing a monetary value which requires precise representation.
56.	Which type is sufficient to handle the age of a person?	The type <code>byte</code> is sufficient and safe (unless you are interested in fractions of years as well). The type <code>byte</code> has

## Practice Your Java Level 1

		a maximum of 127 which is about as high as people live these days.
57.	The population of the world is currently $\sim$ 7.2 billion people. Which type should be used to represent this?	The type <code>long</code> is an appropriate type in this case. The reasons for this are: 1. Human beings are integral units. 2. This is the closest type that can contain at least that number of people.
58.	To count the number of siblings you have, which type would be the most likely candidate?	A <code>byte</code> will do. The reasons for this are: 1. Human beings are integral units. 2. Chances are no one has up to 127 siblings!
59.	Which type is best to represent the Planck constant which has a value of $6.626\ 068\ 9633 \times 10^{-34}$ ?	The type <code>BigDecimal</code> because although this number has only 11 significant digits which can be represented in a <code>double</code> we would be safe with the precision of the <code>BigDecimal</code> type when using this constant in calculations.
60.	The speed of light is precisely defined as 299 792 458 m/s. Which type would be sufficient to represent it?	An <code>int</code> , because this number is an integer of value approximately $\sim$ 300 million which is less than the $\sim$ 2.1 billion that an <code>int</code> supports.
61.	The Rydberg constant has a value of $1.0973731568539 \times 10^7$ . Propose which type is best to represent it.	The type <code>BigDecimal</code> because although this number has only 14 significant digits, we would be safe with the precision of the <code>BigDecimal</code> type when using this constant.
62.	The electron magnetic moment to Bohr magneton ratio has a value of -1.001 159 652 1859. Which type best represents it?	The type <code>BigDecimal</code> .
63.	The electron g factor is -2.002 319 304 3718. Which type best represents it?	The type <code>BigDecimal</code> .
64.	Electron g factor uncertainty is 0.000 000 000 0075. Which type best represents it?	The type <code>BigDecimal</code> .
65.	The value of Coulomb's constant is $8.9875517873681764 \times 10^9$ . Which type best represents it?	A <code>BigDecimal</code> . Not only for its accuracy, but also because this number has 17 significant digits.

A general note on the choice of `BigDecimal` for exercises 59 to 64: Really, a `double` may suffice in these cases for general computation scenarios (perhaps, for example, in writing examinations).

### Casting

66.	Explain the concept of <i>casting</i> .
	<p>Casting is a language feature wherewith you can force the compiler to take <u>a copy</u> of the contents of a variable and convert the copy to the format of another type.</p> <p>For example, I can do the following:</p> <pre>int a = 50; short b = (short)a;</pre> <p>The second line of which means, <i>give me a copy of a that has been converted into the internal format and space of a short and put it into the short variable b (note that a itself is not changed)</i>.</p> <p>We elucidate further using <code>short</code> and <code>int</code>; a <code>short</code> is represented in 2 bytes and an <code>int</code> in 4 bytes. It is reasonable to say that any <code>int</code> value which is less than or equal to the width of a <code>short</code> (i.e. an <code>int</code> whose actual value is between -32,768 and 32,767) can legally fit into a <code>short</code>. However, we have to explicitly inform the compiler that we do indeed want to copy the contents of the <code>int</code> in question into a <code>short</code>.</p> <p>Note that you can actually still cast the value of a higher width type whose contents are greater than the width of a lower width type into the lower width type; for example you can cast an <code>int</code> whose value is greater than 32,767 into a <code>short</code>; while indeed the value in question cannot actually fit into the space of a <code>short</code>, the compiler will still do its best to cast the value, however the value will be distorted, i.e. wrong, since you cannot fit something of a higher actual width into a smaller space properly.</p>

	<p>Casting also pertains to objects; you can cast an object of a given class into a variable of any of its ancestor classes. This ability is facilitated by the concept of polymorphism. We see examples of this in the chapter on classes.</p> <p>The concepts of <i>boxing</i> and <i>unboxing</i> can be seen as a special type of casting. We address the topic of boxing and unboxing in a later chapter.</p>
67.	<p>Why would you want to cast variables?</p> <p>You would want to cast variables in scenarios where you want the contents of a casted variable to be put into a variable of a different type.</p> <p>For example, if I have an <code>int a=10</code> and given how small its value is, I want to put it into a variable of type <code>byte</code> (which supports values from 0 to 255), I would cast it as follows:</p> <pre>byte c=(byte)a;</pre> <p>This line is saying <i>take a copy of the contents of int a and stuff them into the format of a byte as best as you can.</i></p> <p>If for example I have to add an <code>int</code> and a <code>long</code> and want the resultant value to be put into an <code>int</code>, due to the fact that the target variable has a smaller width than at least one of the input variables, the input variables that are larger than the output variable have to be cast to the width of the target variable. See the following example:</p> <pre>long a=10; int b=50; int result = (int)a + b;</pre> <p>Some casts are more dramatic. For example, the <code>char</code> type (which represents characters) can be cast into an integer representation. For example:</p> <pre>char letter = 'A'; int integerEquivalent = (int)letter;</pre> <p>With this cast, you are now able to facilitate the sorting of characters because they have an explicit numerical representation, which lends itself to ordering by magnitude.</p> <p>You can also cast objects. For example, if I have the following object of a class named <code>Road</code>:</p> <pre>Road x = new Road();</pre> <p>I can, if I desire, cast the object <code>x</code> to an object of the ultimate ancestor class <code>Object</code> as follows:</p> <pre>Object y = (Object)x;</pre> <p>(The casting of an object to any of its ancestor classes is made feasible by the concept of polymorphism)</p> <p>I can cast <code>y</code> back to an object of class <code>Road</code> as follows:</p> <pre>Road z = (Road)y;</pre> <p>As previously stated, for numerical values you <i>will</i> lose data in casting if you try to cast a variable which contains a higher value than what the type of the target variable can support. For example if we have the following scenario:</p> <pre>int var01 = 75000; short var02 = (short)var01;</pre> <p>Because the value of the variable <code>var01</code> exceeds the maximum for a <code>short</code>, <code>var02</code> definitely will not contain the value 75,000, but rather a distorted value.</p>
68.	<p>What is <u>implicit casting</u>?</p> <p>Implicit casting is a scenario where, given the operation at hand, the compiler sees it fit to silently cast the contents of a variable of a given type to a more appropriate type that it is safe to cast the given type to.</p> <p>For example, if I wanted to add an <code>int</code> to a <code>long</code> and put the result into a <code>long</code>, the code would look like the following:</p> <pre>long a=10; int b = 50; long result = a + b;</pre> <p>In order to perform this computation, the compiler has to make the contents of both <code>a</code> and <code>b</code> the same type; in this case the compiler will simply silently cast <code>b</code> to the wider type <code>long</code> (note that it doesn't change <code>b</code> itself it just</p>

## Practice Your Java Level 1

	<p>takes a <i>casted</i> value of <b>b</b>) and adds it to <b>a</b> to obtain <b>result</b>. There will not be any ill-effects since an <b>int</b> is of smaller range than the target output variable (a <b>long</b> in this case).</p> <p>Now, if I had the following scenario where <b>result</b> is an <b>int</b>:</p> <pre>long a=10; int b=50; int result = a + b;</pre> <p>The compiler will not implicitly cast the variable <b>a</b> to an <b>int</b>. Why not? Because casting a given type (<b>long</b> in this case) to a smaller type can potentially result in the loss of data. Therefore, in this case, the compiler will insist that you do the casting of the variable <b>a</b> yourself if that is what you really want, requiring you to explicitly write the following:</p> <pre>int result = (int)a + b;</pre>
69.	<p>Explain the result of compiling and running the program below.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         short a = 5;         short b = 2;         short result;          result = a + b;         System.out.println("result = " + result);     } }</pre> <p>It will <i>not</i> compile. The reason for the non-compilation is that in Java, computations done with respect to <b>short</b> or <b>int</b> values result in an <b>int</b>. Therefore Java holds the value of <b>a + b</b> as an <b>int</b>, even though both input values are of type <b>short</b>. Us trying to put the resulting <b>int</b> value into a <b>short</b> is illegal unless we explicitly cast the value. To get this to compile, you have to do either of the following:</p> <ul style="list-style-type: none"><li>(1) <b>result</b> should be declared as an <b>int</b> or</li><li>(2) <b>result = (short)(a+b);</b> that is, you have to cast the result of the computation to the desired type (in this case, a <b>short</b>).</li></ul>
70.	<p>Write a program to add the following two <b>short</b> values (<b>i=250; j=3</b>). Put the result into a <b>short</b> and print the result out to the console.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         short i = 250;         short j = 3;         short result;          result = (short)(i + j);         System.out.println("result = " + result);     } }</pre>
71.	<p>Write a program which multiplies the following two <b>short</b> values (<b>i=250; j=300</b>), puts the result into a <b>short</b> and prints the result out to the console.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         short i = 250;         short j = 300;         short result;          result =(short)(i * j);         System.out.println("result = " + result);     } }</pre> <p><b>Note:</b> This will print out a value of <b>550</b> instead of <b>75000!</b> The reason is because the resultant value is greater than <b>Short.Max</b> (which = <b>32767</b>) and thus the result of casting a 4 byte integer value of <b>75000</b> to a <b>short</b> (which has a</p>

	<p>width of 2 bytes) yields 550 which is not what we want at all. We have two options to avoid such behavior. These are:</p> <ol style="list-style-type: none"> <li>1. Always use the appropriately sized type for your calculations. <i>or</i></li> <li>2. Use the <i>Exact</i> methods of class <b>Math</b> to ensure that an exception is triggered when a computation goes out of bounds. We will look at the <i>Exact</i> methods later in this chapter.</li> </ol>
72.	<p>Write a program to multiply the following two <b>short</b> variables (<b>i=250; j=300</b>). Put the result into an <b>int</b> and print the result out to the console.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         short i = 250;         short j = 300;         int result;          result = i * j;         System.out.println("result = " + result);     } }</pre>
	<p><b>constants</b></p> <p>73. What is a constant (<b>final</b>) in Java?</p> <p>A constant is a variable with a hardcoded and immutable value. For example, in the class <b>Math</b>, there are the constants <b>PI</b> and <b>E</b>.</p> <p>You define constants using the keyword <b>final</b> (indicating that this is the “final value” of this variable). A variable labeled as being <b>final</b> can only be assigned to once! That assignment might be hardcoded in the code itself, or for constants in individual objects or in methods it can be at any time during the running of the code, only that it can only be done only once.</p> <p>Again, a constant can be declared at the class level, or at the instance level, or inside a method. The declaration for a class level constant has the following structure:</p> <pre>class &lt;class name&gt; {     &lt;access level&gt; static final &lt;field type&gt; &lt;field name&gt; = &lt;value&gt;; }</pre> <p>In the class <b>Math</b> for example, the constants <b>PI</b> and <b>E</b> are class level constants and are declared as:</p> <pre>class Math {     public static final double PI = 3.141592653589793; // See the use of the keyword final     public static final double E = 2.718281828459045; }</pre> <p>(The keyword <b>static</b> is what makes these constants <i>class constants</i>, thus making them directly accessible by any object that can access the fields of this class; for example you can simply access the field <b>PI</b> in any of your code directly as <b>Math.PI</b>. We will look closer at the keyword <b>static</b> the chapter entitled <i>Classes I</i>).</p> <p>The declaration for an instance/object level constant is as follows:</p> <pre>class &lt;class name&gt; {     &lt;access level&gt; final &lt;field type&gt; &lt;field name&gt; = &lt;value&gt;; // No keyword static }</pre> <p>Method constants are declared within methods in the following manner (using <b>main</b> as an example):</p> <pre>public static void main(String[] args){     final double ten      = 10.0;     final int DaysInWeek = 7;     final int MonthsInYear = 12;     final String MetricOrImperial; // We can assign to this later in our code, perhaps it is                                    // a field we get from the user. }</pre>

## Practice Your Java Level 1

	<b>Accessing constants</b> Class constants are accessed using the class name as the prefix to the constant name. Therefore, for example, you access the constant PI of the class Math as Math.PI in your code. Instance constants are accessed using the name of the object as the prefix to the constant name. Method constants are accessed using only their name (since a method constant is only accessed from within the method in which it is declared).
74.	What is a <u>blank final</u> ? A blank final is a final variable that is not assigned at compile time.
75.	Write a program to print out the value of the built-in mathematical constant π. <i>Hint: Math.PI</i>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Math.PI);     } }</pre>
76.	Write a program to print out the value of the built-in mathematical constant e.
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Math.E);     } }</pre>
77.	Write a line of code to show the implementation of the Rydberg constant as a <i>public class</i> constant.
	<pre>public static final double rydberg = 1.0973731568539e7;</pre>
78.	Write a line of code to show the implementation of the Rydberg constant as a <i>public instance</i> constant.
	<pre>public final double rydberg = 1.0973731568539e7;</pre>

### Built-in mathematical operators and methods

In this section, exercises on the usage in Java of standard mathematical operators, constants and methods of the class Math are presented.

79.	Write a program which adds the following two int variables; i=5; j=9; and prints the result out to the console.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int i = 5; int j = 9;         int result = i + j;         System.out.println("The sum is: " + result);     } }</pre>
80.	Given two int variables i=5; j=9; write a program which subtracts the latter from the former and prints the result out to the console.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int i = 5; int j = 9;         int result = i - j;         System.out.println("The sum is: " + result);     } }</pre>
81.	Write a program to calculate and print out the remainder of 55 divided by 22. <i>Hint: Use the % operator</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int rem = 55 % 22;         System.out.println("The remainder is: " + rem);     } }</pre>
82.	Write a program to calculate and print out the square root of 81.39. <i>Hint: Math.sqrt</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float x = 81.39f;         double result;          result = Math.sqrt(x);         System.out.println("The square root of " + x + " is: " + result);     } }</pre>
83.	Print out the value of 79.34 raised to the power of 12.	

	<p><i>Hint: Math.pow</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float x = 79.34f;         double result;          result = Math.pow(x, 12);         System.out.println(x + " to the power of " + 12 + " is: " + result);     } }</pre>
84.	<p>Print out the cube root of 89.</p> <p><i>Hint: Math.cbrt</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float x = 89.0f;         double result;          result = Math.cbrt(89);         System.out.println("The cube root of " + x + " is: " + result);     } }</pre>
85.	<p>Print out the fifth root of 89.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float x = 89.0f;         double result;          result = Math.pow(x,(1.0/5.0));         System.out.println("The 5th root of " + x + " is: " + result);     } }</pre>
86.	<p>Print out the 7<sup>th</sup> root of 121.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float x = 121.0f;         double result;          result = Math.pow(x,(1.0/7.0));         System.out.println("The 7th root of " + x + " is: " + result);     } }</pre>
87.	<p>Write a program to calculate and print out the cosine of 180°.</p> <p><i>Hint: Methods Math.Cos and Math.toRadians</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double degrees = 180;         double radians = Math.toRadians(degrees);         double cosine = Math.cos(radians);         System.out.printf("The cosine of %f degrees = %f\n",degrees,cosine);     } }</pre>
88.	<p>Write a program to calculate and print out the tangent of 30°.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double degrees = 180;         double radians = Math.toRadians(degrees);         double tangent = Math.tangent(radians);         System.out.printf("The tangent of %f degrees = %f\n",degrees,tangent);     } }</pre>

## Practice Your Java Level 1

89.	<p>Write a program to calculate and print out the tangent of <math>60^\circ</math>.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double degrees = 60;         double radians = Math.toRadians(degrees);         double tangent = Math.tangent(radians);         System.out.printf("The tangent of %f degrees = %f\n", degrees, tangent);     } }</pre>
90.	<p>Write a program to calculate and print out the sine of <math>75^\circ</math>.</p> <p><i>Hint: Math.toRadians, Math.sin</i></p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double degrees = 75;         double radians = Math.toRadians(degrees);         double sine = Math.sin(radians);         System.out.printf("The sine of %f degrees = %f\n", degrees, sine);     } }</pre>
91.	<p>The sine of a given angle is <math>.8660254037</math>. Write a program to calculate and print out in degrees <i>and</i> in radians the angle that this value corresponds to.</p> <p><i>Hint: Math.asin, Math.toDegrees</i></p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double sine = .8660254037;         double degrees, radians;          radians = Math.asin(sine);         degrees = Math.toDegrees(radians);         System.out.printf("%f is the sine of %f degrees (or %f radians).\n", sine, degrees, radians);     } }</pre>
92.	<p>The cosine of a given angle is <math>.8660254037</math>. Write a program to calculate and print out in degrees and in radians the angle that this value corresponds to.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double cosine = .8660254037;         double degrees, radians;          radians = Math.acos(cosine);         degrees = Math.toDegrees(radians);         System.out.printf("%f is the cosine of %f degrees (or %f radians).\n", cosine, degrees, radians);     } }</pre>
93.	<p>The tangent of a given angle is <math>.8660254037</math>. Write a program to calculate and print out in degrees and in radians the angle that this value corresponds to.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double tangent = .8660254037;         double degrees, radians;          radians = Math.atan(tangent);         degrees = Math.toDegrees(radians);         System.out.printf("%f is the tangent of %f degrees (or %f radians).\n", tangent, degrees, radians);     } }</pre>
94.	<p>Write a program to print out the area of a circle of radius 7.</p>
	<pre>public class PracticeYourJava {</pre>

	<pre>public static void main(String[] args) {     double radius = 7.0;     double area;      area = Math.PI * Math.pow(radius, 2);     System.out.printf("The area of a circle of radius %f = %f.\n", radius, area); }</pre>
95.	<p>Write a program to print out the absolute value of the following values: -743.8, 525.3 and 1501.</p> <p><i>Hint: Math.abs</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double value01 = -743.8; double abs01;         double value02 = 525.3; double abs02;         int value03     = 1501;   int abs03;          abs01 = Math.abs(value01);         abs02 = Math.abs(value02);         abs03 = Math.abs(value03);          System.out.printf("The absolute value of %f is %f\n", value01, abs01);         System.out.printf("The absolute value of %f is %f\n", value02, abs02);         System.out.printf("The absolute value of %d is %d\n", value03, abs03);     } }</pre>
96.	<p>Calculate the logarithm to base 10 of 7 and the logarithm to base 10 of 22. Add the resulting values, putting the sum thereof into a variable x. Then compute and print out the value of <math>10^x</math>.</p> <p><i>Hint: Math.Log10, Math.pow</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double d01 = 7;         double d02 = 22;          double log0fd01 = Math.Log10(d01);         double log0fd02 = Math.Log10(d02);          double x = log0fd01 + log0fd02;         double result = Math.pow(10, x);         System.out.printf("result = %f\n", result);     } }</pre>
97.	<p>Calculate the logarithm to base <math>e</math> of 7 and the logarithm to base <math>e</math> of 22. Add the resulting values, putting the sum thereof into a variable x. Then compute and print out the value of <math>e^x</math>.</p> <p><i>Hint: Math.Log, Math.exp</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double d01 = 7;         double d02 = 22;          double log0fd01 = Math.Log(d01);         double log0fd02 = Math.Log(d02);          double x = log0fd01 + log0fd02;         double result = Math.exp(x);         System.out.printf("result = %f\n", result);     } }</pre>
98.	<p>Pythagoras' theorem states that for a given right-angled triangle with sides <math>a</math>, <math>b</math> and <math>c</math> where <math>c</math> is the hypotenuse, if you have the values <math>a</math> and <math>b</math>, the value for <math>c</math> can be determined as follows:</p> $c = \sqrt{a^2 + b^2}.$ <p>So given a triangle where <math>a=4</math> and <math>b=3</math>, write a program to calculate <math>c</math> and print it out.</p> <p><i>Hint: Math.hypot</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float a = 3;         float b = 4;         double c; // this is for the hypotenuse          c = Math.hypot(a, b);         System.out.printf("hypotenuse = %f\n", c);     } }</pre>

## Practice Your Java Level 1

99.	<p>Given two numbers, 55 and 70, write a program to determine and print out which is the larger of the two.</p> <p><i>Hint: Math.max</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int bigger = Math.max(55, 70);         System.out.printf("The bigger number is %f\n", bigger);     } }</pre>
100.	<p>Given two numbers, 320 and -2.95 of type <code>double</code>, write a program to determine and print out which is the smaller of the two numbers.</p> <p><i>Hint: Math.min</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double smaller = Math.min(320, -2.95);         System.out.printf("The smaller number is %f\n", smaller);     } }</pre> <p><b>Note:</b> to do comparisons using <code>Math.min</code> or <code>Math.max</code> the numbers being compared must be of the same type.</p>
101.	<p>Round the value 12777.899 to the nearest whole number <code>double</code> and print the result out.</p> <p><i>Hint: Math.rint</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double x = 12777.899;         double rounded = Math.rint(x);         System.out.printf("The rounded value is %f\n", rounded);     } }</pre>
102.	<p>Round the value 12777.899 to the nearest <code>long</code> and print the result out.</p> <p><i>Hint: Math.round</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double x = 12777.899;         long rounded = Math.round(x);         System.out.printf("The rounded value is %d\n", rounded);     } }</pre>
103.	<p>Write a program to print out the <u>ceiling</u> of 5.799 and 583.6 respectively.</p> <p><i>Hint: Math.ceil</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double d01 = 5.799;         double d02 = 583.6;          double ceil01 = Math.ceil(d01);         double ceil02 = Math.ceil(d02);          System.out.printf("The ceiling of %f is %f\n", d01, ceil01);         System.out.printf("The ceiling of %f is %f\n", d02, ceil02);     } }</pre>
104.	<p>Write a program to print out the <u>floor</u> of 5.799 and 583.6 respectively.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double d01 = 5.799;         double d02 = 583.6;          double ceil01 = Math.floor(d01);         double ceil02 = Math.floor(d02);          System.out.printf("The floor of %f is %f\n", d01, ceil01);         System.out.printf("The floor of %f is %f\n", d02, ceil02);     } }</pre>
105.	<p>Give the signum of each of the following <code>double</code> values:</p> <ol style="list-style-type: none"> <li>1. -55.6</li> <li>2. 0.0004</li> <li>3. 0</li> <li>4. 79</li> </ol>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double num01=-55.6, num02=0.0004, num03=0.0;         int num04=79;          double signum01 = Math.signum(num01);         double signum02 = Math.signum(num02);         double signum03 = Math.signum(num03);         double signum04 = Math.signum(num04);     } }</pre>

		<pre> System.out.printf("signum of %f is %f\n", num01, signum01); System.out.printf("signum of %f is %f\n", num02, signum02); System.out.printf("signum of %f is %f\n", num03, signum03); System.out.printf("signum of %d is %f\n", num04, signum04); } } </pre>
106.	I have two double variables $x$ and $y$ . Write a program that returns a number with the magnitude of $x$ but the sign of $y$ . Test the program with the following pairs $(x, y)$ of values: <ul style="list-style-type: none"><li>• -89.5, 200</li><li>• 400, -300</li></ul> <i>Hint: Math.copySign</i>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         double set1_num1=-89.5, set1_num2=200;         double set2_num1=400, set2_num2=-300;          double result1 = Math.copySign(set1_num1, set1_num2);         double result2 = Math.copySign(set2_num1, set2_num2);          System.out.printf("First pair result = %f \n", result1);         System.out.printf("Second pair result = %f \n", result2);     } } </pre>
107.	<b>IMPORTANT:</b> Print out the result of the division of the following calculation: $a/b$ , where $a$ and $b$ are both integers of value 1 and 4 respectively. Put the result into a <code>float</code> . Explain the result.	An initial attempt at writing the code might yield the following: <pre> public class PracticeYourJava {     public static void main(String[] args) {         int a=1, b=4;         float result = a/b; // WRONG !!!         System.out.println(result);     } } </pre> <b>Explanation:</b> The result of this code is <code>0</code> ! The issue here is that on seeing integer values (1 and 4 in this case), Java automatically performs an integer calculation (irrespective of the type of the target output variable). If you want the correct results, then ensure that you cast the input variables appropriately. In this case we would write the following calculation for <code>result</code> : <pre> float result = (float)a/(float)b; </pre>
108.	What is the expected output of this code fragment?	The program will stop running and it will throw the exception ' <code>java.lang.ArithmaticException</code> ' for this integer divide by zero situation.
	<b>Infinity/-Infinity/NaN</b>	
109.	Write a program which determines and prints out the result of the following computations given the noted variable values:  <code>float a=0, b=0, c=-1, d=1;</code> 1. $a/b$ 2. $c/a$ 3. $d/a$	<pre> public class PracticeYourJava {     public static void main(String[] args) {         float a = 0, b=0, c=-1, d=1;         System.out.printf("%f/%f = %f\n", a, b, a / b);         System.out.printf("%f/%f = %f\n", c, a, c / a);         System.out.printf("%f/%f = %f\n", d, a, d / a);     } }  <b>Results</b> 1. 0/0 = NaN (which means <u>Not a Number</u>) 2. -1/0 = -Infinity 3. 1/0 = Infinity </pre>

## Practice Your Java Level 1

<p>110. Write a program which determines and prints out the result of each of the following calculations:</p> <ol style="list-style-type: none"> <li>1. <math>-\infty + \infty</math></li> <li>2. <math>\infty / \infty</math></li> <li>3. <math>-\infty / \infty</math></li> <li>4. <math>\infty / 0</math></li> <li>5. <math>-\infty / 0</math></li> </ol> <p><i>Hint: The constants  <code>Float.POSITIVE_INFINITY</code>,  <code>Float.NEGATIVE_INFINITY</code></i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float a = Float.POSITIVE_INFINITY;         float b = Float.NEGATIVE_INFINITY;         System.out.printf("%f+%f = %f\n", b, a, b + a);         System.out.printf("%f/%f = %f\n", b, a, b / a);         System.out.printf("%f/%f = %f\n", b, b, b / b);         System.out.println(a + "/0 = " + (a/0));         System.out.println(b + "/0 = " + (b/0));         //System.out.printf("%f/%f=%f\n", a, 0, a / 0);         //System.out.printf("%f/%f=%f\n", b, 0, b / 0);     } }</pre> <p><b>Results</b></p> <ol style="list-style-type: none"> <li>1. <math>-\text{Infinity} + \text{Infinity} = \text{NaN}</math></li> <li>2. <math>-\text{Infinity} / \text{Infinity} = \text{NaN}</math></li> <li>3. <math>-\text{Infinity} / -\text{Infinity} = \text{NaN}</math></li> <li>4. <math>\text{Infinity} / 0 = \text{Infinity}</math></li> <li>5. <math>-\text{Infinity} / 0 = -\text{Infinity}</math></li> </ol>
<b>Complex Numbers</b>	
<p>111. We have the complex number <math>10 + 5i</math>, with each of its numerical values expressed in a <code>double</code> variable. Express it in polar form, i.e. <math>(r, \theta)</math>.</p> <p><i>Hint: <code>Math.hypot</code>, <code>Math.atan2</code></i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double x_var1 = 5;         double y_var1 = 10;          double r = Math.hypot(x_var1, y_var1);         double theta = Math.atan2(x_var1, y_var1);         System.out.printf("The complex number in polar form is (%f, %f)\n", r, theta);     } }</pre>
<b>Increment, decrement and shorthand operators</b>	
<p>112. Given the following initialized variable:  <code>int i=2;</code>  State what its value is after the statement:  <code>i++;</code></p>	<p>3  The increment operator(<code>++</code>) simply means that the value of the variable in question should be incremented by <b>1</b>.</p>
<p>113. Given the following variable:  <code>double i=8.46;</code>  State what its value is after the statement:  <code>i++;</code></p>	<p>9.46  The increment operator also works on floating point numbers, incrementing the value of the variable in question by <b>1</b>.</p>
<p>114. Given the following initialized variable:  <code>int i=2;</code>  State what its value is after the statement:  <code>++i;</code></p>	<p>3  It does not matter for a variable <u>in a standalone statement like this</u> whether the increment operator appears before or after the variable. The same logic applies to the decrement operator.</p>
<p>115. Given the following variable: <code>int i=26;</code>  State what its value is after the statement:  <code>i--;</code></p>	<p>25  The decrement operator(<code>--</code>) simply means that the value of the variable in question should be decremented by <b>1</b>.</p>
<p>116. Given the following variable: <code>int i=23;</code>  State what the values of <code>j</code> and <code>i</code> are after the following statement:  <code>int j=++i;</code></p>	<p>This statement is saying <code>j = incremented value of i</code>  Thus <code>i</code> is incremented first <i>before</i> assigning the incremented value of <code>i</code> to <code>j</code>. Thus the result is:  <code>j=24</code>  <code>i=24</code></p>
<p>117. Given the following variable:  <code>int i=23;</code>  State what the values of <code>j</code> and <code>i</code> are after the following statement:  <code>int j=i++;</code></p>	<p><code>j=23</code>  <code>i=24</code>  The statement <code>j=i++;</code> means the following:  assign <code>i</code> to <code>j</code> and then afterwards increment <code>i</code>  Thus <code>j</code> is assigned the current value of <code>i</code>, that is <b>23</b> and then <code>i</code> is</p>

		incremented to 24.
118.	Given the following code snippet: <pre>int i=5; j=27; int result = i++ + --j;</pre> 1. What is the expected value of <i>i</i> ? 2. What is the expected value of <i>j</i> ? 3. What is the expected value of <i>result</i> ?	1. <i>i</i> = 6 2. <i>j</i> = 26 3. <i>result</i> = 5+26 = 31 In words we can say that the line of code means the following: <i>result</i> = ( <i>i</i> , (then increment <i>i</i> ), PLUS (decremented value of <i>j</i> ))
119.	Given the following code snippet: <pre>int i=5; j=27; result = --i - --j;</pre> 1. What is the expected value of <i>i</i> ? 2. What is the expected value of <i>j</i> ? 3. What is the expected value of <i>result</i> ?	1. <i>i</i> =4 2. <i>j</i> =26 3. <i>result</i> = 4-26 = -22 In words we can say: <i>result</i> = (decrement <i>i</i> MINUS decrement <i>j</i> )

**Shorthand operators**

120.	Given two numerical variables <i>g</i> and <i>y</i> , what does the following statement mean? <i>g+=y</i> ;	This means <i>increment g by y</i> , in short, <i>g</i> = <i>g+y</i> ;
121.	Given two numerical variables <i>g</i> and <i>y</i> , what does the following statement mean? <i>g*=y</i> ;	This means <i>multiply g by y</i> , in short it means exactly the same thing as: <i>g</i> = <i>g*y</i> ;
122.	Given two numerical variables <i>g</i> and <i>y</i> , what does the following statement mean? <i>g/=y</i> ;	This means <i>divide g by y</i> , in short, <i>g</i> = <i>g/y</i> ;
123.	Given two numerical variables <i>g</i> and <i>y</i> , what does the following statement mean? <i>g-=y</i> ;	This means <i>decrement g by y</i> , in short, <i>g</i> = <i>g-y</i> ;
124.	Given the following variable: <code>int i=12</code> , state the value of <i>i</i> after the statement <code>i+=29</code> ;	<i>i=41</i> The given statement is a shorthand way of saying: <i>i</i> = <i>i+29</i> ; (i.e. <i>add 29 to i</i> )
125.	Given the following variable: <code>int i=39</code> , state the value of <i>i</i> after the statement <code>i/=3</code> ;	<i>i=13</i> The given statement is a shorthand way of saying: <i>i</i> = <i>i/3</i> ; (i.e. <i>divide i by 3</i> )
126.	Given the following variable: <code>int j=9</code> , state the value of <i>j</i> after the statement <code>j*=6</code> ;	<i>j=54</i> The given statement is a shorthand way of saying: <i>j</i> = <i>j*6</i> ; (i.e. <i>multiply j by 6</i> )
127.	Given the following variable: <code>float x=5.2f</code> , state the value of <i>x</i> after the following statement: <code>x-=2</code> ;	3.2 The given statement is a shorthand way of saying: <i>x</i> = <i>x-2</i> ; (i.e. <i>decrement x by 2</i> )
128.	What is the difference between the following two statements? <code>i+=1;</code> <code>i++;</code>	There is no difference; both mean the same thing, which is <i>increment i by 1</i> .

129.	Describe the concept of integer <u>overflow</u> .
	The concept of integer overflow can be described as follows. Using the type <code>int</code> as our example, imagine that all its values are represented <u>on a wheel</u> , starting from the lowest integer to the highest integer. Now, say we have <code>int x=Integer.MAX_VALUE</code> (largest integer). if we add the value of 1 to <i>x</i> , i.e. <code>x=x+1</code> , then interestingly what happens is simply that <i>x</i> is moved to the next point on the wheel, which is <code>Integer.MIN_VALUE!</code>

## Practice Your Java Level 1

	<p>If I added 2 to the value of <code>x</code>, then <code>x</code> would be moved to the position <code>Integer.MIN_VALUE - 1</code>. This is how integer overflow occurs in Java.</p> <p>Integer overflow should be assumed to be a risk for any integer computation.</p> <p>Note: Even though we used the <code>int</code> type as an example, this concept applies to any integer type.</p>
130.	<p>Describe the concept of integer <u>underflow</u>.</p> <p>Integer underflow happens when the result of an integer computation results in a value that is less than (i.e. closer to <math>-\infty</math>) the minimum value that the integer type in question can contain. Continuing with the previous analogy of <code>int</code> values on a wheel, imagine we have <code>x = Integer.MIN_VALUE</code>. If we subtract the value of 1 from <code>x</code>, i.e. <code>x=x-1</code>, we are moving backwards on the wheel and the result is <code>x = Integer.MAX_VALUE</code>, which is clearly incorrect!</p>
131.	<p>How do I guard against the integer overflow/underflow issue?</p> <p>Use the “Exact” set of methods (<i>introduced in Java 8</i>) of the class <code>Math</code> instead of the regular integer operators.</p>
132.	<p>What is the result if a computation overflows or underflows when we use an “Exact” method?</p> <p>An <u>exception</u> will be generated by the program. If the exception is not explicitly caught by the program, the program will stop running. We will look at exceptions in a later chapter.</p>
133.	<p>I have two <code>int</code> variables <code>a</code> and <code>b</code>, the addition of which has a risk of overflowing and I want to catch this overflow. Write a sample program to show how this would be done.</p> <p>State what the output of this program is. Use a value of <code>Integer.MAX_VALUE</code> for both variables <code>a</code> and <code>b</code>.</p> <p><i>Hint: Math.addExact</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MAX_VALUE; //just a value for the purposes of this program         int b = Integer.MAX_VALUE; //as above         int total;          total = Math.addExact(a, b);         System.out.println(total);     } }</pre> <p><b>Note(s):</b></p> <ol style="list-style-type: none"> <li>1. Running this code for the choice of numbers herein will result in an overflow, which causes the throwing of the <u>exception</u> <code>java.lang.ArithmaticException</code>. Exceptions are discussed in a later chapter.</li> <li>2. Note that if you did not use the <code>Math.addExact</code> method keyword, the result would overflow anyway, but <u>not</u> inform you (see the next exercise).</li> <li>3. The same exception occurs for integer underflow conditions.</li> </ol>
134.	<p>I have two <code>int</code> variables <code>a</code> and <code>b</code> each with the value <code>Integer.MAX_VALUE</code>. Add these using the addition operator and print out the result. Contrast the results of this computation with the result of the preceding exercise.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MAX_VALUE; //just a value for the purposes of this program         int b = Integer.MAX_VALUE; //as above         int total;          total = a + b;         System.out.println(total);     } }</pre> <p><b>Output:</b> -2</p> <p><b>Notes:</b> The result of the computation is clearly wrong due to the fact that it has overflowed! Also, note that there was no exception generated!</p> <p><b>Conclusion:</b> If we want to detect under/overflow conditions, then the “Exact” methods should be used. However, their exceptions should be caught in order to avoid program stoppage due to the exception being thrown. We look at how to handle exceptions in a later chapter.</p>

135.	<p>We have the <code>int</code> variable, <code>a = Integer.MAX_VALUE</code>. Write two programs to show the result of incrementing the value <code>a</code> by 1. The first program should use the standard increment operator, the second should use the <code>Math.incrementExact</code> method.</p> <pre data-bbox="241 276 796 487"><code>public class PracticeYourJava {     public static void main(String[] args){         int a = Integer.MAX_VALUE;         int total = ++a;         System.out.println(total);     } }</code></pre> <pre data-bbox="822 276 1377 487"><code>public class PracticeYourJava {     public static void main(String[] args){         int a = Integer.MAX_VALUE;         int total = Math.incrementExact(a);         System.out.println(total);     } }</code></pre> <p><b>Result:</b> The computation results in an overflow. However, in the first case, the overflow is silent, thus putting an erroneous result <code>Integer.MIN_VALUE</code> into the variable <code>total</code>. In the second case using <code>Math.incrementExact</code> the following exception is generated: <code>java.lang.ArithmeticException</code>.</p>
136.	<p>We have two <code>int</code> values, <code>a = Integer.MAX_VALUE</code> and <code>b = 2</code>. Write two programs to show the result of multiplying <code>a</code> by <code>b</code>. The first program should use the standard multiplication operator, the second should use the <code>Math.multiplyExact</code> method.</p> <pre data-bbox="241 709 796 963"><code>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MAX_VALUE;         int b = 2;          int total = a * b;         System.out.println(total);     } }</code></pre> <pre data-bbox="822 709 1410 963"><code>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MAX_VALUE;         int b = 2;          int total = Math.multiplyExact(a, b);         System.out.println(total);     } }</code></pre> <p><b>Result:</b> As expected, the computation results in an overflow. However, in the first case, the overflow is silent, thus putting an erroneous result into the variable <code>total</code>: -2. In the second case using <code>Math.multiplyExact</code> the following exception is generated: <code>java.lang.ArithmeticException</code>.</p>
137.	<p>We have two <code>int</code> values, <code>a = Integer.MIN_VALUE</code> and <code>b = 1</code>. Write two programs to show the result of subtracting <code>b</code> from <code>a</code>. The first program should use the standard subtraction operator, the second should use the <code>Math.subtractExact</code> method.</p> <pre data-bbox="241 1184 796 1438"><code>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MIN_VALUE;         int b = 2;          int total = a - b;         System.out.println(total);     } }</code></pre> <pre data-bbox="822 1184 1410 1438"><code>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MIN_VALUE;         int b = 2;          int total = Math.subtractExact(a, b);         System.out.println(total);     } }</code></pre> <p><b>Result:</b> As expected, the computation results in an <u>underflow</u>. However, in the first case, the underflow is silent, thus putting a clearly erroneous result into the variable <code>total</code>: 2147483646. In the second case using <code>Math.subtractExact</code> the following exception is generated: <code>java.lang.ArithmeticException</code>.</p>
138.	<p>We have the following <code>int</code> variable, <code>a = Integer.MIN_VALUE</code>. Write two programs to show the result of decrementing the value <code>a</code> by 1. The first program should use the standard decrement operator, the second should use the <code>Math.decrementExact</code> method.</p> <pre data-bbox="241 1632 796 1864"><code>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MIN_VALUE;          int total = --a;         System.out.println(total);     } }</code></pre> <pre data-bbox="822 1632 1410 1864"><code>public class PracticeYourJava {     public static void main(String[] args) {         int a = Integer.MIN_VALUE;          int total = Math.decrementExact(a);         System.out.println(total);     } }</code></pre> <p><b>Result:</b> As expected, the computation results in an <u>underflow</u>. However, in the first case, the underflow is silent, thus putting a clearly erroneous result into the variable <code>total</code>: 2147483647. In the second case using</p>

## Practice Your Java Level 1

	<code>Math.decrementExact</code> the following exception is generated: <code>java.lang.ArithmeticException</code> .	
139.	We have the following <code>int</code> variable, <code>a = Integer.MIN_VALUE</code> . Write two programs to show the result of attempting to negate this value. The first program should use the negation operator ( <code>-</code> ) and the second should use the method <code>Math.negateExact</code> . State the result of the program and explain why it is so.	
	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int a = Integer.MIN_VALUE;          int total = -a;         System.out.println(total);     } }</pre>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int a = Integer.MIN_VALUE;          int total = Math.negateExact(a);         System.out.println(total);     } }</pre>
	<p><b>Result:</b> An exception, <code>java.lang.ArithmeticException</code> is generated.</p> <p><b>Reason:</b> When worked out manually, the result of this negation = <code>Integer.MAX_VALUE + 1</code>. This clearly is a number that an <code>int</code> cannot contain (this is an overflow scenario) and therefore the method <code>negateExact</code> throws an exception.</p>	
140.	We have the following <code>long</code> variable, <code>a = 3047483688</code> . Write two separate programs to cast it to an <code>int</code> , the first program using a cast and the second using the method <code>Math.toIntExact</code> . Observe and comment on the output of each program.	
	<pre>public class PracticeYourJava {     public static void main(String[] args) {          long a = 3047483688L;          int result = (int)a;         System.out.println(result);     } }</pre>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          long a = 3047483688L;          int total = Math.toIntExact(a);         System.out.println(total);     } }</pre>
	<p><b>Result:</b> The first solution gives the clearly erroneous result of this cast: <code>-1247483608</code>. The second solution rightfully throws the exception <code>java.lang.ArithmeticException</code>.</p>	
141.	<p>If we have integer computations whose input values or results we expect to exceed the value <code>Long.MAX_VALUE</code>, what means can we use to safely handle such numbers in Java?</p> <p>Use the class <code>BigInteger</code>. This class represents arbitrarily long integers. Exercises on the class <code>BigInteger</code> are presented in Chapter 12, entitled <i>Number Handling III</i>.</p>	

Binary Numbers (Base 2)		
142.	Write a program which assigns the binary value <code>00001110</code> to an <code>int</code> variable named <code>var01</code> . Print out the equivalent base 10 number.	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int var01 = 0b00001110;         System.out.println(var01);     } }</pre>
143.	Write a program which adds the binary values <code>1000011</code> and <code>1111101</code> . Print out the resulting base 10 number.	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int var01 = 0b1000011;         int var02 = 0b1111101;         int var03 = var01 + var02;          System.out.println(var03);     } }</pre>

Hexadecimal numbers		
144.	Write a program which assigns the	<pre>public class PracticeYourJava {     public static void main(String[] args) {     }</pre>

	<p>hexadecimal value FFF0 to an int variable named var01. Print out the equivalent base 10 number.</p>	<pre>int var01 = 0xFFFF0; System.out.println(var01); }</pre>
145.	<p>Write a program which adds the hexadecimal values 1EC05 and F238C. Print out the resulting base 10 number.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int var01 = 0x1EC05;         int var02 = 0xF238C;         int var03 = var01 + var02;         System.out.println(var03);     } }</pre> <p><b>Note:</b> later we'll see how to print out the result in hexadecimal.</p>

Random Number Generation		
146.	<p>Write a program which will generate a random double value <math>x</math>, where the following is the case:  <math>0 \leq x &lt; 1</math>  <i>Hint: Math.random</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double random01 = Math.random();         System.out.println(random01);     } }</pre>
147.	<p>What is the upper limit of the value generated by the method <code>Math.Random</code>?</p>	<p>The upper value is defined as <math>&lt; 1</math>. This means that it doesn't quite get to 1.</p>
148.	<p>The answer to the preceding exercise being the case (where the upper limit does not get to 1) write a program which will generate a random double value <math>x</math> where the following is the case:  <math>0 \leq x &lt; 10</math></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         double random01 = Math.random() * 10;         System.out.println(random01);     } }</pre>
149.	<p>Write a program which will generate a random integer value between 1 and 10.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int random01 = (int) Math.ceil(Math.random() * 10);         System.out.println(random01);     } }</pre> <p><b>Note:</b> Further exercises on random numbers are presented in Chapter 20 entitled <i>Random Numbers</i>.</p>

Formatting		
	<p>In this section we move on to exercises with which to hone our skills on formatting mathematical output. In particular we look at the formatting options available for the method <code>printf</code>. Unless otherwise specified in this section, every output directive implies the usage of <code>printf</code>. It should be noted that the method <code>println</code> does have formatting specifiers but we do not address those here.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int m = 304638;         System.out.printf("In decimal      :%d\n", m);         System.out.printf("In hexadecimal :%x\n", m);         System.out.printf("In octal       :%o\n", m);     } }</pre>

## Practice Your Java Level 1

151.	<p>Given the <code>int</code> variable <code>m=304638</code>, print the value of this variable out in hexadecimal, ensuring that the letters in the hexadecimal number are uppercase.</p> <p><i>Hint: %X</i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int m = 304638;         System.out.printf("In hexadecimal :%X\n", m);     } }</pre>																										
152.	<p>Write a program which adds the hexadecimal values <code>1EC05</code> and <code>F238C</code>. Print out the resulting base 16 number.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int var01 = 0x1EC05;         int var02 = 0xF238C;         int result = var01 + var02;         System.out.printf("result = %X\n", result);     } }</pre>																										
153.	<p>Run this program and comment on each line of output, explaining why each is so.</p>	<table border="0"> <thead> <tr> <th style="text-align: left; vertical-align: top;">public class FormattingExercises {     public static void main(String[] args){         float var01 = 20.59f;         System.out.printf("A) %f\n\n", var01);         System.out.printf("B) %5.3f\n", var01);         System.out.printf("C) %1.2f\n", var01);         System.out.printf("D) %5.1f\n", var01);         System.out.printf("E) %5.0f\n", var01);         System.out.printf("F) %5f\n\n", var01);         System.out.printf("G) %.4f\n", var01);         System.out.printf("H) %.3f\n", var01);         System.out.printf("I) %.2f\n", var01);         System.out.printf("J) %.1f\n", var01);         System.out.printf("K) %.0f\n", var01);         System.out.printf("L) %.15f\n", var01);     } }</th> <th style="text-align: left; vertical-align: top; padding-left: 20px;"><b>Output</b></th> </tr> </thead> <tbody> <tr> <td></td><td>A) 20.590000</td></tr> <tr> <td></td><td>B) 20.590</td></tr> <tr> <td></td><td>C) 20.59</td></tr> <tr> <td></td><td>D) 20.6</td></tr> <tr> <td></td><td>E) 21</td></tr> <tr> <td></td><td>F) 20.590000</td></tr> <tr> <td></td><td>G) 20.5900</td></tr> <tr> <td></td><td>H) 20.590</td></tr> <tr> <td></td><td>I) 20.59</td></tr> <tr> <td></td><td>J) 20.6</td></tr> <tr> <td></td><td>K) 21</td></tr> <tr> <td></td><td>L) 20.590000152587890</td></tr> </tbody> </table>	public class FormattingExercises { public static void main(String[] args){ float var01 = 20.59f; System.out.printf("A) %f\n\n", var01); System.out.printf("B) %5.3f\n", var01); System.out.printf("C) %1.2f\n", var01); System.out.printf("D) %5.1f\n", var01); System.out.printf("E) %5.0f\n", var01); System.out.printf("F) %5f\n\n", var01); System.out.printf("G) %.4f\n", var01); System.out.printf("H) %.3f\n", var01); System.out.printf("I) %.2f\n", var01); System.out.printf("J) %.1f\n", var01); System.out.printf("K) %.0f\n", var01); System.out.printf("L) %.15f\n", var01); } }	<b>Output</b>		A) 20.590000		B) 20.590		C) 20.59		D) 20.6		E) 21		F) 20.590000		G) 20.5900		H) 20.590		I) 20.59		J) 20.6		K) 21		L) 20.590000152587890
public class FormattingExercises { public static void main(String[] args){ float var01 = 20.59f; System.out.printf("A) %f\n\n", var01); System.out.printf("B) %5.3f\n", var01); System.out.printf("C) %1.2f\n", var01); System.out.printf("D) %5.1f\n", var01); System.out.printf("E) %5.0f\n", var01); System.out.printf("F) %5f\n\n", var01); System.out.printf("G) %.4f\n", var01); System.out.printf("H) %.3f\n", var01); System.out.printf("I) %.2f\n", var01); System.out.printf("J) %.1f\n", var01); System.out.printf("K) %.0f\n", var01); System.out.printf("L) %.15f\n", var01); } }	<b>Output</b>																											
	A) 20.590000																											
	B) 20.590																											
	C) 20.59																											
	D) 20.6																											
	E) 21																											
	F) 20.590000																											
	G) 20.5900																											
	H) 20.590																											
	I) 20.59																											
	J) 20.6																											
	K) 21																											
	L) 20.590000152587890																											

### Comments/Explanation

This exercise shows more precise formatting options that are available for the `%f` parameter. We go through each one of the cases presented.

*Note:* The output format pertains to how we want the floating point value to be displayed on the screen, it is does not pertain to how it is stored internally.

A) **20.590000** The value has been output in the default manner in which data is output for the `%f` formatting parameter.

B) **20.590** In the formatting parameter `%5.3f`, the `5` refers to the total field width of the value that we want to output, including the decimal point. The `3` after the decimal point refers to how many digits we want after the decimal point. Given the number `20.59` (which has 5 characters) that we want displayed this fits exactly. Note though, that if the specified field width does not provide sufficient space for the whole number portion of the number being displayed, Java will ignore that field width constraint and will always print out the full whole number portion. Consider this to be a safety feature which ensures that erroneous values (due to truncation) are not presented to the user. It will also always print out the fractional part of the number to the number of spaces specified.

C) **20.59** We used the formatting parameter `%1.2f`, implying a field width of 1 (which is clearly too small) for the whole number and 2 characters after the decimal point. As stated in the preceding explanation, if the field width is too small for the whole number portion (`20` requires 2 character widths), Java will ignore the constraint of the field width which we see it has done here. As requested, it printed out the two characters after the decimal point, further buttressing the point that it will display what is required; if the field width is too small it becomes a secondary factor.

- D) 20.6 We used the formatting parameter `%5.1f`, thus ensuring only 1 character after the decimal point. Observe that rounding was applied to the number. Also observe the alignment of the number within the 5 character field.
- E) 21 We used the formatting parameter `%5.0f`, thus ensuring no digits after the decimal point. Observe that the number was rounded up.
- F) 20.590000 We used the formatting parameter `%5f`, thus ensuring a field width of five characters and implicitly telling `printf` to take care of the decimal portion as it sees fit; we see `printf` simply applying the standard 6 decimal place formatting for `%f` in this case. Again, `printf` ignored the field width which was too small.
- G) 20.5900 Formatting parameter `%.4f`, stating “we don’t care how you display the digits before the decimal point, just ensure that there are only 4 digits after the decimal point”.
- H) 20.590 Similar to the preceding, only that we stated we only want 3 digits after the decimal point.
- I) 20.59 Similar to the preceding, only that we stated we only want 2 digits after the decimal point.
- J) 20.6 Similar to the preceding, only that we stated we only want 1 digit after the decimal point. We see that Java applies rounding to the value.
- K) 21 Similar to the preceding, only that we stated we don’t want any digits after the decimal point. We see that Java applies rounding to the value.
- L) 20.590000152587890 Similar to the preceding, only that we stated we want 15 places after the decimal point. But see, the result wasn’t `20.59000000000000`, but rather a corrupted number! This corruption is due to the way in which computers handle floating point numbers; they sometimes have problems representing them properly internally. Obviously this would be a problem for such issues like the representation of currency values. Later in this book we see how to avoid this misrepresentation of floating point values.

154.	<p>We have the following <code>float</code> numbers, 8.91, -8.9, 27.338 and 1768000111.77. Print each of these out, bounded on both sides by the vertical bar. State your observations of the output.</p>	<pre><code>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=27.338, f4=1768000111.77;         System.out.printf(" %f \n", f1);         System.out.printf(" %f \n", f2);         System.out.printf(" %f \n", f3);         System.out.printf(" %f \n", f4);     } }</code></pre> <p><b>Observations:</b></p> <ol style="list-style-type: none"> <li>1. In all cases, the numbers were written out to 6 decimal places.</li> <li>2. The values are left-aligned.</li> <li>3. The output numbers were not necessarily exactly what was input (discussed previously).</li> </ol>
155.	<p>Repeat the preceding exercise, this time printing the numbers out in a field with a minimum width of 7 characters.</p>	<pre><code>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=27.338, f4=1768000111.77;         System.out.printf(" %7f \n", f1);         System.out.printf(" %7f \n", f2);         System.out.printf(" %7f \n", f3);         System.out.printf(" %7f \n", f4);     } }</code></pre>
156.	<p>Repeat the preceding exercise, this time printing the numbers out in a character field minimum 8 characters minimum, to a precision of 2 decimal places. State your observations of the output.</p>	<pre><code>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=27.338, f4=1768000111.77;         System.out.printf(" %8.2f \n", f1);         System.out.printf(" %8.2f \n", f2);         System.out.printf(" %8.2f \n", f3);         System.out.printf(" %8.2f \n", f4);     } }</code></pre>

## Practice Your Java Level 1

		<p><b>Observations:</b> the numbers are right aligned (contrast this with the results of the preceding exercise). The largest number, which is stored in the variable f4 is larger than the specified minimum field width.</p> <p><b>Note(s):</b> Even if you give <code>printf</code> a field width specifier, if the number that it is supposed to print out exceeds the width specified, Java will ignore the specified field width and print the number out. This in a sense is a safety feature which prevents the user of the output from being presented with wrong data.</p> <p><b>Conclusion:</b> For tidy output, ensure that the chosen field width is at least the width of the maximum value that is being displayed.</p>
157.	Repeat the above exercise using a field width of 14 characters and 2 decimal places.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %14.2f \n", f1);         System.out.printf(" %14.2f \n", f2);         System.out.printf(" %14.2f \n", f3);         System.out.printf(" %14.2f \n", f4);     } }</pre>
158.	What is the difference between the %f and %e floating point format specifiers?	<p>The %e specifier prints the data out in exponential format, as opposed to the %f specifier which prints the full number out. The exponent itself is written out as “e&lt;sign&gt;&lt;value&gt;”, for example <math>10^{-7}</math> is written as e-07. The number <math>10^{-29}</math> is written out as e-29.</p>
159.	Print out the value <code>Float.MIN_VALUE</code> using the %e specifier.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.printf("%e\n", Float.MIN_VALUE);     } }</pre>
160.	Repeat the above exercise using a field width of 9 characters and 2 decimal places. Use the field specifier %e instead of %f.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %9.2e \n", f1);         System.out.printf(" %9.2e \n", f2);         System.out.printf(" %9.2e \n", f3);         System.out.printf(" %9.2e \n", f4);     } }</pre>
161.	Repeat the preceding exercise using the %E format specifier and a minimum field width of 9, with 2 decimal places. Observe and comment on the differences between this output and the output in the preceding exercise.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %9.2E \n", f1);         System.out.printf(" %9.2E \n", f2);         System.out.printf(" %9.2E \n", f3);         System.out.printf(" %9.2E \n", f4);     } }</pre> <p><b>Observation:</b> The difference between this output and the output with the %e format is simply that the “E” that represents the exponent is written as an uppercase letter with the %E format specifier.</p>
162.	Describe the functioning of the %g format.	<p>When a floating point number is printed out using the %g format specifier and unconstrained by a precision specifier, the floating point number is printed out as follows: if the precision of the number is less than 6 significant digits, then it prints the full number out. If the precision of the number is greater than 6 digits, then it</p>

	<p>prints the number out in exponential format. When a precision specifier is present, a number whose whole part is less than the width of the precision specifier will be printed out in full (with rounding), otherwise the number is printed out in exponential format.</p>	
163.	Repeat exercise 161 using the %g format specifier without any field width or precision specifiers.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %g \n", f1);         System.out.printf(" %g \n", f2);         System.out.printf(" %g \n", f3);         System.out.printf(" %g \n", f4);     } }</pre>
164.	Repeat the preceding exercise, this time however with a minimum field width of 9 and a precision of 2 places.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %9.2g \n", f1);         System.out.printf(" %9.2g \n", f2);         System.out.printf(" %9.2g \n", f3);         System.out.printf(" %9.2g \n", f4);     } }</pre>
165.	Redo the preceding exercise ensuring that the sign even of the positive numbers is made to appear in the output.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %+9.2g \n", f1);         System.out.printf(" %+9.2g \n", f2);         System.out.printf(" %+9.2g \n", f3);         System.out.printf(" %+9.2g \n", f4);     } }</pre>
166.	Redo the preceding exercise, ensuring that the output is left-justified.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %-9.2f \n", f1);         System.out.printf(" %-9.2f \n", f2);         System.out.printf(" %-9.2f \n", f3);         System.out.printf(" %-9.2f \n", f4);     } }</pre>
167.	Redo exercise the preceding exercise, ensuring that negative values are represented by brackets rather than by the negative sign.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %(9.2f) \n", f1);         System.out.printf(" %(9.2f) \n", f2);         System.out.printf(" %(9.2f) \n", f3);         System.out.printf(" %(9.2f) \n", f4);     } }</pre> <p><b>Observation:</b> Note that the numbers are right justified.</p>
168.	Redo the preceding exercise, ensuring that the values are left justified.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f1=8.91, f2=-8.9, f3=327.338, f4=1768000111.77;         System.out.printf(" %(-9.2f) \n", f1);         System.out.printf(" %(-9.2f) \n", f2);         System.out.printf(" %(-9.2f) \n", f3);     } }</pre>

## Practice Your Java Level 1

		<pre>         System.out.printf(" %-9.2f \n", f4);     } } </pre>
169.	Print the floating point values 1000, 120,000, -115,000 and 145,000.7743, ensuring that a grouping specifier appears in the output (for example, if the value is 1000, then print out 1,000). <b>Note:</b> the grouping specifier printed out is locale specific; for example, in some countries in Europe, it would be a “.”, instead of a “,”.	<pre> public class PracticeYourJava {     public static void main(String[] args) {         float f1=1000f, f2=120000f, f3=-115000f, f4=145000.7743f;         System.out.printf("% ,f\n", f1);         System.out.printf("% ,f\n", f2);         System.out.printf("% ,f\n", f3);         System.out.printf("% ,f\n", f4);     } } </pre>
170.	Using a formatting specifier, print out the value 87.5 as 87.5%	<pre> public class PracticeYourJava {     public static void main(String[] args) {         float f1=87.5f;         System.out.printf("%f%\n", f1);     } } </pre>
171.	Modify the preceding exercise to print the number out to only 2 decimal places.	<pre> public class PracticeYourJava {     public static void main(String[] args) {         float f1=87.5f;         System.out.printf("%.2f%\n", f1);     } } </pre>

### logical & bitwise operators

172. Write a program to output the logical AND of the values 5 and 2.

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int a = 5, b = 2;
        int result = a & b;
        System.out.println(result);
    }
}

```

173. Write a program to output the logical OR of the values 5 and 3.

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int a = 5, b = 3;
        int result = a | b;
        System.out.println(result);
    }
}

```

174. Write a program to output the logical XOR of the values 5 and 3.

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int a = 5, b = 3;
        int result = a ^ b;
        System.out.println(result);
    }
}

```

175. Write a program to output the one's complement of 26.

```

public class PracticeYourJava {

```

	<pre>public static void main(String[] args) {     int a = 26;     int result = ~a;     System.out.println(result); }</pre>
176.	<p>Write a program which will <i>left-shift</i> by four places the value 38. Print the result to the console.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int a = 38, b = 4;         int result = a &lt;&lt; b;         System.out.println(result);     } }</pre>
177.	<p>What is the difference between</p> <pre>int a=38; result = a*2*2*2*2;</pre> <p>and</p> <pre>int a=38; result = a &lt;&lt; 4;</pre> <p>And why is the result what it is?</p> <p>There is no difference in the result. The reason for the equivalence is because computers represent data in base 2, therefore left shifting a number is the equivalent of multiplying it by 2; and thus left shifting by 4 means multiplying by <math>2^4</math>. Think about it this way: normally we operate in base 10. Left shifting in base 10, would mean multiplying by 10. Left shifting 4 times in base 10 would mean multiplying the number by <math>10^4</math>.</p>
178.	<p>Write a program which will <i>right-shift</i> by two places the value 38. Explain the result.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int a = 38, b = 2;         int result = a &gt;&gt; b;         System.out.println(result);     } }</pre> <p><b>Explanation</b> Right shifting is equivalent to doing an integer division by 2 and returning the quotient of the result. Therefore right-shifting 2 times means, divide by 2 (integer division, ignoring the remainder), then divide the result (the quotient of the earlier computation) by 2 which yields the following computation and result:</p> <pre>38 &gt;&gt; 1 = 19 19 &gt;&gt; 1 = 9</pre>
E1	Given the following two complex numbers, $10 + 5i$ and $2 + 9i$ , write a program to divide the first number by the 2 <sup>nd</sup> ( <i>Hint</i> : convert both to polar form, do the computation and then convert back to Cartesian format).
E2	Write your own equivalent of the <code>Math.copySign</code> method. <i>Hint: Math.signum</i>



# Chapter 4. Boolean operations - The boolean Primitive

This chapter presents exercises which enable you to practice your knowledge of the **boolean** primitive type and its associated operators. Exercises on compound boolean expressions are also presented.

1.	Describe what the <b>boolean</b> type is.	The <b>boolean</b> is a type which represents logical <i>true</i> or logical <i>false</i> .
2.	What is the range of values that the Java type <b>boolean</b> can support?	Either of the following two distinct values: <b>true</b> or <b>false</b> ( <i>case sensitive</i> ).
3.	Write a line of code to declare a <b>boolean</b> variable named <b>b01</b> and assign it a value of <b>true</b> .	<b>boolean b01 = true;</b>

## Boolean Testing

4. I have an **int** variable **x** with a value of 5 and an **int** variable **y** with a value of 7. Write a program which tests whether **x** is *equal to* **y**, putting the result of this test into a **boolean** variable named **result**. Print your findings to the console, stating the following: “The result of whether 5 is equal to 7 is <result>”

```
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        boolean result;  
        int x = 5;  
        int y = 7;  
  
        result = x == y;  
        System.out.printf("The result of whether %d is equal to %d = %b\n",x,y,result);  
    }  
}
```

### Notes

Think about the statement **result = x==y;** in the following way:

**result = (is x equal to y, true or false?)**

Also note that **result = x==y;** can also be written as **result = (x==y);** that is, with brackets, if it makes the statement easier to understand.

5. I have an **int** **x** with a value of 5 and an **int** **y** with a value of 7. Write a program which tests whether **x** is *greater than* **y**, putting the result of this test into a **boolean** variable named **result**. Print your findings out to the console.

```
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        boolean result;  
        int x = 5;  
        int y = 7;  
  
        result = x > y;  
        System.out.printf("The result of whether %d is greater than %d = %b\n",x,y,result);  
    }  
}
```

6. I have an **int** **x** with a value of 5 and an **int** **y** with a value of 7. Write a program which tests whether **x** is *less than or equal to* **y**, putting the result of this test into a **boolean** variable named **result**. Print your findings out to the console.

```
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        boolean result;  
        int x = 5;  
        int y = 7;  
  
        result = x <= y;  
        System.out.printf("The result of whether %d is less than or equal to %d = %b\n",x,y,result);  
    }  
}
```

## Practice Your Java Level 1

	<pre> int y = 7; result = x &lt;= y; System.out.printf("The result of whether %d is less than or equal to %d = %b\n",x,y,result); } } </pre>
7.	I have an <code>int x</code> with a value of 5 and an <code>int y</code> with a value of 7. Write a program which tests whether $x$ is <i>greater than or equal to</i> $y$ , putting the result of this test into a <code>boolean</code> variable named <code>result</code> . Print your findings out to the console.
	<pre> public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x = 5;         int y = 7;         result = x &gt;= y;         System.out.printf("The result of whether %d is &gt;= %d = %b\n",x,y,result);     } } </pre>
8.	I have an <code>int x</code> with a value of 5 and an <code>int y</code> with a value of 7. Write a program which tests whether $x$ is <i>less than</i> $y$ , putting the boolean result of this test into a <code>boolean</code> variable named <code>result</code> . Print your findings out to the console.
	<pre> public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x = 5;         int y = 7;         result = x &lt; y;         System.out.printf("The result of whether %d is less than %d = %b\n",x,y,result);     } } </pre>
9.	I have an <code>int x</code> with a value of 5 and an <code>int y</code> with a value of 7. Write a program which tests whether $x$ is <i>not equal to</i> $y$ . Put the boolean result of this test into a <code>boolean</code> variable named <code>result</code> and print your findings out to the console.
	<pre> public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x = 5;         int y = 7;         result = x != y;         System.out.printf("The result of whether %d is not equal to %d = %b\n",x,y,result);     } } </pre>
10.	I have an <code>int x</code> with a value of 5 and an <code>int y</code> with a value of 7. Write a program which tests whether $x$ is <i>not greater than</i> $y$ . Put the boolean result of this test into a <code>boolean</code> named <code>result</code> and print your findings out to the console.
	<pre> public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x = 5;         int y = 7;         result = !(x &gt; y);         System.out.printf("The result of whether %d is less than or equal to %d = %b\n",x,y,result);     } } </pre>

	<p><b>Note:</b> There is no <code>not greater than</code> operator. Therefore putting the symbol for <code>not</code> (!) outside the bracket where the <code>&gt;</code> is used has the desired effect of <code>not greater than</code>.</p>
11.	<p>I have a <code>boolean x</code> with a value of <code>false</code> and a <code>boolean y</code> with a value of <code>true</code>. Write a program which tests whether <code>x is equal to y</code>. Put the boolean result of this test into a <code>boolean</code> variable named <code>result</code> and print your findings out to the console.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean x=false, y=true;         boolean result = x == y;         System.out.printf("The result of whether %b == %b is %b",x,y,result);     } }</pre>
12.	<p>Write a program which tests whether a given <code>int x</code> is greater than 5 and a given <code>int y</code> is greater than 7. Test the program with the following combinations of <code>x</code> and <code>y</code> respectively and print the result out after each test: (1,1), (6,7) and (15,15).</p> <p><b>Solution #1</b></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y;          x = 1; y = 1;    result = ((x &gt; 5) &amp;&amp; (y &gt; 7)); System.out.println(result);         x = 6; y = 7;    result = ((x &gt; 5) &amp;&amp; (y &gt; 7)); System.out.println(result);         x = 15; y = 15; result = ((x &gt; 5) &amp;&amp; (y &gt; 7)); System.out.println(result);     } }</pre> <p><b>Solution #2</b></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y;          x = 1; y = 1;    result = ((x &gt; 5) &amp; (y &gt; 7)); System.out.println(result);         x = 6; y = 7;    result = ((x &gt; 5) &amp; (y &gt; 7)); System.out.println(result);         x = 15; y = 15; result = ((x &gt; 5) &amp; (y &gt; 7)); System.out.println(result);      } }</pre>
13.	<p>What is the difference between the two solutions presented to the preceding exercise?</p> <p>The first solution is written using a “short-circuiting” logical operator, whereas the second one is not.</p>
14.	<p>What is the difference between short-circuiting logical operators and the regular logical operators?</p> <p>The short-circuiting logical operators stop testing as soon as they realize that the whole test can be deemed to have failed/passed as soon as the failure/passing of a sub-test of the test under assessment can be determined as causing the failure/passing of the whole test; therefore there would be no point in continuing/completing the evaluation. Using the first test in the solution to exercise 12 above as an example, there is no point in bothering to check whether <code>y &gt; 7</code> when you already know that <code>x</code> is less than 5 and that both parts of the test are required to be true. The non-short-circuiting logical operators however will evaluate the whole conditional statement instead of stopping at a failed/passed sub-test.</p> <p>The result of this difference is that the short-circuiting logical operator can potentially perform faster than the regular logical operators.</p>
15.	<p>Write a program which tests whether <code>x</code> is greater than 5 and <code>y</code> is less than 7. (Note: <code>x, y</code> and any other variables in this and subsequent exercises in this chapter are of type <code>int</code>; however you can use any mixture of numerical primitives that you prefer).</p> <p>Test the program with the following combinations of <code>x</code> and <code>y</code> respectively and print the result out after each test:</p>

## Practice Your Java Level 1

	(1,1), (6,5) and (4,5). <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x;         int y;         x = 1; y = 1; result = ((x &gt; 5) &amp;&amp; (y &lt; 7)); System.out.println(result);         x = 6; y = 5; result = ((x &gt; 5) &amp;&amp; (y &lt; 7)); System.out.println(result);         x = 4; y = 5; result = ((x &gt; 5) &amp;&amp; (y &lt; 7)); System.out.println(result);     } }</pre>
16.	Explain the difference between logical OR and XOR. Logical OR means that “at least one of the conditions being compared presented is true”. Logical XOR means “absolutely only one of the conditions being compared can be true”.
17.	Write a program which tests whether x is greater than 5 or y is less than 7. Test the program with the following combinations of x and y respectively and print the result out after each test: (1,1), (6,5) and (4,15). <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y;         x = 1; y = 1; result = ((x &gt; 5)    (y &lt; 7)); System.out.println(result);         x = 6; y = 5; result = ((x &gt; 5)    (y &lt; 7)); System.out.println(result);         x = 4; y = 15; result = ((x &gt; 5)    (y &lt; 7)); System.out.println(result);     } }</pre>
18.	Write a program which tests whether x is greater than or equal to 5 and y is equal to 0. Test the program with the following combinations of x and y respectively and print the result out after each test: (6,0), (5,0) and (4,1). <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y;         x = 6; y = 0; result = ((x &gt;= 5) &amp;&amp; (y == 0)); System.out.println(result);         x = 5; y = 0; result = ((x &gt;= 5) &amp;&amp; (y == 0)); System.out.println(result);         x = 4; y = 1; result = ((x &gt;= 5) &amp;&amp; (y == 0)); System.out.println(result);     } }</pre>
19.	Write a program which tests whether x is greater than or equal to 5 and y is <i>not</i> equal to 0. Test the program with the following combinations of x and y respectively and print the result out after each test: (6,0), (5,0), (4,1) and (7,2). <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y;         x = 6; y = 0; result = ((x &gt;= 5) &amp;&amp; (y != 0)); System.out.println(result);         x = 5; y = 0; result = ((x &gt;= 5) &amp;&amp; (y != 0)); System.out.println(result);         x = 4; y = 1; result = ((x &gt;= 5) &amp;&amp; (y != 0)); System.out.println(result);         x = 7; y = 2; result = ((x &gt;= 5) &amp;&amp; (y != 0)); System.out.println(result);     } }</pre>
20.	Write a program which tests whether <i>only one of</i> x or y is greater than 5. Test the program with the following combinations of x and y respectively and print the result out after each test: (5,5), (6,6), (3,3), (1,10) and (10,1).

	<p><i>Hint: xor (^)</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y;         x = 5; y = 5; result = ((x &gt; 5) ^ (y &gt; 5)); System.out.println(result);         x = 6; y = 6; result = ((x &gt; 5) ^ (y &gt; 5)); System.out.println(result);         x = 3; y = 3; result = ((x &gt; 5) ^ (y &gt; 5)); System.out.println(result);         x = 1; y = 10; result = ((x &gt; 5) ^ (y &gt; 5)); System.out.println(result);         x = 10; y = 1; result = ((x &gt; 5) ^ (y &gt; 5)); System.out.println(result);     } }</pre>
21.	<p>Write a program which tests whether <math>x</math> is greater than 5 or <math>y</math> is greater than 15 or <math>z</math> is less than or equal to 25. Test the program with the following combinations of <math>x</math>, <math>y</math> and <math>z</math> respectively and print the result out after each test: (5,16,25), (5,16,24), (5,15,24) and (4,5,30).</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y,z;         x=5; y=16; z=25; result = ((x &gt; 5)    (y &gt; 15)    (z &lt;= 25)); System.out.println(result);         x=5; y=16; z=24; result = ((x &gt; 5)    (y &gt; 15)    (z &lt;= 25)); System.out.println(result);         x=5; y=15; z=24; result = ((x &gt; 5)    (y &gt; 15)    (z &lt;= 25)); System.out.println(result);         x=4; y=5; z=30; result = ((x &gt; 5)    (y &gt; 10)    (z &lt;= 25)); System.out.println(result);     } }</pre>
22.	<p>Write a program which tests whether <math>x</math> is greater than or equal to 5 and either of <math>y</math> or <math>z</math> are less than 15. Test the program with the following combinations of <math>x</math>, <math>y</math> and <math>z</math> respectively and print the result out after each test: (5,14,20), (5,15,13), (5,10,10) and (4,5,30).</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y,z;         x=5; y=14; z=20; result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=5; y=15; z=13; result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=5; y=10; z=10; result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=4; y=5; z=30; result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15)    (z &lt; 15))); System.out.println(result);     } }</pre> <p>If desired, the individual sub-conditions can be broken down in the following manner:</p> <pre>boolean subResult01 = x &gt;= 5; boolean subResult02 = ((y &lt; 15)    (z &lt; 15)); boolean result = (subResult01 &amp;&amp; subResult02);</pre>
23.	<p>Write a program which tests whether <math>x</math> is greater than or equal to 5 or either <math>y</math> or <math>z</math> are less than 15. Test the program with the following combinations of <math>x</math>, <math>y</math> and <math>z</math> respectively and print the result out after each test: (5,14,20), (5,15,13), (5,10,10), (4,5,30), (1,20,7) and (1,20,20).</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y,z;         x=5;y=14;z=20; result = ((x &gt;= 5)    ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=5;y=15;z=13; result = ((x &gt;= 5)    ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=5;y=10;z=10; result = ((x &gt;= 5)    ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=4;y=5;z=30; result = ((x &gt;= 5)    ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=1;y=20;z=7; result = ((x &gt;= 5)    ((y &lt; 15)    (z &lt; 15))); System.out.println(result);         x=1;y=20;z=20; result = ((x &gt;= 5)    ((y &lt; 15)    (z &lt; 15))); System.out.println(result);     } }</pre>

## Practice Your Java Level 1

	}
24.	<p>Write a program which tests whether <math>x</math> is greater than or equal to 5 or only one of <math>y</math> or <math>z</math> is less than 15. Test the program with the following combinations of <math>x</math>, <math>y</math> and <math>z</math> respectively and print the result out after each test: (5,14,14), (5,20,13), (6,5,30) and (6,20,20).</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y,z;          x=5;y=14;z=14;  result = ((x &gt;= 5)    ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);         x=5;y=20;z=13;  result = ((x &gt;= 5)    ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);         x=6;y=5;z=30;   result = ((x &gt;= 5)    ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);         x=6;y=20;z=20;  result = ((x &gt;= 5)    ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);     } }</pre>
25.	<p>Write a program which tests whether <math>x</math> is greater than or equal to 5 and only one of <math>y</math> or <math>z</math> is less than 15. Test the program with the following combinations of <math>x</math>, <math>y</math> and <math>z</math> respectively and print the result out after each test: (5,14,14), (5,15,13) and (5,10,10).</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean result;         int x,y,z;          x=5;y=14;z=14;  result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);         x=5;y=15;z=13;  result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);         x=5;y=10;z=10;  result = ((x &gt;= 5) &amp;&amp; ((y &lt; 15) ^ (z &lt; 15))); System.out.println(result);     } }</pre>
26.	<p>Given the following values and boolean statements, state at which point the value of the result is determined and the processing moves on to the next command.</p> <ol style="list-style-type: none"> <li>1. <math>x=7;y=14;z=14;</math>      <math>result = ((x &gt;= 5)    ((y &lt; 15) ^ (z &lt; 15)))</math>;</li> <li>2. <math>x=5;y=20;z=13;</math>      <math>result = ((x &gt;= 5)   ((y &lt; 15) ^ (z &lt; 15)))</math>;</li> <li>3. <math>x=6; y=0;</math>      <math>result = ((x &gt;= 5) \&amp;\&amp; (y == 0))</math>;</li> <li>4. <math>x=4; y=0;</math>      <math>result = ((x &gt;= 5) \&amp;\&amp; (y == 0))</math>;</li> <li>5. <math>x=4; y=0;</math>      <math>result = ((x &gt;= 5) \&amp; (y == 0))</math>;</li> </ol> <p>1. The processing can stop right after determining that <math>x &gt;= 5</math> is true. The reason for this is because the statement to be evaluated has two parts; and given that they are bound by an OR statement, as soon as one part of it whole statement is true, then the overall statement is true. Therefore, because there is a short-circuiting OR operator between both parts, the evaluation can stop as early as possible.</p> <p>2. The processing will evaluate right to the end of the statement. The reason is because we use a regular OR operator, rather than a short-circuiting one.</p> <p>3. The statement must be fully evaluated, because even though it uses a short-circuiting boolean operator, the fact that it is an AND operator mandates that all parts have to be true, therefore all parts have to be evaluated until either of 1) a subpart being evaluated is determined to be false or 2) the end of the statement.</p> <p>4. Evaluation is completed as soon as it is determined that <math>x</math> is less than 5, given that we have a short-circuiting operator.</p> <p>5. The whole statement is evaluated, even after it is determined that the first part has failed (and thus in this case the whole statement has failed) because we are not using a short-circuiting operator.</p>

In the chapter on *Conditional Processing*, we will use the return values from boolean operations to make decisions on which parts (branches) of our code to run.

# Chapter 5. Strings: Basic Handling

The objective of this chapter is to ensure that you are conversant with string processing in Java, given the fact that string manipulation is a key feature of any programming language. Among the string exercises presented in this chapter are exercises on string variable declaration, concatenation, case modification, equality testing, substring manipulation, as well as the formatting of strings for output. Your understanding of the immutability of `String` objects is also exercised.

The key class on which exercises are presented in this chapter is the class `String`.

**Note:** Due to the intertwining of the topic of string handling with other topics yet to be discussed, we will deal with strings as far as possible in this chapter and address other aspects of strings again in the chapters on the `StringBuffer` class, the `Char` type and Arrays and also to different degrees in other chapters as appropriate.

1.	Write a program which declares a <code>String</code> variable named <code>firstName</code> and initializes its value to the word <code>'John'</code> using the operator <code>new</code> .	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String firstName = new String("John");     } }</pre>
2.	Repeat the same exercise as above, initializing the variable simply by assigning the word <code>'John'</code> to the <code>String</code> variable using the assignment operator.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String firstName = "John";     } }</pre>
3.	( <i>string concatenation</i> ) I have the <code>String</code> variables <code>s1</code> and <code>s2</code> with values as indicated:  <code>s1 = "John"; s2 = "Leavings";</code>  Using the addition (+) operator, put into a third <code>String</code> variable <code>s3</code> the concatenation of these two strings, ensuring that you put a space in-between them. Print <code>s3</code> out to the console.	<pre>public class PracticeYourJava {     public static void main(String[] args) {          String s1 = "John";         String s2 = "Leavings";         String s3 = s1 + " " + s2;         System.out.println(s3);     } }</pre>
4.	Repeat the preceding exercise, using the <code>concat</code> method of the class <code>String</code> .	<pre>public class PracticeYourJava {     public static void main(String[] args) {          String s1 = "John";         String s2 = "Leavings";         String space = " ";          String s3 = s1;         s3 = s3.concat(space);         s3 = s3.concat(s2);         System.out.println(s3);     } }</pre>
5.	Determine and print out the length of the string contained in the <code>String</code> variable <code>firstName</code> of exercise 2 above. <i>Hint: Use the <code>Length</code> method of the <code>String</code> class.</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          String firstName = "John";         int stringLength = 0; // We will use this variable for the length.     } }</pre>

## Practice Your Java Level 1

	<pre>         stringLength = firstName.length();         System.out.printf("The length of the first name is %s\n", stringLength);     } }  <b>Note:</b> We could, without having declared the variable <code>stringLength</code> have printed the length of the string directly in the <code>System.out.printf</code> statement as shown below: System.out.printf("The length of the first name is %s\n", firstName.length()); </pre>
6.	I have the following string literal, "Basketball!". Print out its length, without declaring any variables. <pre> public class PracticeYourJava {     public static void main(String[] args) {         System.out.printf("The length is %d.\n", "Basketball!".length());     } } </pre>
7.	Print out the value of the variable <code>firstName</code> from exercise 2 in lowercase. <i>Hint: Use the <code>String</code> method <code>toLowerCase()</code>.</i> <pre> public class PracticeYourJava {     public static void main(String[] args) {          String firstName = "John";         String firstNameLowerCase = firstName.toLowerCase();         System.out.printf("firstName as all uppercase is %s\n", firstNameLowerCase);     } } </pre> <p><b>OR</b></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          String firstName = "John";         System.out.printf("firstName as all lowercase is %s\n", firstName.toLowerCase());         //We just executed the desired method within the parameter field of the method         //System.out.printf.     } } </pre>
8.	Print out the value of the variable <code>firstName</code> from the preceding exercise in uppercase. <pre> public class PracticeYourJava {     public static void main(String[] args) {          String firstName = "John";         System.out.printf("firstName as all uppercase is %s\n", firstName.toUpperCase());     } } </pre>
9.	What does the statement " <i>a String is immutable</i> " mean? <p>Before we answer the question, it must first be understood that for variables that pertain to objects, what is contained in the variable is not directly the object itself, but rather a value stating the location in memory where the object that it references is located. Therefore, when you access an object variable, in order to get the object referred to, the Java run-time actually performs two steps; first it looks up what memory address the variable is pointing to and then secondly goes to that memory location to get the contents thereof. Now to the explanation of <code>String</code> immutability.</p> <p>What the statement <i>a String is immutable</i> means is that once a value is assigned to a <code>String</code> object, Java does not actually allow the value contained in the <code>String</code> object to be changed. However, this bears explanation since, in a sense, we can "change" the contents of a <code>String</code> as shown below:</p> <pre> String s1 = "Hello. "; s1 = s1 + " How are you?"; </pre> <p>When we add the string "How are you?" to the string in the object reference <code>s1</code>, what really happens is that under the covers Java creates a brand new string "Hello. How are you?" somewhere else in memory and in the</p>

	<p>background assigns a reference to the location of this new string to the variable <code>s1</code>!</p> <p>Why is this knowledge of the immutability of objects of the <code>String</code> class important to know? It is important to know for a number of reasons, including (1) the fact that modifying a <code>String</code> (which we now know causes a new string to be created, and the variable holding a reference to it to be updated to the location of the new string) is relatively expensive timewise. Therefore, for situations where we have to manipulate a string quite often in a program, it is best that the class <code>StringBuilder</code> which is mutable be used rather than the class <code>String</code> and (2) if the particular memory reference to which a given <code>String</code> variable pertains is important to you then you should know that it will be changed when you modify the contents of the given <code>String</code> variable.</p> <p>Really, strings can be referred to as <i>immutable reference</i> types. They have unique behavior, in particular the following:</p> <ol style="list-style-type: none"> <li>(Previously stated) They cannot be changed once created. If you modify a <code>String</code> (appending, trimming, etc), a new string is created and presented to the program in which it was changed, <i>with the same variable name</i>, giving you the appearance that the contents of the <code>String</code> variable in hand were changed (As stated earlier, in a sense it is changed by Java creating a new string and making the variable <code>s1</code> contain a reference to the location in memory of the new string). For many programs, this fact has no negative impact on the construction of the program as this throwing away and re-creation is hidden from the programmer who requested the modification.</li> <li>Even though they are reference types, due to their immutability, if you pass a <code>String</code> to another method which then tries to modify its contents, unlike other reference types, the contents of the original <code>String</code> in the calling method will not be modified.</li> </ol> <p><b>Note:</b> Despite the extensive explanation above, for general intents and purposes, you can program with the mentality that you <i>can</i> change the contents of a <code>String</code> in a method, however you must be aware of what is going on in the background.</p>
10.	<p>I have a string object reference <code>s1</code>, initialized as follows: <code>String s1 = null;</code> What does this initialization to the value <code>null</code> mean?</p> <p>The initialization to the value <code>null</code> is saying in effect that “this <code>String</code> object reference <code>s1</code> does not contain a reference to any object”.</p> <p>The value <code>null</code> does not only apply to <code>String</code> variables; it can be used for any object reference variable. When any object reference has the value of <code>null</code>, it is saying that the object reference variable in question does not contain a reference to any object.</p>
11.	<p>What is the difference between the following two <code>String</code> objects?</p> <pre>String s1 = null; String s2 = "";</pre> <p>The <code>String</code> variable <code>s1</code> does not point to any object at all, whereas <code>s2</code> points to an empty string. An empty string is still regarded as a <code>String</code> object, whereas <code>null</code> really means nothing; not even empty.</p> <p>If we run the following code:</p> <pre>System.out.println(s1.length());</pre> <p>It will result in an exception, <code>java.lang.NullPointerException</code>, for the action is on an unassigned variable.</p> <p>However,</p> <pre>System.out.println(s2.length());</pre> <p>will result in the value <code>0</code> being output.</p>
<b>String Equality</b>	
12.	<p>Given two strings <code>s1</code> &amp; <code>s2</code>, using the <code>String.equals</code> <i>instance method</i>, write a line of code to test whether <code>s1</code> &amp; <code>s2</code> are equal and put the result into a <code>boolean</code> variable <code>x</code>.</p> <pre>boolean x = s1.equals(s2)); OR boolean x = s2.equals(s1));</pre>
<b>Substrings</b>	
13.	<p>What is the index of the first character in a <code>String</code> in Java?</p> <p>The index of the first character in a Java <code>String</code> object is 0.</p> <p>Therefore if for example we have 10 characters in a string, the indexes of the individual characters therein from the 1<sup>st</sup> to the 10<sup>th</sup> will be from 0 to 9 in order.</p>
14.	<p>Given the string ‘Mary had a little’ The character is at position 5.</p>

## Practice Your Java Level 1

	<p><i>lamb</i>", manually determine the position of the character 'h' in the string.</p>	<p>This is because character position enumeration in Java for strings starts from 0. Therefore the positions in this string would be;</p> <table style="margin-left: 20px;"> <tr><td>M</td><td>0</td></tr> <tr><td>a</td><td>1</td></tr> <tr><td>r</td><td>2</td></tr> <tr><td>y</td><td>3</td></tr> <tr><td></td><td>4</td></tr> <tr><td><b>h</b></td><td><b>5</b></td></tr> <tr><td>a</td><td>6</td></tr> <tr><td>d</td><td>7</td></tr> </table>	M	0	a	1	r	2	y	3		4	<b>h</b>	<b>5</b>	a	6	d	7
M	0																	
a	1																	
r	2																	
y	3																	
	4																	
<b>h</b>	<b>5</b>																	
a	6																	
d	7																	
15.	<p>Given the string "<i>Mary had a little lamb</i>", write a program to determine the position of the letter 'i' in this string. Print this position out to the console stating: <i>The position of letter 'i' is : &lt;position&gt;</i>.</p> <p><i>Hint: String method &lt;String&gt;.indexOf</i></p>	<p style="text-align: center;"><b>Solution #1</b></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb";         int letterPosition = rhymeLine.indexOf('i'); // note single quotes for a single letter         System.out.printf("The position of the letter \'i\' is: %d\n.", letterPosition);     } }</pre> <p><b>Note:</b> a single character literal is referred to using single quotes, not double quotes, as seen in this example when we look for the index of 'i' using the <code>indexOf</code> method.</p> <p style="text-align: center;"><b>Solution #2</b></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb";         System.out.printf("The position of the letter \"i\" is %.d.\n", rhymeLine.indexOf('i'));     } }</pre> <p><b>Note:</b> In this variant of the answer, we simply call the <code>indexOf</code> method directly within the parameter section of the <code>System.out.printf</code> method.</p> <p style="text-align: center;"><b>Solution #3</b></p> <pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         int letterPosition = "Mary had a little lamb".indexOf('i');         System.out.printf("The position of the letter \"i\" is %.d.\n", letterPosition);     } }</pre> <p><b>Note:</b> Observe carefully how in this solution we show how a <code>String</code> method can be applied <i>directly to the literal string without declaring a variable to hold the string itself</i>.</p> <p style="text-align: center;"><b>Solution #4</b></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.printf("The position of the letter \"i\" is %.d.\n",                           "Mary had a little lamb".indexOf('i'));     } }</pre> <p><b>Note(s):</b> In this variant of the answer, note how no variables are declared at all; we just apply the <code>indexOf</code> method directly to the string in the parameter section of the <code>System.out.printf</code> method.</p>																
16.	<p>Given the string "<i>Mary had a little lamb</i>", what is the position of the 2<sup>nd</sup> letter 'a' in this string?</p> <p><i>Hint: the overloaded String.indexOf method.</i></p>																	

	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb";         int positionOfFirstOccurrence = rhymeLine.indexOf('a');         // Now, that we have the position of the 1st letter 'a' we start looking for         // the very next letter 'a' after that.         int positionOfSecondOccurrence = rhymeLine.indexOf('a',(positionOfFirstOccurrence + 1));         System.out.printf("The position of the 2nd letter \"a\" is %d.\n", positionOfSecondOccurrence);     } }</pre>
17.	<p>Given the string ‘Mary had a little lamb, little lamb, little lamb, Mary had a little lamb that was as white as snow’, write a program which will determine and output to the console the position of the 2<sup>nd</sup> occurrence of the word “little” in the string.</p> <p><i>Hint: determine the position of the end of the 1<sup>st</sup> occurrence and then start searching after that.</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb, little lamb, little lamb, Mary had a little lamb that was as white as snow";         // So what we have to do is first look for the 1st occurrence of the word "little",         // and then start searching after that 1st occurrence.         int iFirstAppearancePosition = rhymeLine.indexOf("little");         int iSearchStartPosition = iFirstAppearancePosition + "little".length();         // See that we start searching at the end of the 1st instance of the word "little"         int iSecondAppearancePosition = rhymeLine.indexOf("little", iSearchStartPosition);         System.out.printf("The 2nd occurrence of \"little\" is at position: %d.\n", iSecondAppearancePosition);     } }</pre>
18.	<p>Given the string ‘Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow’, write a program to determine and print out the position of the <i>last</i> occurrence of the word “little” in the string.</p> <p><i>Hint: String.lastIndexOf</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow";         int wordPosition = rhymeLine.lastIndexOf("little");         System.out.printf("The position of the word \"little\" is %d.\n", wordPosition);     } }</pre>
19.	<p>Given the string ‘Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow’, write a program to determine and print out the position of the <i>last</i> occurrence of the letter ‘w’.</p> <p><i>Hint: String method lastIndexOf(char)</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb,\n little lamb,\n little lamb,\n Mary had a little lamb that was as white as snow";         int letterPosition = rhymeLine.lastIndexOf('w');         System.out.printf("The last position of the letter 'w' is %d.\n", letterPosition);     } }</pre>
20.	<p>Given a Java <i>String</i> that contains the rhyme “Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little lamb that was as white as snow”, write code to replace the word “little” with the phrase “big big” (to keep the rhyme ☺) everywhere the word “little” appears. Print out the modified string.</p> <p><i>Hint: String method replace</i></p>

## Practice Your Java Level 1

	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little                            lamb that was as white as snow";         String newRhymeLine = rhymeLine.replace("little", "big big");         System.out.printf("The new rhyme is:\n%s\n", newRhymeLine);     } }</pre>
21.	<p>Given the rhyme “<i>Mary had a little lamb,\\n little lamb,\\n little lamb,\\n Mary had a little lamb that was as white as snow</i>”, write a program which <i>removes</i> the word “<i>little</i>” wherever it appears in the rhyme. Print out the resulting string.</p> <p><i>Hint: replace the word with an empty string</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little                            lamb that was as white as snow";         String newRhymeLine = rhymeLine.replace("little", "");         // "little" is replaced with a blank String. The double-quotes with nothing         // in-between is a blank String.         System.out.printf("The new rhyme is:\n%s\n", newRhymeLine);     } }</pre>
22.	<p>We have a phone number given as the string <b>111-222-3333</b>, of which the first 3 digits are the area code. Extract and print out the area code.</p> <p><i>Hint: String.substring</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String phoneNumber = "111-222-3333";         int startPosition = 0; // We are starting at the beginning of the String         int numberOfCharacters = 3;         String areaCode = phoneNumber.substring(startPosition, (startPosition + numberOfCharacters));         System.out.printf("The area code is :%s.\n", areaCode);     } }</pre>
23.	<p>We have a phone number given as the string <b>111-222-3333</b>, of which the last 7 digits are the subscriber number. Extract and print out the subscriber number, that is, the substring <b>222-3333</b>.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String phoneNumber = "111-222-3333";         int startPosition = 4;         // We are starting after the first dash. Remember start counting at 0.         String SubscriberNum = phoneNumber.substring(startPosition);         // Extracts to the end of the String         System.out.printf("The subscriber number is : %s\n ", SubscriberNum);     } }</pre>
24.	<p>We have a phone number, <b>111-222-3333</b>, of which the first 3 characters are the area code. The next 3 numbers are the “Central Office number”. Extract this number and print it out to the console.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String phoneNumber = "111-222-3333";         int startPosition = 4; // We are starting after the first dash         int NumberOfCharsToExtract = 3;         String CONumberStr = phoneNumber.substring(4, startPosition+NumberOfCharsToExtract);         // Extracts to the end of the String         System.out.printf("The Central Office number is : %s\n", CONumberStr);     } }</pre>

	<pre>}</pre> <p><b>Note:</b> If desired you can write this in a more generic way; determine the position of the first and second dashes and extract what is between them.</p>
25.	<p>We have a phone number given as the string 111-222-3333, of which the first 3 characters are the area code. Find and print out the <i>last</i> position of the number 1 within the area code.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String phoneNumber = "111-222-3333";         int lastSearchPosition = 2;         int foundPosition = phoneNumber.lastIndexOf('1', lastSearchPosition);         System.out.printf("Last position of the number 1 in the area code is : %d\n", foundPosition);     } }</pre>
26.	<p>I have the following string:  <code>"      how are you?      "</code>          Write a program which gets rid of both the leading and trailing spaces and print the result out. Confirm that you have erased the trailing spaces by printing out the letter 'X' right behind the trimmed string.  <i>Hint: String.trim</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String s1 = "      how are you?      ";         String ending = "X";         String s2 = s1.trim();         System.out.printf("%s%s\n", s2, ending);     } }</pre>
27.	<p>We have the following string:  <code>"&gt;&gt;&gt;&gt;&gt;How are you&lt;&lt;&lt;&lt;&lt;"</code>          Using the <code>replace</code> method of class <code>String</code>, erase the extraneous characters bounding the phrase "How are you?". Print the trimmed string out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String s1 = "&gt;&gt;&gt;&gt;&gt;how are you?&lt;&lt;&lt;&lt;&lt;";         String s2 = s1.replace("&gt;", "");         s2 = s2.replace("&lt;", "");         System.out.println(s2);     } }</pre>
28.	<p>The string 'Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little lamb that was as white as snow" prints out as:</p> <p><i>Mary had a little lamb, little lamb, little lamb, Mary had a little lamb that was as white as snow</i></p> <p>Write a program which determines the position relative to the start of the 2<sup>nd</sup> line of the rhyme of the word "lamb".</p> <p><i>Code logic</i>  <i>(This is one possible solution) Extract the 2<sup>nd</sup> line (the \n is the delimiting point) and then look for the word "lamb" in the extracted line.</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String rhymeLine = "Mary had a little lamb,\nlittle lamb,\nlittle lamb,\nMary had a little lamb that was as white as snow";         int startPos = rhymeLine.indexOf('\n') + 1;         String rhymeFrom2ndLine = rhymeLine.substring(startPos);          int endPos = rhymeFrom2ndLine.indexOf('\n');         String rhyme2ndLine = rhymeFrom2ndLine.substring(0, (endPos+1));          // We have now extracted the 2nd line into the variable rhymeFrom2ndLine     } }</pre>

## Practice Your Java Level 1

```

        int wordPos = rhyme2ndLine.indexOf("lamb");
        System.out.printf("The 1st position of the word \"lamb\" in line 2 = %d\n", wordPos);
    }
}

```

### Formatting String Output

	<pre> int wordPos = rhyme2ndLine.indexOf("lamb"); System.out.printf("The 1st position of the word \"lamb\" in line 2 = %d\n", wordPos); } </pre>
29.	I have the following <code>String</code> variables; <code>s1="Mary"</code> , <code>s2="had"</code> , <code>s3="a"</code> , <code>s4="little"</code> and <code>s5="lamb"</code> . Print each of them out, right-justified, in a field that is of width 10 characters.
	<pre> public class PracticeYourJava{     public static void main(String[] args){         String s1 = "Mary", s2 = "had", s3 = "a", s4 = "little", s5 = "lamb";         System.out.printf("%10s\n", s1);         System.out.printf("%10s\n", s2);         System.out.printf("%10s\n", s3);         System.out.printf("%10s\n", s4);         System.out.printf("%10s\n", s5);     } } </pre>
30.	Repeat the preceding exercise, however with the output <i>left-justified</i> . Also, bound each individual justified string with the character ' <code> </code> ' on both the left & right.
	<pre> public class PracticeYourJava{     public static void main(String[] args){         String s1 = "Mary", s2 = "had", s3 = "a", s4 = "little", s5 = "lamb";         System.out.printf(" %-10s \n", s1);         System.out.printf(" %-10s \n", s2);         System.out.printf(" %-10s \n", s3);         System.out.printf(" %-10s \n", s4);         System.out.printf(" %-10s \n", s5);     } } </pre>
31.	<p>What is the use of the <code>String.format</code> method?</p> <p>The <code>String.format</code> method is a method whose behavior is essentially like the <code>System.out.println/print</code> methods, however, its output is put into a <code>String</code>, rather than written to the console.</p> <p>This method can be useful in scenarios where at given points in your code you produce intermediate output and prefer to append the intermediate output to a string with the intent of putting the final string out to the console or to a file later in your processing.</p>
32.	<p>Rewrite exercise 30, this time, putting all the intermediate output into a <code>String</code> variable <code>s</code>. Print <code>s</code> out at the end of all the processing.</p> <p><i>Hint:</i> <code>String.format</code></p> <pre> public class PracticeYourJava{     public static void main(String[] args){         String s1 = "Mary", s2 = "had", s3 = "a", s4 = "little", s5 = "lamb";         String s;         s = String.format(" %-10s \n", s1);         s = s + String.format(" %-10s \n", s2);         s = s + String.format(" %-10s \n", s3);         s = s + String.format(" %-10s \n", s4);         s = s + String.format(" %-10s \n", s5);         System.out.println(s);     } } </pre>

# Chapter 6. Conditional Processing

A key part of any computer language is the ability to direct the execution (*or not*) of specific sections/blocks of code based on the result of boolean conditions. This chapter exercises your skill in using conditional processing to direct the execution of specific blocks of code by using the conditional processing statements **if**, **if/else**, **else** as well as the **switch**, **case**, **break** and **default** statements.

Conditional Operators: if and if/else	
1.	<p>I have two integer variables, <b>x</b> and <b>y</b>, with values 5 and 7 respectively. Compare the two values and on comparison, print a statement that states which is greater. Use only <b>if</b> conditions to do the comparisons.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         boolean b01;         boolean b02;         int x = 5;         int y = 7;          b01 = x &gt; y;         b02 = x &lt; y;          if (b01 == true)         {             System.out.printf("%d is greater                 than %d", x, y);         }         if (b02 == true)         {             System.out.printf("%d is greater                 than %d", x, y);         }     } }</pre> <p style="text-align: center;"><b>OR</b></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int x = 5;         int y = 7;          if (x &gt; y)         {             System.out.printf("%d is greater than                 %d", x, y);         }         if (y &gt; x)         {             System.out.printf("%d is greater than                 %d", x, y);         }     } }</pre>
	<p><b>Notes:</b> On comparing the two solutions, it should be observed that the <b>boolean</b> variables <b>b01</b> and <b>b02</b> are not required, as the statement <b>(x &gt; y)</b> or the statement <b>(y &gt; x)</b> is a self-contained statement that returns a boolean result of <b>true</b> or <b>false</b>. The statement <b>if(&lt;whatever&gt;)</b> really means “<b>if(&lt;whatever is in this bracket is true&gt;)</b>”, then run the commands in the braces following the <b>if</b>”.</p> <p>2. I have two floating point variables, <b>x</b> and <b>y</b>, with values 5.77 and 7.83 respectively. Compare the two values and if they are equal state so, otherwise state that they are not equal. Use an <b>if/else</b> construct.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {          float x = 5.77f;         float y = 7.83f;          if (x == y) {             System.out.println("The numbers are equal");         }         else{             System.out.println("The numbers are not equal");         }     } }</pre> <p><b>Explanation:</b> the <b>else</b> condition can be described as follows: Whatever is false about the <b>if</b> condition is handled by the <b>else</b> condition. Recall from the earlier example that <b>if(&lt;whatever&gt;)</b> means <b>if(&lt;whatever is in this bracket is true&gt;)</b>. The <b>else</b> then handles whatever failed</p>

## Practice Your Java Level 1

	the <b>if</b> question.
3.	I have two integer variables, <i>x</i> and <i>y</i> . Write a program which compares them and on comparison prints which is greater, otherwise if they are equal then the program should state that they are equal. Hardcode the values 5 and 7 for <i>x</i> and <i>y</i> to test your program.
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int x = 5;         int y = 7;          if (x &gt; y) {             System.out.printf("%d is &gt; %d", x, y);    OR         }         if (y &gt; x){             System.out.printf("%d is &gt; %d", y, x);         }         if (x == y){             System.out.printf("%d = %d", x, y);         }     } }</pre> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int x = 5;         int y = 7;          if (x &gt; y){             System.out.printf("%d is &gt; %d", x, y);         }         else if (y &gt; x){             System.out.printf("%d is &gt; %d", y, x);         }         else if (x == y){             System.out.printf("%d = %d", x, y);         }     } }</pre>
4.	What is the difference between the two solutions shown for the preceding exercise? The 1 <sup>st</sup> solution goes through and evaluates each of the conditions in the respective <b>if(&lt;condition&gt;)</b> statements. However, in this exercise where each of the conditions is mutually exclusive, there is no real need to test subsequent related conditions if a preceding condition has been evaluated to be true (for example in this particular exercise since we already know that <i>x</i> > <i>y</i> then there is no point in checking whether <i>y</i> > <i>x</i> , or whether <i>x</i> == <i>y</i> ). Therefore, the 2 <sup>nd</sup> solution which uses the <b>else if</b> conditions which are not evaluated if a preceding condition is true is more efficient.
5.	Write a program, which based on a value given for the temperature gives the following output: ol style="list-style-type: none;"> li>1. If the temperature < 60 then output “ <i>Cold enough to wear a coat</i> ” <li>2. If the temperature <math>\geq 60</math> but &lt; 68 then output “<i>Cold enough to wear a jacket</i>”</li> <li>3. Otherwise simply output “<i>No outerwear required!</i>”</li> <p>For testing purposes, hardcode the temperature value into the program.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         float temperature = 0f;         // Set it to an initial value for our code. We can change it and recompile         // in order to test the code for different temperatures.          if (temperature &lt; 60) {             System.out.println("Cold enough to wear a coat");         }         else if (temperature &lt; 68) {             //We've already handled whatever is less than 60             //so this is implicitly 60 to just under 68             System.out.println("Cold enough to wear a jacket");         }         else {             System.out.println("No outerwear required!");         }     } }</pre>

Combining multiple conditions		
6.	Given two numerical variables <code>x</code> and <code>y</code> , explain what the following code means:  <pre>if ((x &gt; 5) &amp;&amp; (y &gt; 20)) {     // Perform action... }</pre>	The code means, if <code>x</code> is greater than 5 <b>AND</b> <code>y</code> is greater than 20 then perform the actions in the braces.
7.	Given two numerical variables <code>x</code> and <code>y</code> , explain what the following code means:  <pre>if ((x &gt; 5)    (y &lt; 20)) {     // Perform action... }</pre>	The code means, if <code>x</code> is greater than 5 <b>OR</b> <code>y</code> is less than 20 then perform the actions in the braces.  <b>Note:</b> Of course you know that logical OR means “at least one of the conditions is valid”.
8.	Given three numerical variables <code>x</code> , <code>y</code> and <code>z</code> , give an explanation of what the following code means:  <pre>if ((x &gt; 5)    (y &gt; 20)    (z == 0)) {     // Perform action... }</pre>	The code means, if any of <code>x</code> is greater than 5 <b>OR</b> <code>y</code> is greater than 20 <b>OR</b> <code>z</code> equals 0 then perform the actions in the braces.  <b>Note:</b> see the note in the preceding solution.
9.	Given three integers <code>x</code> , <code>y</code> and <code>z</code> , give an explanation of what the following code means (pay close attention to the placement of the braces):  <pre>if ((x &gt; 5) &amp;&amp; ((y &gt; 20) ^ (z == 0))) {     // Perform action... }</pre>	The code is first evaluating two different sets of conditions separately, namely:  $(x > 5)$ <b>AND</b> $((y > 20) ^ (z == 0))$  The <code>&amp;&amp;</code> condition is stating that “if both of the separate conditions are true”, then perform the actions in the braces.  <b>Note:</b> Recall that <code>^</code> (XOR) means that only one of the conditions can be true. Contrast this with the functioning of OR (see the preceding exercise).
10.	Rewrite this code fragment using the <u>ternary operator</u> :  <pre>int x = 50, y = 0; if (x &gt; 10)     y = 5; else     y = 27; System.out.println(y);</pre> <pre>int x = 50, y = 0; y = x &gt; 10 ? 5 : 27; System.out.println(y);</pre>	<b>Note:</b> You can think about the ternary operator in the following way: <code>value = question(the result is either true or false)? value if true: value if false;</code>

The <code>switch</code> and <code>case</code> statements		
11.	Explain your understanding of how the <code>switch</code> statement works. Give an example that uses a <code>switch</code> statement.  A <code>switch</code> statement is a conditional processing construct which, based on the value of a variable under assessment, will execute specific blocks of code called <i>cases</i> according to the value of the variable. A <code>switch</code> construct uses the keywords <code>switch</code> , <code>case</code> , <code>break</code> and <code>default</code> to determine, bound and exit particular blocks of code within the switch construct. Each <i>case</i> is bounded by the keywords <code>case</code> and <code>break</code> . Any condition for which we do not have a <code>case/break</code> block can be handled by a <code>default</code> block at the end of the <code>switch</code> block.  For example, see the following code block:	

## Practice Your Java Level 1

```
int zz=10;
switch(zz)
{
    case 5:
        System.out.println("zz is equal to 5");
        break; // means end of this particular block. You exit the case (& thus the switch block)
                // at break statements.
    case 6:
    case 7:
        System.out.println("zz is equal to 6 or 7");
        // Observe that we can combine multiple cases in one block.
        break;
    case 8:case 9:case 10:
        System.out.println("zz is equal to 8, 9 or 10!");
        break;
    default:
        System.out.println("Default covers anything not covered by a case!");
}
```

12. Explain the output of the following switch block, given that `zz=5` and explain why it is so.

```
int zz=5;
switch(zz)
{
    case 5:
        System.out.println("zz is equal to 5");
    case 6:
    case 7:
        System.out.println("zz is equal to 6 or 7");
        // See that we can combine multiple cases in one block.
        break;
    default:
        System.out.println("Default covers anything not covered by a case!");
}
```

**Output:**

```
zz is equal to 5
zz is equal to 6 or 7
```

**Reason:** There was no `break` statement at the end of case 5; when no such statement appears, the switch simply keeps processing within the `switch` block until either it sees either a `break`, a `default` statement or the end of the block!

Sometimes this effect is intended, but in the case that it isn't we must be careful to put the `break` statement in.

13. Write a program, using a `switch` block to print out how many days any given month (specified as a string) has.  
Hardcode values with which to test your program.

```
public class PracticeYourJava {
    public static void main(String[] args) {

        String month = "jan";
        month = month.toLowerCase(); // For uniform processing
        month = month.substring(0, 3); // 3-leftmost characters
        switch(month) {
            case "jan": case "mar": case "may": case "jul": case "aug": case "oct": case "dec":
                System.out.printf("This month has 31 days");
                break;
            case "apr": case "jun": case "sep": case "nov":
                System.out.println("This month has 30 days");
                break;
            case "feb":
                System.out.println("This month has 28 days");
                break;
            default:
                System.out.println("Invalid month. Enter at least the first 3 letters of a month");
        }
    }
}
```

	}
14.	<p>At our hotel, we have a reward card for frequent guests. There are three card levels, namely gold, silver and bronze, with bronze being the lowest level.</p> <p>The benefits of each of the levels is as follows:</p> <ul style="list-style-type: none"> <li>• bronze = free parking + newspaper</li> <li>• silver = bronze level features + breakfast</li> <li>• gold = silver level features + dinner for one</li> </ul> <p>If you do not have a reward card, then by default all you get is a room.</p> <p>Write a program which on assessing a <code>String</code> variable that contains the card level, prints out, using a <code>switch</code> block, all the benefits of each level. Ensure that you do not replicate code.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {          String customerLevel = "bronze"; //hardcoded test value         customerLevel = customerLevel.toLowerCase(); //for uniformity in processing          System.out.println("Your benefits are:");         switch (customerLevel)         {             case "gold":                 System.out.println("\t included dinner for 1");             case "silver":                 System.out.println("\t included breakfast");             case "bronze":                 System.out.println("\t free parking");                 System.out.println("\t included newspaper");             default:                 System.out.println("\t room");         }     } }</pre> <p><b>Note:</b> Observe that to print out the aggregate benefits of each level, we put the cases in a particular order to take advantage of the fall-through functionality provided when we do not explicitly put a <code>break</code> in each <code>switch</code> block.</p>



# Chapter 7. Arrays

Arrays are a convenient and necessary part of any computing language, as they facilitate the grouping of data items together as a single indexed unit. In this chapter, you get an opportunity to exercise and enhance your skills with arrays in Java. Among the exercises presented in this chapter are exercises on array declaration, creation, initialization, assignment, access, copying, reversal, subset manipulation, sorting and element searches. Exercises on multi-dimensional arrays are also presented.

In addition to the built-in syntax for array handling, recourse is made heavily to the methods of the class `Arrays`.

1.	In Java, what is an <i>array</i> ?	An array is a data element that has an indexed list of items (primitives or objects) that are of the * same type. The array index is an integer index. *If you want to put objects that are of different types into an array, you have to cast the objects to the type of the array (if possible in the given case). In that sense the items in the array are of the same type.
2.	Are arrays descended from the class <code>Object</code> ?	Yes they are, as <u>arrays are objects</u> and the ultimate ancestor class of objects in Java is the class <code>Object</code> .
3.	I have an array of integers defined as follows:  <code>int[] intArray = new int[3];</code> Explain in words what each part of this declaration statement means.	1. <code>int[]</code> is the type of what we are going to define, which in this case is a reference variable to an array of integers. In a variable declaration, <code>[]</code> after the name of a type indicates that it is an array reference variable that will contain the specified type ( <code>int</code> in this case). 2. <code>intArray</code> is the variable name we've chosen to give to this array object reference that contains elements of type <code>int</code> . 3. <code>new int[3]</code> means that we are asking the system to allocate space for an array of 3 integers. 4. <code>=</code> This is saying that the newly allocated array of integers is going to be known by the variable name <code>intArray</code> which appears on the left-hand side of the equals sign.
4.	What is the 1 <sup>st</sup> index of an array?	0. Arrays in Java always start at index 0.
5.	Write a line of code which declares an <code>int</code> array variable named <code>intArray</code> .	<code>int[] intArray;</code>
6.	Write a line of code which will instantiate an <code>int</code> array named <code>intArray</code> with a capacity for 20 integers.	<code>int[] intArray = new int[20];</code>
7.	Assign the value 2 to the 1 <sup>st</sup> index of the array defined above.	<code>intArray[0] = 2; // Remember that the 1<sup>st</sup> index is 0</code>
8.	Assign the value 800 to the 1 <sup>st</sup> index of the array above.	<code>intArray[0] = 800;</code> This simply overwrites what was previously set therein.
9.	Assign the value 17 to the 3 <sup>rd</sup> index of the array <code>intArray</code> defined earlier.	<code>intArray[2] = 17;</code>
10.	Write a program which instantiates an array of size 5 of each of the following types: <code>Float</code> , <code>float</code> , <code>Integer</code> , <code>int</code> , <code>double</code> , <code>boolean</code> .	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         Float[] fArray01      = new Float[5];         float[] fArray02     = new float[5];         Integer[] iArray01   = new Integer[5];         int[] iArray02      = new int[5];         double[] dArray01   = new double[5];     } }</pre>

## Practice Your Java Level 1

		<pre>        boolean[] bArray01 = new boolean[5];     }</pre>
11.	<p>Write a code fragment which will instantiate a <code>String</code> array that has a capacity for 10 <code>String</code> objects.</p> <p>Initialize the first 5 elements of the array with each of the following words respectively: “Mary”, “had”, “a”, “little”, “lamb”.</p>	<pre>String[] sArray01 = new String[10]; sArray01[0]= "Mary"; sArray01[1]="had"; sArray01[2]="a"; sArray01[3]="little"; sArray01[4]="lamb";</pre>
12.	<p>Write a program which will print out the <code>length</code> of the array from the preceding exercise.</p> <p><i>Hint: the <code>Length</code> field of arrays.</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {          String[] sArray01 = new String[10];         sArray01[0] = "Mary";    sArray01[1] = "had"; sArray01[2] = "a";         sArray01[3] = "little"; sArray01[4] = "lamb";         System.out.printf("Array length = %d\n", sArray01.length);     } }</pre> <p><b>Note:</b> As seen from the answer above, the <code>length</code> of the array is the number of allocated spaces in the array, not the number of occupied spaces.</p>	
13.	<p>Write a program which instantiates an integer array named <code>intArray01</code> that can hold 50 integers. Print out the length of the array using the <code>Length</code> field of arrays.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {          int[] intArray01 = new int[50];         int arrayLength = intArray01.length; // The length of the array         System.out.printf("The length of the array is %d.\n", arrayLength);     } }</pre>	
14.	<p>State the difference between the following two initialization methods for an array:</p> <pre>int[] intArray01 = new int[5]; intArray01[0] = 10; intArray01[1] = 11; intArray01[2] = 12; intArray01[3] = 13; intArray01[4] = 14;</pre> <p><i>versus</i></p> <pre>int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };</pre>	<p>In this example, there is no difference at all. The second method is quite convenient when you are hardcoding arrays into a program and also want the length of the array to match the number of elements that you are initializing it with. In this case (the second case), the compiler infers the desired number of elements of the array by counting the number of comma separated items in the braces.</p> <p>The benefit of the first initialization method however is that we can declare the array to be of a size greater than the number of elements that we plan to initialize it with, as opposed to the second method where the compiler only allocates as many spaces to the array as the number of elements specified at instantiation.</p> <p>Nonetheless, irrespective of whichever method is chosen, Java provides an indirect way to modify the number of allocated spaces in an already existing array. We will see this in a later exercise.</p>
15.	<p><i>Yes or No:</i> Does Java initialize the values in an array?</p>	
	<p>Yes.</p> <p>The reader should contrast this with the fact that Java does not initialize individual variables.</p>	
16.	<p>What are the elements of an array of numerical primitives initialized to by default?</p>	
	<p>0.</p>	

17.	What are the elements of an array of objects initialized to by default? <code>null</code>
18.	Explain what is wrong with the following code: <pre>public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };         System.out.printf("%d\n", intArray01[5]);     } }</pre> <p>The code attempts to print out the 6<sup>th</sup> element of an array that only has 5 elements defined. On running the code, the following exception will be produced: <a href="#">java.lang.ArrayIndexOutOfBoundsException</a>. Conclusion: Never attempt to access a non-existent array position.</p>
19.	Write a program which instantiates an <code>int[]</code> with 0 elements. Print the length of the array out. <pre>public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[0];         System.out.println("Length of intArray01 = " + intArray01.length);     } }</pre>
20.	Explain the difference between the following two array declarations: <code>int[] intArray01;</code> <code>int[] intArray02 = new int[0];</code> <p>The first declaration is merely creating a variable that will hold an array; no array has been defined at all. The second declaration actually defines an empty array.</p>
21.	<p><i>Copying array contents</i>  Copy the following array to a new <code>int[]</code> named <code>intArray02</code>. Print the contents of the new array out.  <code>int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };</code>  <i>Hint: Arrays.copyOf</i></p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };         int[] intArray02 = Arrays.copyOf(intArray01, intArray01.length);         System.out.printf("%d, %d, %d, %d, %d \n", intArray02[0], intArray02[1], intArray02[2],                            intArray02[3], intArray02[4]);     } }</pre>
22.	<p>Modify the solution to the preceding exercise to also check whether the contents of <code>intArray01</code> and <code>intArray02</code> are the same. Print out your findings.  <i>Hint: Arrays.equals</i></p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };         int[] intArray02 = Arrays.copyOf(intArray01, intArray01.length);          boolean areEqual = Arrays.equals(intArray01, intArray02);         if(areEqual == true)             System.out.println("The arrays have the same content!");         else             System.out.println("The arrays do NOT have the same content!");     } }</pre>

## Practice Your Java Level 1

23.	<p><i>Copying part of an array to another</i></p> <p>We have an integer array <code>intArray01</code> that has 5 elements as follows:</p> <pre>[0] - 55 [1] - 747 [2] - 15000 [3] - 89 [4] - 2333</pre> <p>Copy the 1<sup>st</sup> three elements of <code>intArray01</code> into an array <code>intArray02</code>. Print the length of <code>intArray02</code> and its contents out.</p> <p><i>Hint: Arrays.copyOfRange</i></p>	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] {500, 747, 15000, 89, 2333};         int[] intArray02;         // Now the 1st 3 elements of intArray01 to intArray02         // Therefore we are index 0 to index 2         intArray02 = Arrays.copyOfRange(intArray01, 0, 3);         // Observe the assignment to intArray02. An array of the         // exact length returned by the copy is returned.         System.out.printf("intArray02's length is %d\n",                            intArray02.length);         System.out.printf("intArray02's contents are %d, %d, %d \n",                            intArray02[0], intArray02[1],                            intArray02[2]);     } }</pre> <p><b>Notes:</b> Observe that the length of the array returned is exactly the number of elements put into it.</p>
24.	<p>We have an integer array <code>intArray01</code> which has 5 elements with the following content:</p> <pre>[0] - 55 [1] - 747 [2] - 15000 [3] - 89 [4] - 2333</pre> <p>Copy the 2<sup>nd</sup> to the 4<sup>th</sup> elements of <code>intArray01</code> into the 3<sup>rd</sup> to 5<sup>th</sup> position of <code>intArray02</code> which also is an int array of 5 elements. Print the contents of <code>intArray02</code> out before and after the copy operation.</p> <p><i>Hint: method System.arraycopy</i></p>	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};         int[] intArray02 = new int[intArray01.length]; //the  //target array has to be &gt;= the source in length         System.arraycopy(intArray01, 0, intArray02, 2, 3);         // The 2nd element in intArray01 is position [1],         // the 3rd element in intArray02 is position [2].         // and we are copying 3 elements.          System.out.printf("intArray02's contents are now                            %d,%d,%d,%d,%d\n", intArray02[0],                            intArray02[1], intArray02[2],                            intArray02[3], intArray02[4]);     } }</pre>
25.	<p>Rewrite the preceding program, using the <code>Arrays.toString</code> method to output the contents of the array.</p>	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};         int[] intArray02 = new int[intArray01.length];         //the target array has to be &gt;= the source in length          System.arraycopy(intArray01, 0, intArray02, 2, 3);         // The 2nd element in intArray01 is position [1],         // the 3rd element in intArray02 is position [2].         // and we are copying 3 elements.          System.out.printf("intArray02's contents are now %s\n", Arrays.toString(intArray02));     } }</pre>
26.	<p>Run the following program and state your observations on it and its output.</p>	<pre>import java.util.*;</pre>

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
        int[] intArray02;
        System.out.printf("intArray01's contents are currently %s\n", Arrays.toString(intArray01));
        intArray02 = Arrays.copyOfRange(intArray01, 0, 10);
        System.out.println("Length of intArray02 = " + intArray02.length);
        System.out.printf("intArray02's contents are %s\n", Arrays.toString(intArray02));
    }
}

```

**Observations:** In the call to the method `Arrays.copyOfRange`, the program states that it is going to copy the contents of `intArray01` to `intArray02`. However, the specified number of elements to “copy” is **10**, which is greater than the number of elements in the source array `intArray01`. Nevertheless, the code runs and we see that `intArray02` has a length of 10 elements! On looking at the contents of `intArray02` we observe that indeed the values in `intArray01` were copied to `intArray02` and the extra positions were filled with the default value of **0** for the `int[]` type.

This mechanism of specifying a greater length than the length of the source array is actually a way to use the method `Arrays.copyOfRange` to programmatically create an array of longer length than the source array. This same mechanism can be used to lengthen the source array itself by specifying the same array as source and destination (what really happens is that another array object would be created in the background with the same data as the original array and that the memory address of the new array object would be assigned to the variable name of the original array).

27. Shorten the following array to only its 1<sup>st</sup> three elements:

```
int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
```

Print out the length of `intArray01` to prove that its length has indeed been modified.

```

import java.util.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
        intArray01 = Arrays.copyOfRange(intArray01, 0, 3); //source & destination arrays are the same
        System.out.println("New length of intArray01 = " + intArray01.length);
    }
}

```

28. *Extending the length/Increasing the capacity of an array*

Extend the length of the following array to ten elements:

```
int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
```

Print out the length of `intArray01` as well as its contents.

*Hint: Read the solution to exercise 26.*

```

import java.util.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
        System.out.println("Current length of intArray01 = " + intArray01.length);

        intArray01 = Arrays.copyOfRange(intArray01, 0, 10);
        // Note that the source & destination arrays are the same

        System.out.println("New length of intArray01 = " + intArray01.length);
        System.out.printf("intArray01's contents are now %s\n", Arrays.toString(intArray01));
        // Note that the extra elements are automatically set to the default value of the type
        // of the array.
    }
}

```

29. We have an integer array named `intArray01` initialized with the following 5 elements: {17, 42, 43, 8, 23}.

Write a program which extends the length of this array and then puts two more elements, namely the values 57 and 84 at the end of the array. Print out the length of the new array as well as its contents.

## Practice Your Java Level 1

```
import java.util.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        int[] intArray01 = new int[] {17,42,43,8,23};  
  
        // We use an indirect mechanism to extend the length of the array;  
        intArray01 = Arrays.copyOf(intArray01, intArray01.length + 2);  
        // So we gave it the contents of intArray01 and told it to return the same data  
        // in an array of length of intArray + 2.  
  
        // Now put our desired data into the two new elements.  
        intArray01[5] = 57;  
        intArray01[6] = 84;  
  
        System.out.printf("intArray01's new length is %d\n", intArray01.length);  
        System.out.printf("intArray01's contents are now %s\n", Arrays.toString(intArray01));  
    }  
}
```

30. *Shortening the length/Decreasing the capacity of an array*

We have the following array: `int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};`  
Programmatically modify this array to the following: `int[] intArray01 = new int[] {15000, 89, 2333};`  
Print out the new length of the array as well as its contents.

```
import java.util.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        int[] intArray01 = new int[] {55,747,15000,89,2333};  
        int startingIndex = 2;  
        int finalIndex = intArray01.length;  
  
        intArray01 = Arrays.copyOfRange(intArray01, startingIndex, finalIndex);  
        System.out.println("New length of intArray01 = " + intArray01.length);  
        System.out.printf("intArray01's contents are now %s\n", Arrays.toString(intArray01));  
    }  
}
```

31. We've seen that when an array is created but not initialized, it is filled with the default value for its type. For example, an `int[]` is filled with the value `0`. However, there are times when `0` is a valid data point for us and so we wouldn't know whether we ourselves set particular entries to `0`, or whether it is the default value that is the content of any given array element. This being the case, what is the easiest solution with which to set the elements in an array to a value that is not a valid data point for our data so that we can tell whether or not we have set the value? For example, we might want to set the entries to `-1` as the default.

*Hint: Arrays.fill*

The solution is to fill the array with the desired flag value, using the `Arrays.fill` method.

32. Create an `int` array of length 5 and initialize each of its elements with the value `-1`. Print the array out afterwards.

```
import java.util.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        int[] intArray01 = new int[5];  
  
        Arrays.fill(intArray01, -1);  
        System.out.printf("intArray01's contents are now %s\n", Arrays.toString(intArray01));  
    }  
}
```

33. Sort the following array and print its contents out: `int[] intArray01 = new int[] {55,747,15000,89,2333};`  
*Hint: Arrays.sort*

```
import java.util.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {
```

	<pre> int[] intArray01 = new int[] {55, 747, 15000, 89, 2333}; Arrays.sort(intArray01); System.out.printf("intArray01's contents are now %s\n", Arrays.toString(intArray01)); } </pre>
34.	<p>Repeat the same exercise as above, this time using the <code>Arrays.parallelSort</code> method.</p> <pre> import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};         Arrays.parallelSort(intArray01);         System.out.printf("intArray01's contents are now %s\n", Arrays.toString(intArray01));     } } </pre>
35.	<p>Briefly explain the difference between <code>Arrays.parallelSort</code> and <code>Arrays.sort</code>.</p> <p>The difference between the two methods is that <code>Arrays.parallelSort</code> is designed to be able to sort using multiple threads. The result is that on multi-core or multi-threaded processors <code>Arrays.parallelSort</code> should be faster than <code>Arrays.sort</code>.</p>
36.	<p>Determine using the <code>binarySearch</code> method of class <code>Arrays</code>, which array element contains the value <code>747</code> in the sorted array of the preceding exercise. (<code>binarySearch</code> only works on sorted arrays).</p> <pre> import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};         int valueToSearchFor = 747;         int position;         Arrays.sort(intArray01);         position = Arrays.binarySearch(intArray01, valueToSearchFor);         if(position &gt;= 0)             System.out.printf("The value %d was found at position %d\n", valueToSearchFor, position);     } } </pre>
37.	<p>I have the following <code>String</code> object: <code>s01 = "Hello how are you?"</code>. Convert this to an array of bytes. State how many elements are in the byte array.</p> <p><i>Hint: String.getBytes</i></p> <pre> import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         String s01 = "Hello how are you?";         byte[] strAsByteArray = s01.getBytes();         System.out.printf("The length of the byte array is %d\n", strAsByteArray.length);     } } </pre> <p><b>Important note:</b> The conversion is done in the default <code>charset</code> of the system you are running your program on. We do not cover the concept of charsets in this book, however you should be aware of the concept of charsets, as the selected/specify charset can affect the equivalent number of bytes that a given string converts to.</p>

38.	<p><i>Matrices/Multi-dimensional arrays</i></p> <p>Write a line of code to declare a variable <code>matrixA</code> that can refer to a 2-dimensional <code>int</code> array/matrix.</p>	<pre>int[][] matrixA;</pre> <p>The number of braces (two in this case) is what tells the compiler how many dimensions the multi-dimensional array in question will have.</p>
39.	<p>Write a line of code to instantiate a 2-dimensional integer matrix named <code>matrixA</code> of dimensions <math>2 \times 3</math>.</p>	<pre>int[][] matrixA = new int[2][3];</pre>

## Practice Your Java Level 1

	(i.e. 2 rows and 3 columns).	
40.	Write the appropriate line of code to declare a 5-dimensional integer array/matrix variable named <code>matrixB</code> .	<code>int[][][][][] matrixB;</code>
41.	Write a line of code that shows the instantiation of a 5-dimensional integer matrix named <code>matrixB</code> of dimensions $10 \times 4 \times 3 \times 5 \times 12$ .	<code>int[][][][][] matrixB = new int[10][4][3][5][12];</code>
42.	Instantiate two $2 \times 3$ integer matrices, with variable names <code>matrix1</code> and <code>matrix2</code> respectively and initialize them as follows:  <code>matrix1</code> <code>1 2 3</code> <code>4 5 6</code>  <code>matrix2</code> <code>5 5 8</code> <code>3 1 3</code>	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[][] matrix1 = new int[2][3];         matrix1[0][0] = 1; matrix1[0][1] = 2; matrix1[0][2] = 3;         matrix1[1][0] = 4; matrix1[1][1] = 5; matrix1[1][2] = 6;          int[][] matrix2 = new int[2][3];         matrix2[0][0] = 5; matrix2[0][1] = 5; matrix2[0][2] = 8;         matrix2[1][0] = 3; matrix2[1][1] = 1; matrix2[1][2] = 3;     } }</pre> <p><i>OR, in compact form,</i></p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[][] matrix1 = new int[][] { { 1, 2, 3 }, { 4, 5, 6 } };         int[][] matrix2 = new int[][] { { 5, 5, 8 }, { 3, 1, 3 } };         // Observe that the data is entered row by row,         // comma delimited in curly braces per row.     } }</pre>
43.	Add the contents of the arrays of the preceding exercise, putting the result into a new matrix, <code>matrix3</code> .	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int[][] matrix1 = new int[][] { { 1, 2, 3 }, { 4, 5, 6 } };         int[][] matrix2 = new int[][] { { 5, 5, 8 }, { 3, 1, 3 } };          int[][] matrix3 = new int[2][3];         matrix3[0][0] = matrix1[0][0] + matrix2[0][0];         matrix3[0][1] = matrix1[0][1] + matrix2[0][1];         matrix3[0][2] = matrix1[0][2] + matrix2[0][2];         matrix3[1][0] = matrix1[1][0] + matrix2[1][0];         matrix3[1][1] = matrix1[1][1] + matrix2[1][1];         matrix3[1][2] = matrix1[1][2] + matrix2[1][2];     } }</pre>
E1	Note the method <code>Arrays.asList(T... a)</code> . We will see it later in the chapter on Collections.	

There are a number of other built-in different ways and forms in which arrays and more advanced variants thereof are presented in Java; these will be seen in the chapter entitled “Collections”. Also, we will see further manipulations of arrays in the next chapter.

# Chapter 8. Iterations/Loops

The facility in a programming language to iterate easily through a set of items (for example the individual elements of an array) and perform common actions on each element is a key enabler of compact code. This chapter contains exercises on different loop mechanisms within Java, such as the `while`, `do/while`, `for` and enhanced `for` loops. Exercises which require the use of “labels” as well as the loop jump keywords `continue` and `break` are also presented, as well as exercises on nested loops and infinite loops.

while loop		
1.	Use a <code>while</code> loop to print out the numbers 0 to 9 in order, on sequential lines.	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int myCounter = 0; // We are starting at 0         while (myCounter &lt; 10) {             System.out.println(myCounter);             myCounter = myCounter + 1; //we increment the counter here         }     } }</pre>
2.	Use a <code>while</code> loop to add the numbers 1 to 15,000. Print out the result.	<pre>public class PracticeYourJava {     public static void main(String[] args) {          long myCounter = 1; // We are starting at 1         long total = 0L;         while (myCounter &lt;= 15000) {             total+= myCounter;             myCounter++; //we increment the counter here         }         System.out.println("Total = " + total);     } }</pre>
3.	We have an integer array <code>intArray01</code> which has 5 elements with the following content in the noted positions in the array:  [0] - 18 [1] - 42 [2] - 26 [3] - 44 [4] - 55  Using a <code>while</code> loop, print out the values contained in this array, in the following format: <code>intArray01[&lt;index&gt;] = &lt;value&gt;</code>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         int loopCounter = 0; // We are starting at element 0         int arrayLength = intArray01.length; // upper limit          while (loopCounter &lt; arrayLength) {             System.out.printf("intArray01[%d] = %d\n",                 loopCounter,intArray01[loopCounter]);             loopCounter++;         }     } }</pre>
4.	Print out the contents of the array in the preceding exercise <i>backwards</i> , in the same output format, using a <code>while</code> loop.	<pre>public class PracticeYourJava {     public static void main(String[] args) {          int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         int loopCounter = intArray01.length - 1; // We are starting at the last element          while (loopCounter &gt;= 0) {             System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);             loopCounter = loopCounter - 1;         }     } }</pre>

## Practice Your Java Level 1

```
}
```

### for loop

5. Explain the components and the functioning of a **for** loop, using the sample code below:

```
int loopCounter;
for(loopCounter=3; loopCounter < 10; loopCounter++)
{
    // do action
}
```

The control section of a **for** loop consists of the following elements:

1. The keyword **for**, which brackets the other control elements of the loop.
2. The other control elements which are semi-colon separated within the brackets.

In the brackets we have, in order, the following:

1. A statement indicating the name of the control variable of the **for** loop, with its initial value specified. In our example above, we state that our loop counting variable named **loopCounter** has an initial value of 3. Note that the initialization of the control variable does not have to be done inside the control section of the **for** loop; in such a case, the initialization section of the **for** loop will be left blank.
2. A conditional statement that states the condition which while true permits the **for** loop to continue running. In our example we are saying “keep performing this loop *while* the value of the variable **loopCounter** is less than 10”.
3. Finally, an increment statement, which indicates the incremental value to add to the loop counter after each iteration of the actions that are supposed to be performed in the statements governed by the **for** loop are performed. In our example here, we indicate that the loop counter **loopCounter** should be incremented by 1.

The body of the **for** loop is shown in this example as:

```
{
    // do action
}
```

These braces that immediately follow the **for** loop surround the code that is to be run on each iteration of the **for** loop.

A **for** loop is, in a sense, a repackaged **while** loop in Java. This can be said because its conditional statement is saying “*while this condition is true*, continue looping”.

### Notes

1. If you only have one line that you want to be performed for each iteration of a **for** loop, there is no need to put braces around the single line to be run. For example we would write:

```
int loopCounter;
for(loopCounter=3; loopCounter < 10; loopCounter++)
    single line of action.
```

2. The loop counter can be declared directly in the control section of the **for** loop. For example we can write:

```
for(int loopCounter = 0; loopCounter < 20; loopCounter++)
{
    //do action
}
```

The scope of a variable declared in the loop control section is limited to the scope of the **for** loop itself.

3. Another point to note is that it is acceptable for any (or all) of the loop control components to be missing! Only the two semicolons in the control statement are mandatory.

6.	<p>Predict the output of the following <code>for</code> loop.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         for (int loopCounter = 0; loopCounter &lt; 5; loopCounter = loopCounter + 1) {             System.out.println("Hello!");         }     } }</pre> <p><b>Output:</b></p> <pre>Hello! Hello! Hello! Hello! Hello!</pre>
7.	<p>Write a program which using a <code>for</code> loop will output the values 2 to 10.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         for (int i = 2; i &lt; 11; i++) {             System.out.println(i);         }     } }</pre>
8.	<p>What is the difference between the following two <code>for</code> loop code samples?</p> <pre>int i; for (i=0; i &lt; 10; i++) {     System.out.println(i); }</pre> <p>vs.</p> <pre>for (int i=0; i &lt; 10; i++) {     System.out.println(i); }</pre> <p>The difference between the loops is that in the second code snippet, the loop control variable is defined within the control structure of the <code>for</code> loop. Otherwise the loops are equivalent in function. Also note that in the second code sample, the lifetime of the loop control variable expires at the end of the <code>for</code> loop since it was defined within the control structure of the loop.</p>
9.	<p>What is the difference between the following two <code>for</code> loop code samples?</p> <pre>int i; for (i=0; i&lt;10; i++) {     System.out.println(i); }</pre> <p>vs.</p> <pre>int i=0; for ( ; i&lt;10; i++) {     System.out.println(i); }</pre> <p>The difference between the loops is that in the second code snippet, the loop control variable is initialized outside/before the control structure of the <code>for</code> loop. Otherwise the loops are completely equivalent in function.</p>
10.	<p>What is the difference between the following two <code>for</code> loop code samples?</p> <pre>for (int i=0; i&lt;10; i++){     System.out.println(i); }</pre> <p>vs.</p> <pre>for (int i=0; i&lt;10; ) {     System.out.println(i);     i++; }</pre> <p>The difference between the loops is that in the second code snippet the loop increment variable is incremented within the code block, rather than in the control section. Otherwise the loops are completely equivalent in function.</p>
11.	<p>Predict and explain the output of the following code snippet:</p> <pre>for (int i=0; i &lt; 3; i++)     System.out.println(i);     System.out.println("hello");</pre> <p>The output is:</p> <pre>0 1 2 hello</pre> <p><b>Explanation</b></p>

## Practice Your Java Level 1

		If there are no braces around lines of code following a <b>for</b> loop, only the first line of code immediately following the <b>for</b> loop is considered to be the body of the loop.
12.	Predict and explain the output of the following code snippet: <pre>for(int i=0;i&lt;10;i++);</pre>	Nothing is output, however the code is valid. Note the semi-colon at the end of the statement.
13.	Predict the output of this program. <pre>public class PracticeYourJava {     public static void main(String[] args) {         for(int j=0; j &lt; 10; j++, System.out.println("Hello"));     } }</pre>	<p>It prints out the word “Hello” 10 times.      Normally we may have written the <b>for</b> loop as follows:</p> <pre>for(int j=0; j &lt; 10; j++)     System.out.println("Hello");</pre> <p>However, the compact form shown in the exercise, showing that we can perform other actions in the incrementing part of the control section of the loop is also valid (however it might be deemed to be less readable though).</p>
14.	We have the following array definition: <code>int[] intArray01 = new int[]{18,42,26,44,55};</code> Using a <b>for</b> loop, print out the values contained in this array, in the following format: <code>intArray01[&lt;index&gt;] = &lt;value&gt;</code>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         int loopCounter;         for (loopCounter = 0; loopCounter &lt; intArray01.length; loopCounter = loopCounter + 1) {             System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);         }     } }</pre> <p><b>Notes/Explanation of the <b>for</b> loop in Java</b>      Really, in Java, the <b>for</b> loop can be thought of as a <b>while</b> loop of the following structure (using the answer above as an example):</p> <pre>loopCounter=0; // initial condition while(loopCounter &lt; intArray01.length) // the termination condition {     System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);     loopCounter=loopCounter+1; // the increment of the counter as specified }</pre> <p>If you are having trouble with formulating a <b>for</b> loop, you can first formulate its <b>while</b> loop counterpart.</p>
15.	Print the same array of the preceding exercise out <i>backwards</i> , in the same output format, using a <b>for</b> loop. <i>Hint: decrement the loop control variable</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         int loopCounter;         for (loopCounter = intArray01.length - 1; loopCounter &gt;= 0; loopCounter-=1)             System.out.printf("intArray01[%d] = %d\n", loopCounter,intArray01[loopCounter]);     } }</pre>
16.	Using a <b>for</b> loop, fill an <b>int</b> array of 15 elements with random integers between the values of 0 and 50,000. After filling all the elements, print them out using a <b>while</b> loop.	

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int numElements = 15;
        int[] array01 = new int[numElements];
        for(int i=0;i<array01.length;i++)
            array01[i] = (int)Math.ceil(Math.random() * 50000);
        int counter=0;
        while(counter < numElements) {
            System.out.printf("intArray01[%d] = %d\n", counter, array01[counter]);
            counter++;
        }
    }
}

```

### Enhanced for loop

17.	<p>We have an integer array <code>intArray01</code> which has 5 elements with the following content in the noted positions:</p> <table border="0" data-bbox="290 834 399 971"> <tr><td>[0]</td><td>- 18</td></tr> <tr><td>[1]</td><td>- 42</td></tr> <tr><td>[2]</td><td>- 26</td></tr> <tr><td>[3]</td><td>- 44</td></tr> <tr><td>[4]</td><td>- 55</td></tr> </table> <p>Using an enhanced <code>for</code> loop, print out the contents of the array. (<i>don't print out the index, just print the values</i>).</p>	[0]	- 18	[1]	- 42	[2]	- 26	[3]	- 44	[4]	- 55	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         for(int item : intArray01) {             System.out.println(item);         }     } } </pre>	<p><b>Note:</b> The enhanced <code>for</code> loop is more useful when you have to perform an action on each and every element in the array under consideration. Unlike the <code>for</code> and <code>while</code> loops, the enhanced <code>for</code> loop does not present an accessible index.</p>
[0]	- 18												
[1]	- 42												
[2]	- 26												
[3]	- 44												
[4]	- 55												
18.	<p>Using the same array as in the exercise above, use an enhanced <code>for</code> loop to print out the values contained in this array, in the following manner:</p> <pre>intArray01[&lt;index&gt;] = &lt;value&gt;</pre>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         int index = 0;         for(int item : intArray01) {             System.out.printf("intArray01[%d] = %d\n", index, item);             index = index + 1;         }     } } </pre>	<p><b>Notes/Explanation:</b> The enhanced <code>for</code> loop does not present an index to the user; rather, in each iteration it gets the next item in the sequence/array under assessment into the variable that is present in its declaration statement (in this exercise for example, it sequentially puts the content that is in the array <code>intArray01</code> element by element into the variable <code>item</code>). Therefore, we have to create our own index if we want to print an index out.</p>										
19.	<p>With respect to the array in the exercise above, use an enhanced <code>for</code> loop to print out every 2<sup>nd</sup> value (i.e. elements 0, 2 and 4 only) contained in this array, in the following format:</p> <pre>intArray01[&lt;index&gt;] = &lt;value&gt;</pre>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };         int index = 0; </pre>											

## Practice Your Java Level 1

```

        for(int item : intArray01) {
            if (index % 2 == 0) {
                System.out.printf("intArray01[%d] = %d\n", index, item);
            }
            index = index + 1;
        }
    }
}

```

**Explanation:** Again, due to the nature of the enhanced `for` loop statement, it does not lend itself as smoothly as using any of the other loop constructs for this same scenario.

20. Repeat the preceding exercise, this time using a `for` loop.

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

        for (int j = 0; j < intArray01.length; j+=2)
            System.out.printf("intArray01[%d] = %d\n", j, intArray01[j]);
    }
}

```

**Note:** Note how the loop index was incremented by 2 to achieve the specified output effect of printing every 2<sup>nd</sup> entry.

21. Repeat the preceding exercise using a `while` loop.

```

public class PracticeYourJava {
    public static void main(String[] args) {
        int[] intArray01 = new int[] { 18, 42, 26, 44, 55 };

        int counter = 0; // We will be starting at the 0th index.
        int arrayLength = intArray01.length;

        while (counter < intArray01.length) {
            System.out.printf("intArray01[%d] = %d\n", counter, intArray01[j]);
            counter = counter + 2;
        }
    }
}

```

### do/while loop

22. Under what circumstances is a `do/while` loop a good candidate to use?

When you have something that must run at least once before loop conditions are checked. Think of this loop as: *you have to do first and then check conditions for potential subsequent actions.*

23. Write a program which uses a `do/while` loop to print out the numbers 1 to 10.

```

public class PracticeYourJava {
    public static void main(String[] args) {

        int index = 1;
        do{
            System.out.println(index);
            index++;
        }while (index <= 10);
    }
}

```

24. Using a `do/while` loop, calculate and print out the factorial of 10.

```

public class PracticeYourJava {
    public static void main(String[] args) {

        int iNumToCalculateFactorial = 10;
        long result = 1;
        int index = 1;

        do {
            result = result * index;
        }
    }
}

```

	<pre>         index++;     }while (index &lt;= iNumToCalculateFactorial);     System.out.printf("%d! = %d\n", iNumToCalculateFactorial, result); } } </pre>
25.	<p>Using an <i>infinite while</i> loop, print the following string out indefinitely:</p> <p style="padding-left: 40px;">“Hello!”</p>
	<pre> public class PracticeYourJava {     public static void main(String[] args) {          while (true) {             System.out.println("Hello!");         }     } } </pre> <p><b>Note:</b> Recall that a <code>while</code> loop runs while whatever it evaluates in its control statement equals the boolean value <code>true</code>. Therefore, we simply need to put the boolean value <code>true</code> directly into the loop.</p>
26.	<p>Repeat the preceding exercise using an infinite <code>for</code> loop.</p>
	<pre> public class PracticeYourJava {     public static void main(String[] args) {          for ( ; ; ) {             System.out.println("Hello!");         }     } } </pre>
<b>Nested loops</b>	
27.	<p>Predict the output of the following code:</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          int j, k;         for (j = 1; j &lt; 5; j++) {             for (k = 1; k &lt; 5; k++) {                 System.out.printf("(%d,%d) ", j, k);             }             System.out.println();         }     } } </pre> <p><b>Output:</b></p> <p style="padding-left: 40px;">(1,1) (1,2) (1,3) (1,4)  (2,1) (2,2) (2,3) (2,4)  (3,1) (3,2) (3,3) (3,4)  (4,1) (4,2) (4,3) (4,4)</p> <p><b>Note:</b> The objective of this exercise was to show the application of <b>nested loops</b> in outputting matrices.</p>
28.	<p>Write a program which will output the following pattern:</p> <p style="padding-left: 40px;">(A,1) (A,2) (A,3) (A,4)  (B,1) (B,2) (B,3) (B,4)  (C,1) (C,2) (C,3) (C,4)  (D,1) (D,2) (D,3) (D,4)  (E,1) (E,2) (E,3) (E,4)</p> <pre> public class PracticeYourJava {     public static void main(String[] args){          int j, k;         String[] letters = new String[] {"A", "B", "C", "D", "E"};         // OR we can use a char array instead: char[] letters = new char[] {'A', 'B', 'C', 'D', 'E'};          for (j = 0; j &lt; letters.length; j++){             for (k = 1; k &lt; 5; k++){                 System.out.printf("(%s,%d) ", letters[j], k);             }         }     } } </pre>

## Practice Your Java Level 1

```

        }
        System.out.println();
    }
}

-OR-

public class PracticeYourJava {
    public static void main(String[] args){
        int j, k;
        int start = (int)'A';
        for (j = start; j < (start + 5); j++){
            for (k = 1; k < 5; k++){
                System.out.printf("(%s,%d) ", (char)j, k);
            }
            System.out.println();
        }
    }
}

```

### The keywords `continue` and `break` and the use of labels

29.	Write a <code>for</code> loop that can print out the values 1 to 1000 in sequence; however, using a <code>break</code> statement halt the running of the loop when the value of the loop counter is 10.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         for (int i = 0; i &lt;= 50; i++) {             if (i == 10) break;             System.out.println(i);         }     } }</pre>
30.	What does the <code>continue</code> operator do?	The <code>continue</code> operator is an operator which when present in a loop causes the code to immediately go back to the start of the loop.
31.	Write a <code>for</code> loop which loops from 1 to 50, printing out each value; however, using a <code>continue</code> statement, skip every value that is divisible by 5.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         for (int i = 0; i &lt;= 50; i++) {             if (i % 5 == 0) continue;             System.out.println(i);         }     } }</pre>
32.	What is a <u>label</u> in Java and how is it used?	A label is a name applied to a block of code. The label is placed at the beginning of the code block and is always terminated with a colon. The naming convention for a label is simply any valid identifier (i.e. any name that is valid for a variable). Note that you can put labels at any point in your code.
33.	Describe the functioning of the labeled <code>break</code> statement.	The labeled <code>break</code> statement is a statement that allows you to transfer control from a given point in a block of code to the <i>end of the labeled block</i> that the labeled <code>break</code> appears within. The usage of the labeled <code>break</code> statement is as follows:  <code>break &lt;label&gt;;</code> The labeled <code>break</code> statement has the benefit of allowing you to move around in code in a non-sequential manner if so desired. The labeled <code>break</code> statement even gives you the ability to jump out of loops. Also note that the labeled <code>break</code> statement is not exclusively used in loops, but can be used in any other code. The short example below can be run in order to observe the functioning of the labeled <code>break</code> statement.  <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {</pre>

```

POINT_A:{  

    System.out.println("1");  

    if (1==1) break POINT_A; //Of course 1 == 1 so the break will be triggered  

    System.out.println("2");  

    } // This is where the labeled break jumps to; the end of the labeled block  

    System.out.println("3");  

}
}

```

As can be seen when this code is run, the output is as follows:

```

1
3

```

The labeled break statement jumped to the end of the code block labeled `POINT_A`, thus skipping the line which was supposed to print out the string “2”.

Note that loop blocks (for example `for` blocks) are also deemed to be blocks that can be labeled.

34. I want to output the following pattern:

```

A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
C1 C2 C3 C4

```

Write a program to print the pattern out using a nested `for` loop and a `break` that breaks out of the loop to the end of the labeled block that it is in. Outside/after the end of the labeled block, print out “\nJust saw a C5”.

```

public class PracticeYourJava {  

    public static void main(String[] args) {  

        String[] sArray01 = { "A", "B", "C" };  

        POINT_A: for (int i = 0; i < sArray01.length; i++) {  

            for (int j = 1; j <= 10; j++) {  

                String sf01 = String.format("%s%d", sArray01[i], j);  

                if (sf01.equals("C5") == true) break POINT_A;  

                System.out.print(sf01); System.out.print(" ");  

            }  

            System.out.println();  

        }  

        System.out.println("\nJust saw a C5");  

    }
}

```

35. Describe the purpose of the `break` statement (not the labeled break) within the context of a loop.

A `break` statement within the context of a loop causes the loop to end processing immediately and for control to go to the statement immediately after the loop.

36. I want to output the following pattern:

```

A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
C1 C2 C3 C4
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10
E1 E2 E3 E4 E5 E6 E7 E8 E9 E10

```

Write a nested `for` loop to implement this.

*Hint: Use a `break` statement within the inner loop when C5 is seen. No label is necessary for this solution.*

```

public class PracticeYourJava {  

    public static void main(String[] args) {  

        String[] sArray01 = { "A", "B", "C", "D", "E" };  

        for (int i = 0; i < sArray01.length; i++) {  

            for (int j = 1; j <= 10; j++) {  

                String sf01 = String.format("%s%d", sArray01[i], j);  

                if (sf01.equals("C5") == true) break;  

                System.out.print(sf01); System.out.print(" ");
            }
        }
    }
}

```

## *Practice Your Java Level 1*

<b>E1</b>	<p>Using the same source pattern as the preceding exercise, write code to output the following pattern:</p> <pre> o          o oo         oo ooo        ooo oooo       oooo oooooo     oooooo ooooooo    ooooooo oooooooo   ooooooooo ooooooooo  oooooooooo </pre>
<b>E2</b>	<p>Using the same source pattern as above, write code to output the following pattern:</p> <pre> o          oooooooooo oo         oooooooo ooo        oooooo oooo       ooooo oooooo     oooo ooooooo    ooo oooooooo   oo ooooooooo  o </pre>
<b>E3</b>	<p>I have an array of strings, each string therein of undetermined length.    Write out the array of strings, with the same width for each, bounding the strings left &amp; right with the ‘ ’ character.</p> <p><i>Hint: determine the length of the longest string. That will be your field width. Then use <code>String.format</code> to output each line as appropriate.</i></p>



# Chapter 9. Number Handling II – Numeric Wrapper Classes

As stated in the chapter entitled *Number Handling I*, the numeric primitives have corresponding wrapper classes (`Integer`, `Float`, `Byte`, `Short`, `Long` and `Double`); we had in that chapter used some of the constants from the wrapper classes. This chapter presents exercises that focus on the numeric wrapper classes.

The functionality of the wrapper classes can largely be broken up into the following groups:

- Constants that pertain to their respective primitives (already seen in the chapter entitled *Number Handling I*)
- Methods and means (boxing/autoboxing) which convert between the primitive and its wrapper class
- Methods which mimic the functionality of their primitives
- Methods which provide additional functionality for their respective primitives

It should be understood that at present the wrapper classes do not provide the full functionality of their respective primitives, but largely provide new functionality and in some cases overlapping functionality. Nonetheless, to enjoy the full functionality of numbers in Java one must be conversant with both the primitives and their wrapper classes.

Given the natural relationship of the primitives to their respective classes, we will also see a number of exercises in this chapter that show the auto-boxing/unboxing of primitives and their respective classes.

This chapter is the second of five chapters in this book on number handling in Java.

1.	List the corresponding wrapper classes for the following primitive types: <code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> .	The wrapper classes are (respectively): <code>Byte</code> , <code>Short</code> , <code>Integer</code> , <code>Long</code> , <code>Float</code> and <code>Double</code> .
2.	Write a program which instantiates an object of class <code>Integer</code> , with an initial value of 5. Use the <code>new</code> operator.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer i = new Integer(5);     } }</pre>
3.	I have a given <code>int</code> , <code>k</code> , with a value of 53. Instantiate an object of class <code>Integer</code> using the <code>int k</code> as the initialization parameter.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int k = 5;         Integer int01 = new Integer(k);     } }</pre>
4.	What is the concept of <i>boxing</i> ?	The concept of boxing is the means by which code can be written in Java to take a primitive type variable and create its equivalent wrapper object by using the primitive type variable as a parameter with which to initialize the corresponding wrapper class object. For example if we want to assign an <code>int j</code> with a value of <code>100</code> to an object of its corresponding class <code>Integer</code> the code would be as follows: <pre>int j = 100; Integer boxedEquivalent = new Integer(j);</pre> We have seen this done in the immediately preceding exercise; its formal name is <i>boxing</i> .
5.	What is the concept of <i>auto-boxing</i> ?	This feature is the automated version of boxing. Auto-boxing is the name of the Java feature by which we create a wrapper class object simply by assigning a primitive value directly to it without the use of standard object creation syntax (i.e. use of the keyword <code>new</code> ). For example, if we want to represent an <code>int j</code> as an object of its corresponding class <code>Integer</code> the code would simply be as follows: <pre>Integer boxedEquivalent = j;</pre>

## Practice Your Java Level 1

	<p>The compiler takes care of the details in the background. Contrast the ease of this with the use of boxing. This feature was made available in Java as of version 1.5.</p>
6.	<p>What is the concept of <i>unboxing</i>?</p> <p>Unboxing is the means by which you explicitly, by means of a cast, convert a boxed type back to its corresponding primitive type. For example if we want to convert the contents of an <code>Integer</code> object <code>obj01</code> with a value of <code>100</code> to an <code>int</code> <code>k</code>, the code would be as follows:</p> <pre>Integer obj1 = new Integer(100); int k = (int)obj1;</pre>
7.	<p>What is the concept of <i>auto-unboxing</i>?</p> <p>(This feature is the automated version of unboxing). Auto-unboxing is the name of the Java feature by which the compiler converts an object of a numeric wrapper class to its primitive equivalent simply by means of using the assignment operator. For example if we have an <code>Integer</code> object <code>obj1</code> and we want to put its contents into a primitive <code>k</code> of type <code>int</code>, we simply need to do the following:</p> <pre>int k = obj1;</pre> <p>The compiler takes care of any details in the background.</p>
8.	<p>Why is the following (see below) a valid way to instantiate an object of class <code>Integer</code> as opposed to using the <code>new</code> operator?</p> <pre>Integer int01 = 500;</pre> <p>As described earlier, this is auto-boxing in effect. The compiler knows to create an object when it sees such an assignment.</p>
9.	<p>Rewrite the solution to exercise 3, using autoboxing instead of the <code>new</code> operator.</p>
10.	<p>Rewrite the following program, using an <code>Integer</code> object in place of an <code>int</code>.</p> <pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         int i = 753;         byte b = (byte)i;     } }</pre> <p><i>Hint: method Integer.byteValue</i></p>
11.	<p>With respect to the solution to the preceding exercise, explain why are the following lines of code equivalent, keeping in mind that <code>i</code> is an object of class <code>Integer</code>:</p> <pre>byte b = i.byteValue(); and b = (byte)((int)i);</pre> <p>The lines of code are equivalent because (referring to the second line of code), by means of unboxing we can cast an object of class <code>Integer</code> to its corresponding primitive (<code>int</code>). After we obtain the <code>int</code>, by means of a regular cast we can cast it to any primitive of choice.</p>
12.	<p>Rewrite the solution to exercise 10 above, using auto-unboxing.</p>
13.	<p>With respect to the <code>Integer</code> object <code>i</code> in the preceding exercise, explain why the following attempted conversion is illegal:</p> <pre>b = (byte)i;</pre> <p>The conversion is illegal because auto-boxing and auto-unboxing only convert directly between a variable of a given primitive class and its own wrapper class.</p>
14.	<p>Rewrite the following program, using an <code>Integer</code> object</p> <pre>public class PracticeYourJava {</pre>

	<p>in place of an <code>int</code>. Do not use casting or any type of boxing to achieve the desired result.</p> <pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         int i = 753;         short k = (short)i;     } }</pre> <p><i>Hint: method <code>Integer.shortValue</code></i></p>	<pre>public static void main(String[] args) {     Integer i = 753;     short k = i.shortValue(); }</pre>
15.	<p>Rewrite the following code fragment, using an <code>Integer</code> object in place of an <code>int</code>.</p> <pre>int i = 49; long k = (long)i;</pre> <p><i>Hint: method <code>Integer.LongValue</code></i></p>	<pre>Integer i = 49; long k = i.longValue();</pre>
16.	<p>Rewrite the following code fragment, using an <code>Integer</code> object in place of an <code>int</code>.</p> <pre>int i = 49; float f = (float)i;</pre> <p><i>Hint: method <code>Integer.floatValue</code></i></p>	<pre>Integer i = 49; float f = i.floatValue();</pre>
17.	<p>Rewrite the following code fragment, using an <code>Integer</code> object in place of an <code>int</code>.</p> <pre>int i = 2759; double k = (double)i;</pre>	<pre>Integer i = 2759; double k = i.doubleValue();</pre>
18.	<p>Rewrite the following code fragment, using an <code>Integer</code> object in place of an <code>int</code> for the variable <code>i</code>. Leave <code>k</code> as an <code>int</code> variable.</p> <pre>int i = 500; int k = i;</pre>	<p><i>or</i></p> <pre>Integer i = new Integer(500); int k = i.intValue();  Integer i = new Integer(500); int k = i; // the compiler auto-unboxes.</pre>
19.	<p>Rewrite the solution to exercise 17 above to take advantage of unboxing and casting to achieve the same result, instead of using the method <code>Integer.doubleValue</code>.</p>	<pre>Integer i = 2759; double k = (double)((int)i));</pre>
20.	<p>Rewrite this code, using an <code>Integer</code> in place of an <code>int</code>.</p> <pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         int i = 753;         System.out.printf("%d",i);     } }</pre>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer i = 753;         System.out.printf("%d", i);         // The object i will be unboxed         // automatically     } }</pre>
21.	<p>Rewrite the code below using <code>Integer</code> objects in place of the <code>int</code> primitives.</p> <pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         int j = 753;         int k = j; //set k to the value of j.         k++;         System.out.printf("%d",j); //j is still 753         System.out.printf("%d",k); //k is now 754     } }</pre>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer j = 753;         Integer k = j;         k++;         System.out.printf("%d\n", j);         System.out.printf("%d\n", k);     } }</pre>

## Practice Your Java Level 1

22. Rewrite the following <code>for</code> loop using an <code>Integer</code> as the control variable instead of an <code>int</code> .	<pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         Integer j = new Integer(0);         for (; j &lt; 5; j = j + 1) {             System.out.println("Hello!");         }     } }</pre>
23. Rewrite the following code, using <code>Integer</code> objects and the method <code>Integer.compareTo</code> in place of primitives.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer h=5; Integer j=7;         if(h&gt;j)             System.out.println("i&gt;j");         if(h&gt;i)             System.out.println("i&lt;j");         if(h==j)             System.out.println("i==j");     } }</pre>
24. Rewrite the solution to the preceding exercise using auto-unboxing.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer h=5; Integer j=7;         if(h &gt; j) System.out.println("h &gt; j");         if(j &gt; h) System.out.println("h &lt; j");         if(h == j) System.out.println("h == j");     } }</pre> <p><b>Note:</b> It can be seen clearly from this exercise that through the instrumentality of auto-boxing/unboxing, primitives and their corresponding classes can be used interchangeably in a significant number of scenarios. Auto-boxing does indeed appear to make a number of wrapper class methods that appeared in Java before the auto-boxing feature did obsolete, for example, the method <code>Integer.compareTo</code>.</p>
25. Given an <code>Integer</code> object <code>m</code> with a value of 86, write a program which will increment the value held by the <code>Integer</code> object by 5. Print out the value of the <code>Integer</code> object. <i>Hint: auto-boxing/unboxing.</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer m = 86;         m+=5;         System.out.println(m);     } }</pre>
26. Redo the preceding exercise, without using auto-boxing. Give your assessment of which style you prefer.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer m = 86;         int result = m.intValue() + 5;         m = new Integer(result);         System.out.println(m);     } }</pre> <p><b>Note:</b> clearly using auto-boxing/unboxing is a much</p>

		easier way to process <code>Integer</code> objects that need mathematical operations performed on them.
27.	<p>Using the method <code>Integer.min</code> which takes as parameter two <code>int</code> primitives to compare, print out which is the smaller of the following two <code>Integer</code> objects:</p> <pre>Integer x1 = 86; Integer x2 = 92;</pre> <p>Explain why it is possible to pass to the method <code>Integer.min</code> which has been defined as taking two primitives as parameters two objects of its corresponding wrapper class.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer x1 = 86;         Integer x2 = 92;         Integer smaller = Integer.min(x1, x2);         System.out.println(smaller);     } }</pre> <p><b>Explanation:</b> Due to autoboxing/unboxing, it has been made possible within Java to use primitives and their corresponding classes in places where the other would be expected; the compiler handles the conversion as necessary.</p>
28.	<p>Write a program which determines which is the larger of the following two items:</p> <pre>Integer x1 = new Integer(50); int x2 = 19;</pre>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Integer x1 = 86;         int x2 = 19;         Integer larger = Integer.max(x1, x2);         System.out.println(larger);     } }</pre> <p><b>Note:</b> The objective of this exercise was to show that it is generally acceptable to present a primitive or its corresponding wrapper class where either of them is the expected parameter.</p>
29.	<p>Using the method <code>Integer.decode(String)</code> and given the following object, <code>String s = "76"</code>; write code to convert this <code>String</code> to an <code>int</code>.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String s = "76";         int m = Integer.decode(s);     } }</pre>
30.	<p>Using the method <code>Integer.parseInt(String)</code> and given the following object, <code>String s = "76"</code>; write code to convert this <code>String</code> to an <code>int</code>.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String s = "76";         int m = Integer.parseInt(s);     } }</pre>
31.	<p>Repeat the above exercise for a <code>String s</code> with value “ABCD”. State the results.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String s = "ABCD";         int m = Integer.parseInt(s);     } }</pre> <p><b>Results:</b> The result is an <u>exception</u> named <code>java.lang.NumberFormatException</code>, for clearly “ABCD” is not a number at all. Also, the program stops running at the point of the exception. What an exception is and how it is handled is looked at further in the chapter entitled <i>Exception Handling</i>.</p>	
32.	<p>We have the binary value “111011” passed to us as a <code>String</code>. Using the method <code>Integer.parseInt(value, radix)</code>, convert this <code>String</code> to its corresponding <code>int</code> value in base 10 and print the <code>int</code> out.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String s = "111011";         int numberBase = 2;         int m = Integer.parseInt(s, numberBase);     } }</pre>

## Practice Your Java Level 1

		<pre>         System.out.println(m);     } }  <b>Note:</b> This exercise shows how to convert binary <code>String</code> input to its corresponding decimal value. </pre>
33.	We have the value “100” in base 7 passed to us as a <code>String</code> . Convert this <code>String</code> to its corresponding <code>int</code> value in base 10 and print the <code>int</code> out.  <i>Hint: method <code>Integer.toBinaryString</code></i>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         String s = "100";         int numberBase = 7;         int m = Integer.parseInt(s, numberBase);         System.out.println(m);     } } </pre>
34.	Given an <code>int m</code> with a value of 5, convert this to its binary equivalent, putting the result into a <code>String</code> . Print the <code>String</code> out.  <i>Hint: method <code>Integer.toBinaryString</code></i>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int m = 5;         String result = Integer.toBinaryString(m);         System.out.println(result);     } } </pre>
35.	Write a program, using the method <code>Integer.toHexString</code> , to convert the following <code>int</code> to its hexadecimal value:  <code>int m = 74;</code> Ensure that the letters in the output are in uppercase.	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int m = 74;         String result = Integer.toHexString(m);         result = result.toUpperCase();         System.out.println(result);     } } </pre>
36.	Write a program, using the method <code>Integer.toOctalString</code> , to convert the following <code>int</code> to its equivalent base 8 value:  <code>int m = 74;</code>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int m = 74;         String result = Integer.toOctalString(m);         System.out.println(result);     } } </pre>
37.	Write a program which will calculate and print out the number of “1” bits in the internal representation of the number 227.  <i>Hint: <code>Integer.bitCount</code></i>  <i>(This functionality is useful for computing checksums)</i>	<pre> public class PracticeYourJava {     public static void main(String[] args) {         int m = 227;         int numBits = Integer.bitCount(m);         System.out.println(numBits);     } } </pre>
38.	Explain the reason for the output of the following code snippet:  <code>int y; Integer x = null; y=x;</code>	<p>The following exception results:  <code>java.lang.NullPointerException</code></p> <p>The reason for the exception is because the value of the <code>Integer</code> is <code>null</code>, which means that there is nothing to box to an <code>int</code>! The value <code>null</code> is meaningless in the context of primitives.</p>
39.	Rewrite the code snippet in the preceding exercise to ensure that it no longer triggers an exception.	<pre> int y; Integer x = null; if(x != null)     y=x; </pre>

Class Float		
40.	Write a program which instantiates an object of class <code>Float</code> , with an initial value of 27.53. Show the initialization using the operator <code>new</code> and by autoboxing.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Float f = new Float(27.53);         // OR         //Float f = 27.53;     } }</pre>
41.	I have a given <code>Float</code> with a value of 27.53. Write a program to show the assigning of its value to each of the following: <code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>double</code> .	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Float f = new Float(27.53);         byte b = Float.byteValue(f);         short s = Float.shortValue(f);         int i = Float.intValue(f);         long l = Float.longValue(f);         double d = Float.doubleValue(f);     } }</pre>
42.	Write code to test whether a given <code>float</code> has a value of <code>Float.NaN</code> . Print out “Yes it is NaN” if it indeed has the value of <code>Float.NaN</code> , otherwise, print out “it is not a NaN”. <i>Hint: Float.isNaN</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f = Float.NaN; // set the value for                              // testing purposes         if(Float.isNaN(f))             System.out.println("Yes it is NaN!");         else             System.out.println("No it is finite");     } }</pre>
43.	Write code to test whether a given <code>float</code> (primitive) has a finite value. Print out “Yes it is finite”, if it indeed it is finite, otherwise, print out “it is not finite”. <i>Hint: Float.isFinite</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         float f = Float.NaN; // set the value as desired for testing purposes         if(Float.isFinite(f))             System.out.println("Yes it is finite!");         else             System.out.println("No it is not finite");     } }</pre>
44.	Write code to test whether a given <code>Float</code> has a finite value. Print out “Yes it is finite”, if it indeed it is finite, otherwise, print out “it is not finite”.	<pre>public class PracticeYourJava {     public static void main(String[] args) {         Float f = Float.POSITIVE_INFINITY; // set the value as desired for testing purposes         if(f.isInfinite())             System.out.println("No it is not finite");         else             System.out.println("Yes it is finite!");     } }</pre>
45.	Using the method <code>Float.parseFloat(String)</code> and given the following object, <code>String s = "98.773"</code> ; write code to convert this <code>String</code> to a <code>float</code> .	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String s = "98.773";         float m = Float.parseFloat(s);     } }</pre>
E1	The treatment of the numeric wrapper classes <code>Short</code> , <code>Long</code> and <code>Double</code> are similar to the treatment of <code>Integer</code> and <code>Float</code> shown herein. The reader is urged to peruse these classes.	



# Chapter 10. The Boolean Wrapper Class

The `Boolean` class is the wrapper class for the primitive `boolean`. As certain features in Java do not accept primitives as input (such as Collections, which we will see in later chapters), the `Boolean` class clearly stands in for `boolean` primitives in such cases.

The `Boolean` wrapper class presents the same operations that can be performed on `boolean` primitives, albeit in method form rather than operator form. This chapter presents exercises on this simple wrapper class.

1.	<p>Rewrite the following code using <code>Boolean</code> class objects instead of <code>boolean</code> primitives:</p> <pre>public class PracticeYourJava {     public static void main(String[] args){         boolean a = true, b = false;         System.out.println(a &amp; b);         System.out.println(a   b);         System.out.println(a ^ b);     } }</pre> <p><i>Hint: methods <code>Boolean.LogicalAnd</code>, <code>LogicalOr</code>, <code>LogicalXor</code></i></p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          Boolean a = new Boolean(true);         Boolean b = new Boolean(false);          System.out.println(Boolean.LogicalAnd(a,b));         System.out.println(Boolean.LogicalOr(a,b));         System.out.println(Boolean.LogicalXor(a,b));     } }</pre>
2.	<p>Given the following variable: <code>boolean a = true</code>, assign this to <code>Boolean</code> objects using:</p> <ol style="list-style-type: none"><li>autoboxing</li><li>assignment via a constructor</li></ol>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          boolean a = false;          Boolean obj1 = a; //autoboxing         Boolean obj2 = new Boolean(a);     } }</pre>
3.	<p>Given the following variable: <code>String a = "true"</code>, assign this to a <code>Boolean</code> object.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          String a = "true";         Boolean obj1 = new Boolean(a);     } }</pre>
4.	<p>Given the following variable <code>Boolean obj1 = new Boolean ("false")</code>, assign the value of this variable to a <code>boolean</code> variable.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          Boolean obj1 = new Boolean ("false");         boolean a = obj1; // by autoboxing;         //OR         boolean b = obj1.booleanValue();     } }</pre>
5.	<p>Given the following variable: <code>boolean a = true</code>, assign its value to a <code>String</code> object.</p>	<pre>public class PracticeYourJava {     public static void main(String[] args) {          boolean a = true;         String str = Boolean.toString(a);     } }</pre>



# Chapter 11. Enumerations I

This chapter presents exercises on the concept of enumerations in Java. This chapter is the first of two in this book to present exercises on this topic.

1.	<p>What is an enumeration (<code>enum</code>) and why is it useful? Also show an example of a simple enumeration named <code>Gender</code> that contains the values <code>MALE</code> and <code>FEMALE</code> in that order.</p> <p>In its most basic form, <a href="#">in Java</a>, an <code>enum</code> (short for <i>enumeration</i>) can be described as a type that is assigned (by the programmer) a given <i>ordered set of strings</i> that in the context of an <code>enum</code> can be treated as integer constants, each of whose “value” is simply their name, however they have a recognized order; this order is the order in which they are placed into the enumeration, with the first value having the lowest “order number” and the last having the highest “order number”. (Note though, an advanced feature of enumerations provides us the choice of assigning values, of any type, in any quantity of your choosing to be associated with the named string that represents the enumeration. We will see this in the chapter <i>Enumerations II</i>).</p> <p>Why are enumerations useful? Enumerations are useful because they allow the programmer to use meaningful names in code for related items among which numerical ordering is determined to be desirable. Also, by this numerical ordering, the writing of code for comparisons is simplified. If for example I have the set of months January to December and have to write code to do month order comparisons, I can either write an extensive <code>if/else</code> block to compare the string values of the month names, or I can create an <code>enum</code> that represents each of these months as an ordered set of integer values, an integer comparison clearly being an easier way of doing month order comparisons instead of comparing the strings directly.</p> <p>Here is an example of an enumeration named <code>Gender</code> which defines string values for gender:</p> <pre>public enum Gender{     MALE,     FEMALE; }</pre> <p>The values in this would be used/assigned as <code>Gender.MALE</code> and <code>Gender.FEMALE</code>, assigning them to variables of type <code>Gender</code>.</p> <p>Beyond the basic definition of an enum, <a href="#">in Java, an enumeration is actually a special type of class!</a> It is a descendant of the special class <code>Enum</code>. Being that it is a class, in addition to the ability to define constants in it, it also supports most of the other features of classes; for example, you can create an enumeration class that has methods. This differs from the definition of enumerations in many other programming languages (for example C, C#), where enumerations are not classes and each of the strings in an <code>enum</code> definition represents a single numerical value.</p> <p>Given that in Java an enumeration is actually a class, <a href="#">each of the constants in an enumeration is actually the name of an object of that enumeration class (the definition style is special/different)</a>! This is a very important distinction between enumerations in Java and in many other languages where an enum only represents a number. (The class <code>Enumeration</code> by design presents a unique mechanism for defining its objects, as we have seen the definition of the objects <code>MALE</code> and <code>FEMALE</code> in the enum class <code>Gender</code> above).</p> <p><b>Think of an enum as a class all of whose objects are known by name in advance and the names of all of these objects are listed in the class. The Java runtime can then simply create all of the objects of the enumeration class in question at system initialization time.</b></p> <p>For programmers who are used in other languages to enum entries being an ordered list that is backed by integer representation, the way to achieve the same effect with Java enumerations as in other languages is to either use the ordinality of the entries in the enumeration for comparison, or always use the <code>compareTo</code> method. Exercises on these are presented later.</p> <p>Recall that it was stated earlier that <code>enum</code>'s are special classes and that each of the constants therein actually is an object. We will see in the chapter entitled <i>Enumerations II</i> how to implement methods in enumerations.</p> <p><b>Note:</b> Being a class, an enumeration is defined in its own class file.</p>
----	---

## Practice Your Java Level 1

2.	<p>Define an <code>enum</code> named <code>Gender</code> that has the values <code>MALE</code> and <code>FEMALE</code> in that order.</p> <pre>public enum Gender{     MALE , FEMALE; }</pre>
3.	<p>What is the output of the following code (compiled in the same package as the <code>enum</code> <code>Gender</code> defined above)?</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Gender.MALE);         System.out.println(Gender.FEMALE);     } }</pre>
	<p>MALE FEMALE</p>
4.	<p>When you compile any <code>enum</code> how many objects are created for that <code>enum</code>?</p> <p>The system creates one object per <code>enum</code> entry; recall that an <code>enum</code> is a special type of class and that each of the entries actually represents of an object.</p>
5.	<p>When you compile the earlier defined <code>enum</code> <code>Gender</code>, how many objects of class <code>Gender</code> (for an <code>enum</code> really is a class definition) are created automatically?</p> <p>Two objects are created; one for <code>Gender.MALE</code> and the other for <code>Gender.FEMALE</code>.</p>
6.	<p>When are these object created?</p>
	<p>At program initialization time.</p>
7.	<p><i>Declaring and assigning enums</i></p> <p>In <code>main</code>, assign a value of <code>Gender.MALE</code> to a variable of type <code>Gender</code>. Print the variable out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         Gender gender01 = Gender.MALE; // Observe that we don't need the keyword "new".         System.out.println(gender01);     } }</pre> <p><b>Output:</b> MALE</p>
8.	<p>Explain why the keyword <code>new</code> was not required when assigning <code>Gender.MALE</code> to the variable <code>gender01</code> in the preceding exercise.</p> <p>The reason for why the keyword <code>new</code> is not required is because the system has already created the object <code>Gender.MALE</code> and we are just referencing this object with the variable <code>gender01</code>.</p>
9.	<p>Using the method <code>&lt;enum name&gt;. &lt;enum entry name&gt;.ordinal()</code>, write a program which will print out the <i>position</i> of each of the entries in the enum <code>Gender</code> defined earlier.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println(Gender.MALE.ordinal());         System.out.println(Gender.FEMALE.ordinal());     } }</pre> <p><b>Output:</b> 0 and 1 respectively.</p>
10.	<p>Using the method <code>&lt;enum name&gt;.valueOf(enum String)</code>, print out the value assigned to each of the entries in the enum <code>Gender</code>.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("The value of MALE = " + Gender.valueOf("MALE"));         System.out.println("The value of FEMALE = " + Gender.valueOf("FEMALE"));         //Interestingly, an assigned variable of the class Gender is what is used.     } }</pre> <p><b>Output:</b> MALE and FEMALE respectively.</p> <p><b>Comment:</b> Java simply uses the name of the enum value in question as its value, unless you cause it to be otherwise.</p>

11.	<p>In the enum <code>Gender</code>, swap the positions of the entries <code>MALE</code> and <code>FEMALE</code> and rerun the code in the solution to exercise 9. Comment on the difference in what is output.</p>
	<p>The enum now looks as follows:</p> <pre>public enum Gender{     FEMALE;     MALE; }</pre>
	<p><b>Output:</b> On re-running the code, the output is the following: <code>1</code> and <code>0</code> respectively.  <b>Comments:</b> As could be expected, the positional values are reversed from what they were before. If your code has a particular need for constant ordinality in the elements of the enum in question, then ensure that the order of the elements are not changed in the enum by anyone.</p>
12.	<p>Define an enum named <code>Day</code>; enter the days of the week from Sunday to Saturday into it in order.</p>
	<pre>public enum Day{     SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;</pre>
13.	<p>Using the <code>values</code> method of the class <code>enum</code>, loop through the enum <code>Day</code> using an enhanced for loop and print each of its elements and their respective values out.</p>
	<p><i>Hint: <code>Day.values()</code></i></p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         for(Day x:Day.values()){             System.out.println(x + " : its value = " + x.valueOf(x.toString()));         }     } }</pre>
14.	<p>Write a program which looks at the values of two <code>enum</code> variables of type <code>Day</code> (the <code>enum</code> defined in exercise 12) and determines which day is the earlier one. If they are the same, then state so.</p>
	<p>Test your code using different combinations of days.</p>
	<p><i>Hint: use the <code>compareTo</code> method of enum instances to implement this.</i></p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {          Day day01 = Day.SUNDAY;         Day day02 = Day.SATURDAY;          if(day01.compareTo(day02) &lt; 0)             System.out.println(day01 + " is before " + day02);         else if(day01.compareTo(day02) &gt; 0)             System.out.println(day02 + " is before " + day01);         else             System.out.println("The same day was specified!");     } }</pre>
	<p><b>Note:</b> Clearly, in scenarios where the order of the entries in an <code>enum</code> in your program is important to you, you must ensure that the order of the entries in the <code>enum</code> are never changed otherwise your code will be affected.</p>
15.	<p>Rewrite the preceding program directly using the ordinality of the <code>enum</code> values to do the comparison.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {          Day day01 = Day.SUNDAY;         Day day02 = Day.SATURDAY;          if(day01.ordinal() &lt; day02.ordinal())             System.out.println(day01 + " is before " + day02);         else if(day01.ordinal() &gt; day02.ordinal())             System.out.println(day02 + " is before " + day01);         else             System.out.println("The same day was specified!");     } }</pre>

## Practice Your Java Level 1

	<p>}</p> <p><b>Note:</b> As with using the <code>compareTo</code> method, you must ensure that the order of the entries in the enum are never changed because your code will be affected.</p>
16.	<p>How is an enumeration different from directly defining constants in a class?</p> <p>An enumeration is different in the following ways:</p> <ol style="list-style-type: none"> <li>1. An enumeration has ordering for the elements that it represents, thus facilitating comparison between elements.</li> <li>2. Enumerations present a clearly defined related set of ordered items.</li> <li>3. An enumeration is a class and the enumerations are actually objects. Being a class, supporting methods, fields, interfaces and other valid class elements can be added directly to the enumeration to ensure that the enumeration class in question clearly encapsulates everything that pertains to the enumeration in question.</li> </ol>
17.	<p>I have different positions within a company. Put the listed positions in the listed order order into an <code>enum</code> named <code>Position</code>: Staff1, Staff2, Staff3, Staff4, Manager, SeniorManager, Director, VicePresident and CEO.</p> <pre>enum Position{     Staff1, Staff2, Staff3, Staff4, Manager, SeniorManager, Director, VP, CEO; }</pre>
18.	<p>There are 10 salary bands in this company. The bands are listed as <code>BAND1...BAND10</code>, with <code>BAND10</code> being the highest band. Put them all in ascending order into an enum named <code>SalaryBand</code>.</p> <pre>enum SalaryBand{     BAND1, BAND2, BAND3, BAND4, BAND5, BAND6, BAND7, BAND8, BAND9, BAND10; }</pre>
19.	<p>I have permanent employees and contractors. Create an <code>enum</code> named <code>EmploymentType</code> to represent the employee types <code>permanent</code> and <code>contractor</code>.</p> <pre>enum EmploymentType {     PERMANENT, CONTRACTOR; }</pre>
20.	<p>Create an <code>enum</code> named <code>VehicleTypes</code> that supports the following vehicle types:  <code>bicycle, motorbike, car, truck, boat, train, trailer, airplane</code>.</p> <pre>enum VehicleType {     BICYCLE, MOTORBIKE, CAR, TRUCK, BOAT, TRAIN, TRAILER, AIRPLANE; }</pre>
E1	<p>Please study the constants in the following Java enumerations:</p> <ol style="list-style-type: none"> <li>1. <code>RoundingMode</code></li> <li>2. <code>Month</code></li> <li>3. <code>DayOfWeek</code></li> <li>4. <code>TextStyle</code></li> <li>5. <code>StandardOpenOption</code></li> </ol>

# Chapter 12. Number Handling III – BigInteger & BigDecimal

Java provides the class `BigInteger` to handle integer values which exceed the size of a `long` and the class `BigDecimal` to handle decimal values with proper precision. The exercises in this chapter address these two classes.

This chapter is the third of five chapters in this book on number handling in Java.

<b>BigInteger</b>	
1.	<p>Explain the reason for the existence of the class <code>BigInteger</code>.</p> <p>The class <code>BigInteger</code> exists to facilitate the representation in Java of integer values which exceed the range of the largest integer primitive/class, <code>long/Long</code>. The primitive type <code>long</code> can actually be represented natively by most modern CPUs on which Java is run; any integer value that exceeds that range has to be implemented in software as <code>BigInteger</code> has been.</p>
2.	<p>Create a program which will initialize an object of class <code>BigInteger</code> to the value <code>Long.MAX_VALUE</code>.</p> <p>A <code>BigInteger</code> cannot be directly initialized with an integer value, but rather with the string representation of that value. Therefore, we have to convert the desired value <code>Long.MAX_VALUE</code> to a string. We first have to convert the value of <code>Long.MAX_VALUE</code> (which is a <code>long</code>) to a <code>Long</code> object since the object permits us to convert its contents to a string.</p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         Long long_max = Long.MAX_VALUE;         BigInteger bigInt01 = new BigInteger(long_max.toString());         System.out.println(bigInt01);     } }</pre>
3.	<p>Enhance the solution to the preceding exercise by incrementing the value held by the variable <code>bigInt01</code> by 5 and then print the resulting out.</p> <p><i>Hint: BigInteger.add</i></p> <p>A <code>BigInteger</code> can only be directly added to by an object of class <code>BigInteger</code> (nothing smaller as at this version of Java), therefore we have to:</p> <p>(1) convert the primitive number that we seek to add to an object of its wrapper class, (2) convert that object to its <code>String</code> representation using its <code>toString</code> method and then, (3) use that <code>String</code> to create an object of class <code>BigInteger</code>, which you now add to the original <code>BigInteger</code>.</p> <p>It might seem like a lot of steps, but it really isn't. Let's go!</p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         Long long_max = Long.MAX_VALUE;         int numberToAdd = 5;         Integer numberToAddL = Integer(numberToAdd);         String numberToAddS = numberToAddL.toString();         BigInteger numberToAddBI = new BigInteger(numberToAddS);         BigInteger bigInt01 = new BigInteger(long_max.toString());         bigInt01.add(numberToAddBI);         System.out.println(bigInt01);     } }</pre>

## Practice Your Java Level 1

Or more succinctly we can write this in the following manner:

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        Long long_max = Long.MAX_VALUE;
        int numberToAdd = 5;
        BigInteger bigInt01 = new BigInteger(long_max.toString());
        bigInt01 = bigInt01.add(new BigInteger(((Integer)(numberToAdd)).toString()));
        // Explanation: Okay, here's the explanation of this complex looking compact line:
        // do pay attention to the braces, we'll enlarge them below for clarity:
        // bigInt01 = bigInt01.add( new BigInteger( ((Integer)(numberToAdd)).toString() ) );
        // (1) We via auto-boxing, cast the int variable numberToAdd to an object of
        //      class Integer; so we now have an object in there.
        // (2) We use the toString method of the class Integer on that object.
        // (3) We need a BigInteger object to add to the 1st BigInteger object, so we
        //      create one, unnamed, inside the braces by saying new BigInteger(contents).
        System.out.println(bigInt01);
    }
}
```

4. Write a program which will print out the absolute value of the integer value -799223372036854776808.

*Hint: BigInteger.abs*

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        BigInteger bigInt01 = new BigInteger("-799223372036854776808");
        System.out.println(bigInt01.abs());
    }
}
```

5. Print out the value of 98765432109876543210 to the power of Integer.MAX\_INT. Print the result out.

*Hint: BigInteger.pow*

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        BigInteger bigInt01 = new BigInteger("98765432109876543210");
        BigInteger bigInt03 = bigInt01.pow(Integer.MAX_INT);
        System.out.println(bigInt01 + " * " + bigInt02 + " = " + bigInt03);
    }
}
```

6. Multiply the value 855223372036854776234 by 98765432109876543210 and put it into a different object of class BigInteger.

Print out the returned value in the following fashion: *The result of <value<sub>1</sub>> \* <value<sub>2</sub>> = <result>*

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        BigInteger bigInt01 = new BigInteger("855223372036854776234");
        BigInteger bigInt02 = new BigInteger("98765432109876543210");
        BigInteger bigInt03 = bigInt01.multiply(bigInt02);
        System.out.println(bigInt01 + " * " + bigInt02 + " = " + bigInt03);
    }
}
```

7. Using the same numbers as in the preceding exercise, divide the first value by the second, determining the whole part and the remainder.

Write out the result in the following manner: *The result of <value<sub>1</sub>> / <value<sub>2</sub>> = <result> REM <remainder>*.  
*Hint: BigInteger.divideAndRemainder*

	<pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bigInt01 = new BigInteger("855223372036854776234");         BigInteger bigInt02 = new BigInteger("98765432109876543210");         BigInteger[] result = bigInt01.divideAndRemainder(bigInt02); //this method returns an array         System.out.println(bigInt01 + " / " + bigInt02 + " = " + result[0] + " REM " + result[1]);     } }</pre>
8.	<p>Divide the first value in the preceding exercise by the second, determining only the whole part of the calculation.  <i>Hint: BigInteger.divide</i></p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bigInt01 = new BigInteger("855223372036854776234");         BigInteger bigInt02 = new BigInteger("98765432109876543210");         BigInteger result = bigInt01.divide(bigInt02);         System.out.println(bigInt01 + " / " + bigInt02 + " = " + result);     } }</pre>
9.	<p>Divide the first value in the preceding exercise by the second, determining and printing only the remainder of the calculation.  <i>Hint: BigInteger.mod</i></p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bigInt01 = new BigInteger("855223372036854776234");         BigInteger bigInt02 = new BigInteger("98765432109876543210");         BigInteger result = bigInt01.mod(bigInt02);         System.out.println(bigInt01 + " mod " + bigInt02 + " = " + result);     } }</pre>
10.	<p>Find and print out the greatest common denominator of the two numbers from the preceding exercise.</p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bigInt01 = new BigInteger("855223372036854776234");         BigInteger bigInt02 = new BigInteger("98765432109876543210");         BigInteger result = bigInt01.gcd(bigInt02);         System.out.println("gcd = " + result );     } }</pre>
11.	<p>I have the following hexadecimal string “FF19146BAAC9BA17DEA914B”. Print out its equivalent decimal value.  <i>Hint: BigInteger(String, radix)</i></p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         String hexString = "FF19146BAAC9BA17DEA914B";         BigInteger bigInt01 = new BigInteger(hexString, 16);         System.out.println("The equivalent decimal = " + bigInt01);     } }</pre>
12.	<p>I have the following base 7 number: 67. Print out its equivalent decimal value.</p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {</pre>

## Practice Your Java Level 1

	<pre>         String numberString = "67";         BigInteger bigInt01 = new BigInteger(numberString, 7);         System.out.println("The equivalent decimal = " + bigInt01);     } } </pre>
13.	<p>I have a <code>BigInteger</code> object with value 57. Put this value into a <code>byte</code> (primitive).</p> <p><i>Hint: BigInteger.byteValue</i></p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         String numberString = "77";         BigInteger bigInt01 = new BigInteger(numberString);         byte newByte = bigInt01.byteValue();     } } </pre>
14.	<p>Put the value of the <code>BigInteger</code> object from the preceding exercise into a <code>long</code> (primitive).</p> <p><i>Hint: BigInteger.LongValue</i></p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         String numberString = "77";         BigInteger bigInt01 = new BigInteger(numberString);         long newLong = bigInt01.longValue();     } } </pre>
15.	<p>Put the value of the <code>BigInteger</code> object from the preceding exercise into a <code>float</code> (primitive).</p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bigInt01 = new BigInteger("77");         float newFloat = bigInt01.floatValue();     } } </pre>
16.	<p>Print out and comment on the value obtained when we attempt to put a <code>BigInteger</code> of value “7555599991234134133” into a variable of type <code>long</code>. Print out the received value.</p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         String numberString = "7555599991234134133";         BigInteger bigInt01 = new BigInteger(numberString);         long newLong = bigInt01.longValue();         System.out.println("long = " + newLong);     } } </pre> <p><b>Comments:</b> Clearly the value of the <code>long</code> does not match the <code>BigInteger</code> value. What has happened is that the value put into the <code>long</code> is the result of an overflow computation.</p> <p>Conclusion: If an underflow or overflow situation is not desired, then use the appropriate <code>BigInteger.Exact</code> method to attempt to put the desired value into the target variable.</p>
17.	<p>Attempt to put a <code>BigInteger</code> of value “7555599991234134133” into a variable of type <code>long</code>. Ensure that the value is not put into the variable if it exceeds the range of <code>long</code>. State the results.</p> <p><i>Hint: BigInteger.LongValueExact</i></p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         String numberString = "7555599991234134133";         BigInteger bigInt01 = new BigInteger(numberString);     } } </pre>

	<pre>         long newLong = bigInt01.longValueExact();         System.out.println("long = " + newLong);     } }  <b>Result:</b> An exception, <a href="#">java.lang.ArithmaticException</a> is generated since the value is outside the range of the type long. </pre>
18.	<p>I have the following two hexadecimal numbers: FFFF82312DFADE1234ED and ABCD32432576879909F. Perform a logical AND of these numbers and print the result out in decimal, hexadecimal and binary. Ensure that the hexadecimal number is in uppercase.</p> <p><i>Hint: methods BigInteger.toString(radix), BigInteger.and</i></p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bigInt01 = new BigInteger("FFFF82312DFADE1234ED",16);         BigInteger bigInt02 = new BigInteger("ABCD32432576879909F",16);         BigInteger bigInt03 = bigInt01.and(bigInt02);          System.out.println("result (decimal) = " + bigInt03.toString(10));         System.out.println("result (hex)      = " + bigInt03.toString(16).toUpperCase());         System.out.println("result (binary)   = " + bigInt03.toString(2));     } } </pre>

<b>BigDecimal</b>	
19.	<p>Why does the class <b>BigDecimal</b> exist?</p> <p>The class <b>BigDecimal</b> exists to support numbers of arbitrary decimal precision. The precision and rounding of the value contained in objects of class <b>BigDecimal</b> is user determinable.</p>
20.	<p>Please run the following program and note the output for each respective <b>BigDecimal</b> object. The code and its output will be referred to in later exercises.</p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger bi01 = new BigInteger(((Long)(1570335L)).toString());          BigDecimal bd01 = new BigDecimal(bi01);           System.out.println(bd01);         BigDecimal bd02 = new BigDecimal(bi01, 5);         System.out.println(bd02);         BigDecimal bd03 = new BigDecimal(bi01, 2);         System.out.println(bd03);         BigDecimal bd04 = new BigDecimal(bi01, -2);        System.out.println(bd04);         BigDecimal bd05 = new BigDecimal(bi01, -10);       System.out.println(bd05);     } } </pre> <p>Output (respectively):</p> <ol style="list-style-type: none"> <li>1. 1570335</li> <li>2. 15.70335</li> <li>3. 15703.35</li> <li>4. 1.570335E+8</li> <li>5. 1.570335E+16</li> </ol>
21.	<p>Explain the concept of “scale” as pertains to the initialization of <b>BigDecimal</b> objects. Use the code and output from the preceding exercise as reference examples in your solution.</p> <p>It is understood that the user of a <b>BigDecimal</b> may want to express entered numbers as a mantissa/significand with an exponent. This is essentially what the concept of scale in <b>BigDecimal</b> entails. Therefore, facility is made in some of the constructors of <b>BigDecimal</b> to present the desired number as mantissa and exponent. The mantissa is expressed as a <b>BigInteger</b>. The exponent is expressed as an integer.</p> <p>The resulting number = mantissa <math>\times 10^{\text{exponent}}</math>. Therefore, if the entered exponent is a positive number, we are</p>

## Practice Your Java Level 1

	<p>actually indicating a negative exponent and if the entered exponent is a negative number we are actually indicating a positive exponent, as mathematically <math>-- = +</math>. This can be observed in the output of the preceding exercise which yielded <code>BigDecimal</code> values according the preceding logic in the following fashion:</p> <table border="0"> <tr> <td>no entry for the exponent</td><td><math>\rightarrow</math> the <code>BigDecimal</code> value is the same as the entered number</td></tr> <tr> <td>a positive entry, 5</td><td><math>\rightarrow</math> therefore a negative exponent; the resultant exponent = <math>x10^{-5}</math></td></tr> <tr> <td>a positive entry, 2</td><td><math>\rightarrow</math> therefore a negative exponent; the resultant exponent = <math>x 10^{-2}</math></td></tr> <tr> <td>a negative entry, -2</td><td><math>\rightarrow</math> therefore a positive exponent, the resultant exponent = <math>x10^{(-2)} = x10^2</math></td></tr> <tr> <td>another negative entry, -10</td><td><math>\rightarrow</math> therefore again a positive exponent, the resultant exponent = <math>x10^{(-10)} = x10^{10}</math></td></tr> </table>	no entry for the exponent	$\rightarrow$ the <code>BigDecimal</code> value is the same as the entered number	a positive entry, 5	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x10^{-5}$	a positive entry, 2	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x 10^{-2}$	a negative entry, -2	$\rightarrow$ therefore a positive exponent, the resultant exponent = $x10^{(-2)} = x10^2$	another negative entry, -10	$\rightarrow$ therefore again a positive exponent, the resultant exponent = $x10^{(-10)} = x10^{10}$					
no entry for the exponent	$\rightarrow$ the <code>BigDecimal</code> value is the same as the entered number															
a positive entry, 5	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x10^{-5}$															
a positive entry, 2	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x 10^{-2}$															
a negative entry, -2	$\rightarrow$ therefore a positive exponent, the resultant exponent = $x10^{(-2)} = x10^2$															
another negative entry, -10	$\rightarrow$ therefore again a positive exponent, the resultant exponent = $x10^{(-10)} = x10^{10}$															
22.	<p>Explain succinctly the concept of “precision” with respect to <code>BigDecimal</code>.</p> <p>The concept of precision for a <code>BigDecimal</code>, is simply the definition for the <code>BigDecimal</code> object at hand of how many significant digits it is supposed to have. The precision pertains to the total count of significant digits in the whole number portion of the number and in the fractional part of the number. For example, the number 8.3 has two significant digits, 0.83 has 2 significant digits, 1.35 has three significant digits, 0.135 has 3 significant digits.</p>															
23.	<p>Explain the concept of a “rounding mode” as applies to objects of the class <code>BigDecimal</code>.</p> <p>The concept of the “rounding mode” pertains to the determination of how the least significant digit of a number whose precision we have specified should be determined when the raw number at hand has more significant digits than our desired precision for it.</p> <p>The reader is likely familiar with one particular rounding mode; that is “rounding up”, i.e. any digit that is greater than or equal to 5 results in a value of 1 being added to its adjacent superior digit. In Java currently there are eight rounding modes for <code>BigDecimal</code>.</p>															
24.	<p>List and describe with examples the eight different rounding modes present in the enumeration <code>RoundingMode</code>. Your examples should pertain to the rounding of both positive and negative numbers of target precision = 2.</p> <table border="1"> <tr> <td>CEILING</td><td>The value is rounded to the nearest positive value.</td><td> <p>8.73 <math>\rightarrow</math> 8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, CEILING nevertheless moves the number up.  -8.73 <math>\rightarrow</math> -8.7 Same logic; nearest higher number for that precision.</p> </td></tr> <tr> <td>DOWN</td><td>Round towards 0. This actually implies that positive numbers reduce in value and negative numbers increase in value! For this think like this “<u>zero</u> is seen as DOWN”.</td><td> <p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, DOWN nevertheless moves the number towards 0.  -8.78 <math>\rightarrow</math> -8.7 Towards zero as stated!</p> </td></tr> <tr> <td>FLOOR</td><td>The value is rounded to the nearest lower value for the specified precision.</td><td> <p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, FLOOR moves it lower. In a sense, for positive values, FLOOR truncates the excess digits.  -8.73 <math>\rightarrow</math> -8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, FLOOR moves it lower.</p> </td></tr> <tr> <td>HALF_DOWN</td><td>Rounds towards the nearest neighbor for the specified precision. The value 5 is counted among the lower half of numbers to be rounded. The values to be rounded can be thought of as 2 rounding groups: 1-5 and 6-9. Contrast with HALF_UP. Think of “down” as meaning “to a lower digit”.</td><td> <p>8.78 <math>\rightarrow</math> 8.8  8.75 <math>\rightarrow</math> 8.7  -8.74 <math>\rightarrow</math> -8.7  -8.77 <math>\rightarrow</math> -8.8  <b>-8.75 <math>\rightarrow</math> -8.7</b> Note the difference in the handling for 5 when it is positive versus when it is negative.  Interestingly it isn't moved to a lower number when negative.</p> </td></tr> <tr> <td>HALF_EVEN</td><td>Rounds towards the nearest neighbor for the specified precision, however</td><td> <p>8.74 <math>\rightarrow</math> 8.7  <b>8.78 <math>\rightarrow</math> 8.8</b></p> </td></tr> </table>	CEILING	The value is rounded to the nearest positive value.	<p>8.73 <math>\rightarrow</math> 8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, CEILING nevertheless moves the number up.  -8.73 <math>\rightarrow</math> -8.7 Same logic; nearest higher number for that precision.</p>	DOWN	Round towards 0. This actually implies that positive numbers reduce in value and negative numbers increase in value! For this think like this “ <u>zero</u> is seen as DOWN”.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, DOWN nevertheless moves the number towards 0.  -8.78 <math>\rightarrow</math> -8.7 Towards zero as stated!</p>	FLOOR	The value is rounded to the nearest lower value for the specified precision.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, FLOOR moves it lower. In a sense, for positive values, FLOOR truncates the excess digits.  -8.73 <math>\rightarrow</math> -8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, FLOOR moves it lower.</p>	HALF_DOWN	Rounds towards the nearest neighbor for the specified precision. The value 5 is counted among the lower half of numbers to be rounded. The values to be rounded can be thought of as 2 rounding groups: 1-5 and 6-9. Contrast with HALF_UP. Think of “down” as meaning “to a lower digit”.	<p>8.78 <math>\rightarrow</math> 8.8  8.75 <math>\rightarrow</math> 8.7  -8.74 <math>\rightarrow</math> -8.7  -8.77 <math>\rightarrow</math> -8.8  <b>-8.75 <math>\rightarrow</math> -8.7</b> Note the difference in the handling for 5 when it is positive versus when it is negative.  Interestingly it isn't moved to a lower number when negative.</p>	HALF_EVEN	Rounds towards the nearest neighbor for the specified precision, however	<p>8.74 <math>\rightarrow</math> 8.7  <b>8.78 <math>\rightarrow</math> 8.8</b></p>
CEILING	The value is rounded to the nearest positive value.	<p>8.73 <math>\rightarrow</math> 8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, CEILING nevertheless moves the number up.  -8.73 <math>\rightarrow</math> -8.7 Same logic; nearest higher number for that precision.</p>														
DOWN	Round towards 0. This actually implies that positive numbers reduce in value and negative numbers increase in value! For this think like this “ <u>zero</u> is seen as DOWN”.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, DOWN nevertheless moves the number towards 0.  -8.78 <math>\rightarrow</math> -8.7 Towards zero as stated!</p>														
FLOOR	The value is rounded to the nearest lower value for the specified precision.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, FLOOR moves it lower. In a sense, for positive values, FLOOR truncates the excess digits.  -8.73 <math>\rightarrow</math> -8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, FLOOR moves it lower.</p>														
HALF_DOWN	Rounds towards the nearest neighbor for the specified precision. The value 5 is counted among the lower half of numbers to be rounded. The values to be rounded can be thought of as 2 rounding groups: 1-5 and 6-9. Contrast with HALF_UP. Think of “down” as meaning “to a lower digit”.	<p>8.78 <math>\rightarrow</math> 8.8  8.75 <math>\rightarrow</math> 8.7  -8.74 <math>\rightarrow</math> -8.7  -8.77 <math>\rightarrow</math> -8.8  <b>-8.75 <math>\rightarrow</math> -8.7</b> Note the difference in the handling for 5 when it is positive versus when it is negative.  Interestingly it isn't moved to a lower number when negative.</p>														
HALF_EVEN	Rounds towards the nearest neighbor for the specified precision, however	<p>8.74 <math>\rightarrow</math> 8.7  <b>8.78 <math>\rightarrow</math> 8.8</b></p>														
	<p>actually indicating a negative exponent and if the entered exponent is a negative number we are actually indicating a positive exponent, as mathematically <math>-- = +</math>. This can be observed in the output of the preceding exercise which yielded <code>BigDecimal</code> values according the preceding logic in the following fashion:</p> <table border="0"> <tr> <td>no entry for the exponent</td><td><math>\rightarrow</math> the <code>BigDecimal</code> value is the same as the entered number</td></tr> <tr> <td>a positive entry, 5</td><td><math>\rightarrow</math> therefore a negative exponent; the resultant exponent = <math>x10^{-5}</math></td></tr> <tr> <td>a positive entry, 2</td><td><math>\rightarrow</math> therefore a negative exponent; the resultant exponent = <math>x 10^{-2}</math></td></tr> <tr> <td>a negative entry, -2</td><td><math>\rightarrow</math> therefore a positive exponent, the resultant exponent = <math>x10^{(-2)} = x10^2</math></td></tr> <tr> <td>another negative entry, -10</td><td><math>\rightarrow</math> therefore again a positive exponent, the resultant exponent = <math>x10^{(-10)} = x10^{10}</math></td></tr> </table>	no entry for the exponent	$\rightarrow$ the <code>BigDecimal</code> value is the same as the entered number	a positive entry, 5	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x10^{-5}$	a positive entry, 2	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x 10^{-2}$	a negative entry, -2	$\rightarrow$ therefore a positive exponent, the resultant exponent = $x10^{(-2)} = x10^2$	another negative entry, -10	$\rightarrow$ therefore again a positive exponent, the resultant exponent = $x10^{(-10)} = x10^{10}$					
no entry for the exponent	$\rightarrow$ the <code>BigDecimal</code> value is the same as the entered number															
a positive entry, 5	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x10^{-5}$															
a positive entry, 2	$\rightarrow$ therefore a negative exponent; the resultant exponent = $x 10^{-2}$															
a negative entry, -2	$\rightarrow$ therefore a positive exponent, the resultant exponent = $x10^{(-2)} = x10^2$															
another negative entry, -10	$\rightarrow$ therefore again a positive exponent, the resultant exponent = $x10^{(-10)} = x10^{10}$															
22.	<p>Explain succinctly the concept of “precision” with respect to <code>BigDecimal</code>.</p> <p>The concept of precision for a <code>BigDecimal</code>, is simply the definition for the <code>BigDecimal</code> object at hand of how many significant digits it is supposed to have. The precision pertains to the total count of significant digits in the whole number portion of the number and in the fractional part of the number. For example, the number 8.3 has two significant digits, 0.83 has 2 significant digits, 1.35 has three significant digits, 0.135 has 3 significant digits.</p>															
23.	<p>Explain the concept of a “rounding mode” as applies to objects of the class <code>BigDecimal</code>.</p> <p>The concept of the “rounding mode” pertains to the determination of how the least significant digit of a number whose precision we have specified should be determined when the raw number at hand has more significant digits than our desired precision for it.</p> <p>The reader is likely familiar with one particular rounding mode; that is “rounding up”, i.e. any digit that is greater than or equal to 5 results in a value of 1 being added to its adjacent superior digit. In Java currently there are eight rounding modes for <code>BigDecimal</code>.</p>															
24.	<p>List and describe with examples the eight different rounding modes present in the enumeration <code>RoundingMode</code>. Your examples should pertain to the rounding of both positive and negative numbers of target precision = 2.</p> <table border="1"> <tr> <td>CEILING</td><td>The value is rounded to the nearest positive value.</td><td> <p>8.73 <math>\rightarrow</math> 8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, CEILING nevertheless moves the number up.  -8.73 <math>\rightarrow</math> -8.7 Same logic; nearest higher number for that precision.</p> </td></tr> <tr> <td>DOWN</td><td>Round towards 0. This actually implies that positive numbers reduce in value and negative numbers increase in value! For this think like this “<u>zero</u> is seen as DOWN”.</td><td> <p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, DOWN nevertheless moves the number towards 0.  -8.78 <math>\rightarrow</math> -8.7 Towards zero as stated!</p> </td></tr> <tr> <td>FLOOR</td><td>The value is rounded to the nearest lower value for the specified precision.</td><td> <p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, FLOOR moves it lower. In a sense, for positive values, FLOOR truncates the excess digits.  -8.73 <math>\rightarrow</math> -8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, FLOOR moves it lower.</p> </td></tr> <tr> <td>HALF_DOWN</td><td>Rounds towards the nearest neighbor for the specified precision. The value 5 is counted among the lower half of numbers to be rounded. The values to be rounded can be thought of as 2 rounding groups: 1-5 and 6-9. Contrast with HALF_UP. Think of “down” as meaning “to a lower digit”.</td><td> <p>8.78 <math>\rightarrow</math> 8.8  8.75 <math>\rightarrow</math> 8.7  -8.74 <math>\rightarrow</math> -8.7  -8.77 <math>\rightarrow</math> -8.8  <b>-8.75 <math>\rightarrow</math> -8.7</b> Note the difference in the handling for 5 when it is positive versus when it is negative.  Interestingly it isn't moved to a lower number when negative.</p> </td></tr> <tr> <td>HALF_EVEN</td><td>Rounds towards the nearest neighbor for the specified precision, however</td><td> <p>8.74 <math>\rightarrow</math> 8.7  <b>8.78 <math>\rightarrow</math> 8.8</b></p> </td></tr> </table>	CEILING	The value is rounded to the nearest positive value.	<p>8.73 <math>\rightarrow</math> 8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, CEILING nevertheless moves the number up.  -8.73 <math>\rightarrow</math> -8.7 Same logic; nearest higher number for that precision.</p>	DOWN	Round towards 0. This actually implies that positive numbers reduce in value and negative numbers increase in value! For this think like this “ <u>zero</u> is seen as DOWN”.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, DOWN nevertheless moves the number towards 0.  -8.78 <math>\rightarrow</math> -8.7 Towards zero as stated!</p>	FLOOR	The value is rounded to the nearest lower value for the specified precision.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, FLOOR moves it lower. In a sense, for positive values, FLOOR truncates the excess digits.  -8.73 <math>\rightarrow</math> -8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, FLOOR moves it lower.</p>	HALF_DOWN	Rounds towards the nearest neighbor for the specified precision. The value 5 is counted among the lower half of numbers to be rounded. The values to be rounded can be thought of as 2 rounding groups: 1-5 and 6-9. Contrast with HALF_UP. Think of “down” as meaning “to a lower digit”.	<p>8.78 <math>\rightarrow</math> 8.8  8.75 <math>\rightarrow</math> 8.7  -8.74 <math>\rightarrow</math> -8.7  -8.77 <math>\rightarrow</math> -8.8  <b>-8.75 <math>\rightarrow</math> -8.7</b> Note the difference in the handling for 5 when it is positive versus when it is negative.  Interestingly it isn't moved to a lower number when negative.</p>	HALF_EVEN	Rounds towards the nearest neighbor for the specified precision, however	<p>8.74 <math>\rightarrow</math> 8.7  <b>8.78 <math>\rightarrow</math> 8.8</b></p>
CEILING	The value is rounded to the nearest positive value.	<p>8.73 <math>\rightarrow</math> 8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, CEILING nevertheless moves the number up.  -8.73 <math>\rightarrow</math> -8.7 Same logic; nearest higher number for that precision.</p>														
DOWN	Round towards 0. This actually implies that positive numbers reduce in value and negative numbers increase in value! For this think like this “ <u>zero</u> is seen as DOWN”.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, DOWN nevertheless moves the number towards 0.  -8.78 <math>\rightarrow</math> -8.7 Towards zero as stated!</p>														
FLOOR	The value is rounded to the nearest lower value for the specified precision.	<p>8.78 <math>\rightarrow</math> 8.7. Note that even though the number in the 2<sup>nd</sup> decimal place is greater than 5, FLOOR moves it lower. In a sense, for positive values, FLOOR truncates the excess digits.  -8.73 <math>\rightarrow</math> -8.8. Note that even though the number in the 2<sup>nd</sup> decimal place is less than 5, FLOOR moves it lower.</p>														
HALF_DOWN	Rounds towards the nearest neighbor for the specified precision. The value 5 is counted among the lower half of numbers to be rounded. The values to be rounded can be thought of as 2 rounding groups: 1-5 and 6-9. Contrast with HALF_UP. Think of “down” as meaning “to a lower digit”.	<p>8.78 <math>\rightarrow</math> 8.8  8.75 <math>\rightarrow</math> 8.7  -8.74 <math>\rightarrow</math> -8.7  -8.77 <math>\rightarrow</math> -8.8  <b>-8.75 <math>\rightarrow</math> -8.7</b> Note the difference in the handling for 5 when it is positive versus when it is negative.  Interestingly it isn't moved to a lower number when negative.</p>														
HALF_EVEN	Rounds towards the nearest neighbor for the specified precision, however	<p>8.74 <math>\rightarrow</math> 8.7  <b>8.78 <math>\rightarrow</math> 8.8</b></p>														

		numbers right in the middle (i.e. ending with 5) are rounded to the nearest even number.	8.75 → 8.8 -8.74 → -8.7 -8.77 → -8.8 -8.75 → -8.8
	HALF_UP	Rounds towards the nearest neighbor for the specified precision. The value 5 is rounded up (think of “up” as meaning “to a higher digit”). <i>Note:</i> This corresponds to the normal rounding up mechanism that is well known.	8.74 → 8.7 8.75 → 8.8 8.78 → 8.8 -8.74 → -8.7 -8.77 → -8.8 -8.75 → -8.8
	UP	Means round up to the next highest digit, always. This holds equally for both positive and negative numbers.	8.73 → 8.8 8.74 → 8.8 8.75 → 8.8 8.78 → 8.8 -8.73 → 8.8 -8.74 → -8.8 -8.77 → -8.8
	UNNECESSARY	Means “Do not bother attempting to round the number, I the programmer will always ensure that it has the right precision”. The implications of this are that if you enter a number of higher precision an exception WILL be generated.	
25.	Explain the concept of a <code>MathContext</code> object.  A <code>MathContext</code> object is a configuration object for a <code>BigDecimal</code> object. It facilitates in one single object the specification of the number of significant digits (precision) and the rounding mechanism to be applied to the <code>BigDecimal</code> that it configures. The <code>MathContext</code> object is passed as a variable to a constructor for a <code>BigDecimal</code> .		
26.	Explain what the following code statement means: <code>MathContext mc01 = new MathContext(4, RoundingMode.HALF_UP);</code>  The code statement is saying, create a <code>MathContext</code> object which specifies a precision of four significant digits and a <code>RoundingMode</code> of <code>HALF_UP</code> .		
27.	Write a program which will instantiate a <code>BigDecimal</code> object <code>bd01</code> with the string “1455734”, with the <code>MathContext</code> object in the preceding exercise. Print the resulting number out.  <code>import java.math.*;</code>  <code>public class PracticeYourJava {</code> <code>public static void main(String[] args) {</code> <code>MathContext mc01 = new MathContext(4, RoundingMode.HALF_UP);</code> <code>BigInteger bi01 = new BigInteger("1455734");</code> <code>BigDecimal bd01 = new BigDecimal(bi01, mc01);</code> <code>System.out.println(bd01);</code> } }		
	In more compact form:  <code>import java.math.*;</code>  <code>public class PracticeYourJava {</code> <code>public static void main(String[] args) {</code> <code>MathContext mc01 = new MathContext(4, RoundingMode.HALF_UP);</code> <code>BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), mc01);</code> <code>System.out.println(bd01);</code> <code>//We didn't really want a BigInteger object for any other purpose, so we just created</code> <code>//and assigned an anonymous one at the point of need.</code>		

## Practice Your Java Level 1

```
}
```

And in even more compact form:

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"),
                                         new MathContext(4, RoundingMode.HALF_UP));
        System.out.println(bd01);
        // If we are not reusing the MathContext object, then we can simply create and assign
        // an anonymous one and assign it at the point of need.
    }
}
```

**Output:** 1.456E+6

28. Starting out with the string “1455734”, instantiate a `BigDecimal` object `bd01`, with a resultant value of 1.456, using a `BigInteger` and the same `MathContext` as in the preceding exercise. Print the value out.

*Hint: `BigDecimal(String value, scale, MathContext)`*

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(4, RoundingMode.HALF_UP);
        BigInteger bi01 = new BigInteger("1455734");
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(bi01, scale, mc01);
        System.out.println(bd01);
    }
}
```

**Output:** 1.456

29. Modify the code in exercise 27 to print out the value in non-exponential format.

*Hint: method `BigInteger.toPlainString`*

Modify the code line `System.out.println(bd01);`  
to the following:  
`System.out.println(bd01.toPlainString());`

30. Instantiate a `BigDecimal` object `bd01` with the value 1.455734, a precision of 15 and a rounding mode of `UNNECESSARY`. Print the number out.

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(15, RoundingMode.UNNECESSARY);
        BigInteger bi01 = new BigInteger("1455734");
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(bi01, scale, mc01);
        System.out.println(bd01);
    }
}
```

31. Repeat the preceding exercise, giving a precision of 4 instead of 15. State and explain the output of the program.

**Output:** And exception, `java.lang.ArithmaticException` is produced. The following statement is output:  
`Exception in thread "main" java.lang.ArithmaticException: Rounding necessary`

**Explanation:** As stated in the description of the rounding modes, if a rounding mode of `UNNECESSARY` is presented along with a precision, then `BigDecimal` demands that any number assigned to the `BigDecimal` instance in question conform to the specified precision; i.e. you the programmer are guaranteeing that you will ensure that the precision is correct.

32. Explain what a given precision of 0 in a `MathContext` means.

A precision of 0 in a `MathContext` implies that the precision of the value applied to the `BigDecimal` in question

	should be retained as entered.
33.	Repeat exercise 26, using a precision of 0 with the rounding mode <b>UNNECESSARY</b> . State and explain the output of the program.  Output: <b>1.455734</b> Explanation: The number entered is the number output, as we stated that it should retain its original precision.
34.	Repeat the preceding exercise, using the other rounding modes. State your findings.  Output: in all cases, <b>1.455734</b> Explanation: When a precision of 0 is given, it is in effect stating, leave the number alone with the given precision. This implies that rounding should not be attempted on the number in question.
35.	(Observing the rounding more closely) Write a program which will apply to the value 3445 a scale of 3, a precision of 2 and <b>RoundingMode = HALF_UP</b> . Before running the program, predict what you expect the resultant value to be. This number was chosen deliberately in order to facilitate careful observation of the mode(s) on it. The objective of this exercise is for you to observe and note where, irrespective of the length of the number, the rounding of it according to the specified mode starts. <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(2, RoundingMode.HALF_UP);         BigInteger bi01 = new BigInteger("3445");         int scale = 3;         BigDecimal bd01 = new BigDecimal(bi01, scale, mc01);         System.out.println(bd01.toString());     } }</pre>
<b>Rounding &amp; Scale</b>	
36.	I have the following <b>BigDecimal</b> object defined by the code snippet below:  <pre>MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP); int scale = 6; BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);</pre> The scaled value of the <b>BigDecimal</b> <b>bd01</b> = <b>1.455734</b> . Write a program which will create a new <b>BigDecimal</b> object <b>bd02</b> whose value = <b>1.455734000000</b> (now 12 digits behind the decimal point) using the value directly in the already existent <b>BigDecimal</b> <b>bd01</b> . <i>Hint: BigDecimal.setScale</i>  <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);         int scale = 6;         BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);         BigDecimal bd02 = bd01.setScale(12);         System.out.println(bd02);     } }</pre> <b>Output:</b> <b>1.455734000000</b>
37.	Given the value in <b>BigDecimal</b> <b>bd01</b> in the preceding exercise, return a new <b>BigDecimal</b> whose value = <b>bd01 × 10<sup>7</sup></b> . Implement this using three different methods. <i>Hint: BigDecimal.movePointRight, BigDecimal.multiply, BigDecimal.scaleByPowerOfTen</i>  <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);         int scale = 6;         BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);         BigDecimal bd02 = bd01.movePointRight(7);</pre>

## Practice Your Java Level 1

```
        System.out.println(bd02);
    }
}
```

**OR**

```
public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);
        BigDecimal bd02 = bd01.multiply(new BigDecimal(new BigInteger("10000000")));
        //here we multiply by 107
        System.out.println(bd02);
    }
}
```

**OR**

```
public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);
        BigDecimal bd02 = bd01.setScaleByPowerOfTen(7);
        //here we multiply by 107
        System.out.println(bd02);
    }
}
```

38. Using the value in `BigDecimal bd01` in the preceding exercise, return a new `BigDecimal` whose value =  $bd01 \times 10^{-8}$ .

Show this computation using three different methods.

*Hint: BigDecimal.movePointLeft, BigDecimal.divide, BigDecimal.setScaleByPowerOfTen*

```
import java.math.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);
        BigDecimal bd02 = bd01.movePointLeft(5);
        System.out.println(bd02);
    }
}
```

**OR**

```
public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);
        BigDecimal bd02 = bd01.divide(new BigDecimal(new BigInteger("100000")));
        //here we multiply by 107
        System.out.println(bd02);
    }
}
```

**OR**

```
public class PracticeYourJava {
    public static void main(String[] args) {
        MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);
        int scale = 6;
        BigDecimal bd01 = new BigDecimal(new BigInteger("1455734"), scale, mc01);
```

	<pre>         BigDecimal bd02 = bd01.setScaleByPowerOfTen(-5);         //here we multiply by 10<sup>7</sup>         System.out.println(bd02);     } } </pre>
39.	<p>Write a program which will assign each of the following floating point values <math>(40/3)</math>, <math>(4/3)</math> and <math>(1/3)</math> to objects of class <code>BigDecimal</code>, using a precision of 2 and a rounding mode of <code>HALF_UP</code>. Print out the resulting values.</p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(2, RoundingMode.HALF_UP);         BigDecimal bd01 = new BigDecimal((40f/3f), 2, mc01);         BigDecimal bd02 = new BigDecimal((4f/3f), 2, mc01);         BigDecimal bd03 = new BigDecimal((1f/3f), 2, mc01);         System.out.println(bd01); System.out.println(bd02); System.out.println(bd03);     } } </pre> <p><b>Output:</b> 13, 1.3 and 0.33 respectively.</p>
40.	<p>With the output from the preceding exercise in mind, what can we do to ensure that the output is consistently to a desired number of decimal places in such scenarios? I.e., how do we specify the precision of the decimal places in particular rather than the precision of the whole value?</p> <p>The way in which to specify the scale of the output (i.e. the decimal places), would be to apply the method <code>setScale</code>, specifying the desired scale.</p> <p>In order for this to work properly, the precision applied to the original number must be wide enough to support being rounded by the scale applied. For example if you want 2 digits of precision for the fractional part, then your initially stated precision for the number that is input has to be wide enough to cover that. In fact, depending on circumstances, you might be inclined to not specify a precision and rounding type for the input number, but rather just for the computed output.</p> <p>Applying this logic to the preceding exercise, the code can be expressed as follows:</p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(5, RoundingMode.HALF_UP);         //Note: We increased the precision; we know that 5 will cover the numbers in this case         BigDecimal bd01 = (new BigDecimal((40f/3f), mc01)).setScale(2, RoundingMode.HALF_UP);         BigDecimal bd02 = new BigDecimal((4f/3f), mc01).setScale(2, RoundingMode.HALF_UP);         BigDecimal bd03 = new BigDecimal((1f/3f), mc01).setScale(2, RoundingMode.HALF_UP);         System.out.println(bd01); System.out.println(bd02); System.out.println(bd03);     } } </pre> <p>In this modification to the code to the earlier exercise, we have actually rounded the number up twice. This is probably excessive (at least in this case) and need only be done when the method <code>setScale</code> is applied.</p> <p><b>Note:</b> Observe that the word “scale” here has a different meaning from when “scale” is used in the constructors.</p>
41.	<p>Rewrite the code to the preceding solution ensuring that rounding is only applied to the number once.</p> <pre> import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigDecimal bd01 = new BigDecimal(40f/3f).setScale(2, RoundingMode.HALF_UP);         BigDecimal bd02 = new BigDecimal(4f/3f).setScale(2, RoundingMode.HALF_UP);         BigDecimal bd03 = new BigDecimal(1f/3f).setScale(2, RoundingMode.HALF_UP);         System.out.println(bd01); System.out.println(bd02); System.out.println(bd03);     } } </pre> <p><i>Observe the following:</i> We don't have to worry that our value will not have the right precision, since we have</p>

## Practice Your Java Level 1

	<p>removed that constraint by removing the <code>MathContext</code> object. We also only round the number up once.</p>
42.	<p>Predict the result of running this program. Explain the output and propose a solution to the observation.</p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigDecimal bd01 = new BigDecimal(5f/2f).setScale(2);         System.out.println(bd01);         BigDecimal bd02 = new BigDecimal(40f/3f).setScale(2);         System.out.println(bd02);     } }</pre> <p><b>Results</b></p> <p>2.5  <b>Exception in thread "main" java.lang.ArithmeticException: Rounding necessary</b></p> <p>The first computation yielded a value of 2.5, which is correct.  The second computation actually caused an exception.</p> <p><b>Explanation</b></p> <p>The first computation yielded a rational result (i.e. non-recurring decimal). The result from such a computation can actually be written as 2.5000000000 if so desired; therefore due to its rationality, our setting of the scale to a value of 2 was feasible without impacting the actual value of the number. The second computation however yields a recurring decimal. If the JVM had decided to set a scale, it would have cut off what it considers to be a valid part of the number, which it prefers not to do; it prefers that we set the precision ourselves first, before we request that it be scaled.</p> <p><b>Conclusion/Solution:</b> Set a precision for non-rational computations to which we desire to add a scale.</p>

## BigDecimal: Issues in Initialization

43.	<p><b>Important:</b> Observe and explain the result of the computation on the left. Contrast it with the result of the computation on the right and state why the obtained values are different.</p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigInteger one = new BigInteger("1");         BigInteger three = new BigInteger("3");         BigDecimal bd01 = new BigDecimal(one);         BigDecimal bd02 = new BigDecimal(three);         BigDecimal bd03 = bd01.divide(bd02);         System.out.println(bd03);     } }</pre> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         BigDecimal bd03 = new BigDecimal(1f/3f);         System.out.println(bd03);     } }</pre> <p><b>Result:</b> For the computation on the <i>left</i>, an exception results, with the following statement being output:  <b>Exception in thread "main" java.lang.ArithmeticException: Non-terminating decimal expansion; no exact representable decimal result.</b></p> <p>For the computation on the <i>right</i>, the following is the result: 0.3333333432674407958984375</p> <p>The reason for the result on the left is because the computation 1/3 yields a non-terminating decimal. <code>BigDecimal</code> does not appreciate this and informs you via an exception.</p> <p>The reason for why the computation on the right did not fail, even though its result is wrong, is because the number input was first calculated as a floating point computation for type <code>float</code>, which yields approximations for non-terminating and irrational decimals, as can be seen from the result: beyond the 7<sup>th</sup> decimal place, the answer is wrong. If we used <code>double</code>, instead of <code>float</code>, the result would be 0.3333333333333314829616256247390992939472198486328125 which itself is wrong after the 16<sup>th</sup> decimal place.</p> <p><b>Conclusion:</b> Indeed the initialization mechanism the left is the correct one. However it is interminable and thus</p>
-----	--

	we must bound it by specifying a precision.
44.	<p><b>Important:</b> When inputting data into a <code>BigDecimal</code> for instantiation or any other purpose, why is it better to use <code>BigInteger</code> or <code>String</code> values rather than <code>float</code> or <code>double</code> values?</p> <p>The reason is that beyond a certain precision, many <code>float</code>/<code>double</code> numbers are not represented exactly in any computer. The computation in the preceding exercise is a clear example of this. However, integer numbers, being discrete, can be represented exactly. If you desire precise results, you need precise inputs: integer values which can be represented in objects of class <code>BigInteger</code>.</p> <p>While indeed in this chapter we have instantiated <code>BigDecimal</code> objects using floating point values, unless these yield rational results within the precision of the floating point type at hand, they are not right to use for initializing <code>BigDecimal</code> objects.</p>
45.	<p>Write a program which instantiates a <code>BigDecimal</code> with a <code>double</code> of value <code>0.1</code> and print the <code>BigDecimal</code> out. In the same program, initialize another <code>BigDecimal</code> with the <code>String</code> value “<code>0.1</code>” and print the <code>BigDecimal</code> out. Comment on your observations of the output.</p> <pre>import java.math.*; public class PracticeYourJava {     public static void main(String[] args) {         double d = 0.1d;         BigDecimal bd01 = new BigDecimal(d);         System.out.println(bd01);         BigDecimal bd02 = new BigDecimal("0.1");         System.out.println(bd02);     } }</pre> <p><b>Observations:</b></p> <p><code>bd01</code> has a value of <code>0.10000000000000055511151231257827021181583404541015625</code>.</p> <p><code>bd02</code> has a value of <code>0.1</code> precisely.</p> <p><b>Conclusion:</b> This exercise confirms the statements in the preceding exercise: initialize <code>BigDecimal</code> objects with integer values or string values.</p>
46.	<p>Put the following numbers <math>1/3</math> and <math>4/9</math> into <code>BigDecimal</code> objects, with a precision of 10 and with a scale of 10. Then perform the following operations on the numbers, printing the results out: Multiplication, Division (<math>a/b</math>), Addition, Subtraction (<math>a - b</math>).</p> <pre>import java.math.*; public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);         //no specific reason for choosing this rounding mode.         //Prepare the 1<sup>st</sup> number         BigDecimal bd01 = new BigDecimal(new BigInteger("1"));         BigDecimal bd02 = new BigDecimal(new BigInteger("3"));         BigDecimal a = bd01.divide(bd02, mc01).setScale(10);         //Prepare the 2<sup>nd</sup> number         BigDecimal bd03 = new BigDecimal(new BigInteger("4"));         BigDecimal bd04 = new BigDecimal(new BigInteger("9"));         BigDecimal b = bd03.divide(bd04, mc01).setScale(10);          //Now the computations...         BigDecimal result;         result = a.multiply(b,mc01).setScale(10); System.out.println(result);         result = a.divide(b,mc01).setScale(10); System.out.println(result);         result = a.add(b,mc01).setScale(10); System.out.println(result);         result = a.subtract(b,mc01).setScale(10); System.out.println(result);     } }</pre>
47.	<p>Given a <code>BigDecimal</code> of value <code>7.5</code>, print out the results of this number raised to the power of <code>20</code>.</p> <pre>import java.math.*;</pre>

## Practice Your Java Level 1

	<pre>public class PracticeYourJava {     public static void main(String[] args) {         //no specific reason for choosing this rounding mode.         //Prepare the 1<sup>st</sup> number         BigDecimal bd01 = new BigDecimal(new BigInteger("75"), 1); //scale 75 to 7.5         BigDecimal result = bd01.pow(20);         System.out.println(result);     } }</pre>
48.	<p>Why wasn't a precision required for the preceding computation?</p> <p>A precision wasn't required because the result is not an irrational number.</p>
49.	<p>I have a <code>BigDecimal</code> with value -7.5. Convert this to a <code>float</code>.</p> <p><i>Hint: <code>BigDecimal.floatValue</code></i></p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);         //no specific reason for choosing this rounding mode.         BigDecimal bd01 = new BigDecimal(new BigInteger("-75"), 1); //scale -75 to -7.5         float f01 = bd01.floatValue();         System.out.println(f01);     } }</pre>
50.	<p>I have a <code>BigDecimal</code> with value -7.5. Convert this to a <code>BigInteger</code>. Print out the resultant value.</p> <p><i>Hint: <code>BigDecimal.toBigInteger</code></i></p> <pre>import java.math.*;  public class PracticeYourJava {     public static void main(String[] args) {         MathContext mc01 = new MathContext(10, RoundingMode.HALF_UP);         //no specific reason for choosing this rounding mode.         BigDecimal bd01 = new BigDecimal(new BigInteger("-75"), 1); //scale -75 to -7.5         BigInteger bi01 = bd01.toBigInteger();         System.out.println(bi01);     } }</pre> <p><b>Observations:</b> The fractional part of the number is truncated.</p>
51.	<p>What is the result if instead of the method <code>toBigInteger</code> used in the preceding exercise we used the method <code>toBigIntegerExact</code>?</p> <p>An exception, <code>java.lang.ArithmaticException</code> will result with the following output string:  <b>Exception in thread "main" java.lang.ArithmaticException: Rounding necessary</b></p> <p>This is stating that it will function without generating an exception only if integer values are input.</p>

# Chapter 13. Number Handling IV – `strictfp` & `StrictMath`

In this chapter, we present exercises on strict handling of numerical computations in Java using the keyword `strictfp` and the class `StrictMath`.

The topic is not difficult, it however has been put into its own separate chapter for emphasis.

This chapter is the fourth of five chapters in this book on number handling in Java.

- What is the concept behind the existence of the keyword `strictfp`?

There are a number of factors that comprise the result of a numerical computation: the design of the CPU on which the computation is performed and the algorithm which is used to perform the computation.

**Hardware:** When performing floating point computations, some hardware platforms actually have wider intermediate computation registers than what a `float` or a `double` can support; the width of the registers aids in handling intermediate computations which exceed the width of the type in question. We explain by example:

First, let's say we have a processor that supports decimal numbers with up to 4-significant digits. In this example, we have the CPU Model-1 which also only uses 4 significant digits all throughout internally. If I have the computation:  $1000 \times 10 - 5000$ , with only 4 significant digits everywhere, the result will be as follows:

```
1000
x 10
-----
0000 ← Overflow error in CPU! We lost the "1" because our register is only 4 digits wide
-5000
-----
-5000
-----
```

Now, if another manufacturer decided to create a decimal CPU Model-2 which supports up to only 4 significant digits, however internally it uses an expanded width of 5 significant digits for intermediate computations, we would have the following:

```
1000
x 10
-----
10000 ← We still have the "1" because the intermediate internal register is 5 digits wide
-5000
-----
5000 ← This result fits within 4 digits.
-----
```

Clearly if my program is run on a computer with CPU Model-1 my answer will be different from the answer on one run on a computer with CPU Model-2. The `strictfp` keyword prevents this variance in intermediate computation by forcing intermediate computations to conform to the IEEE-754 standard that defines the handling of floating point numbers; this specification demands that `double` values occupy a maximum of 64 bits at all times and a `single/float` occupies a maximum of 32 at all times. This is called the strict-FP requirement. With the preceding example in mind, the output of CPU-1 (which has/uses no extra places internally) is what would result under the strict-FP requirement (Yes, that output in some situations might not be correct due to under/overflow, however it is consistent across platforms. It is left to the programmer to handle the under/overflow properly).

When the keyword `strictfp` is applied to a method, all floating point computations within that method are made to conform to the strict-FP requirement. When the keyword is applied to a class, all floating point

## Practice Your Java Level 1

	<p>computations within that class are made to conform to the strict-FP requirement.</p> <p>The keyword <code>strictfp</code> only applies to the floating point types <code>double</code> and <code>float</code>.</p>
2.	<p>What is the concept behind the existence of the class <code>StrictMath</code>?</p> <p>The class <code>StrictMath</code> exists in order to ensure strict reproducibility, to the bit level, of the output of mathematical operations in Java irrespective of the platform on which the mathematical operations are run.</p> <p>Certain applications require bit-level reproducibility across the different platforms on which the program may be run. The implementation of the class <code>Math</code> only requires accuracy within a certain “high tolerance”; this implies that there may very well be slight variations in the output due to the different hardware on which the program is being run or differences due to the choice of algorithm for implementing the floating point method in question. The strict reproducibility of results is achieved by <u>specifying the specific algorithm</u> that each method in <code>StrictMath</code> utilizes to compute its results. These algorithms are generally software (not hardware) algorithms. You can consider that any computation that uses <code>StrictMath</code> is effectively <code>strictfp</code> compliant.</p>
3.	<p>What is the difference in performance between <code>Math</code> methods and their <code>StrictMath</code> counterparts?</p> <p>The <code>StrictMath</code> methods can generally be expected to be slower as they are more often than not software algorithms.</p> <p>In some Java implementations, the <code>Math</code> methods are programmed to call their <code>StrictMath</code> equivalents. However even in such cases the virtual machine is free to use platform specific functionality for such <code>Math</code> methods, in short, it is free to ignore the directive to use <code>StrictMath</code> in such cases.</p>
4.	<p>Compute the cosine, tangent and sine of 75 degrees using the appropriate methods from class <code>StrictMath</code>. Print the results out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double degrees = 75;         double radians = StrictMath.toRadians(degrees);         double cosine = StrictMath.cos(radians);         double sine = StrictMath.sin(radians);         double tangent = StrictMath.tan(radians);          System.out.printf("The cosine of %f degrees = %f\n", degrees, cosine);         System.out.printf("The sine of %f degrees = %f\n", degrees, sine);         System.out.printf("The tangent of %f degrees = %f\n", degrees, tangent);     } }</pre>
5.	<p>True or False: Java makes every floating point constant (defined by the keyword <code>final</code>) that is defined at compile time strict-FP.</p> <p>True</p>
6.	<p>Explain the difference in functionality between the programs below:</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double degrees = 75;         double radians =             StrictMath.toRadians(degrees);         double cosine = StrictMath.cos(radians);         double sine = StrictMath.sin(radians);         double tangent = StrictMath.tan(radians);     } }</pre> <pre>public class PracticeYourJava {     public static strictfp void main(String[] args){         double degrees = 75;         double radians =             Math.toRadians(degrees);         double cosine = Math.cos(radians);         double sine = Math.sin(radians);         double tangent = Math.tan(radians);     } }</pre> <p>The difference is that the program on the left uses <code>StrictMath</code> which demands specific algorithms and thus the results on every JVM will be exactly the same, bit by bit. The program on the right demands standard intermediate result handling, however the algorithm used is not enforced across JVMs.</p>
7.	<p>Why is there no <code>StrictMath</code> equivalent of the class <code>BigDecimal</code>?</p> <p>There is none because <code>BigDecimal</code> does not actually use the IEEE floating point number format to represent numbers.</p>

# Chapter 14. Number Handling V – Unsigned Integers

In this chapter, we look at the topic of handing unsigned integers in Java. This feature, existent in many other languages, has now had support for it added to Java as of version 8.

This chapter is the fifth of five chapters in this book on number handling in Java.

1.	Explain what an <u>unsigned</u> number is.
	An unsigned number is one that is always non-negative, i.e. 0 or higher. The word “unsigned” means that the number does not need to have a sign written in front of it, i.e. the non-negative numbers.
2.	Describe briefly the difference in internal storage between a signed number and an unsigned number.  With a signed number, the most significant bit in the set of bits that represent the number is used to indicate the sign of the number. This implies that there is one less bit for representing the actual numerical value in such a case. With unsigned numbers, since unsigned numbers are known to always be positive, there is no need to use any bits to represent the sign of the number and thus the full set of bits allocated can be used to represent the numerical value at hand.  For example, the <code>int</code> type which represents signed integers is represented using 32 bits. However, the highest bit represents the sign bit, leaving us with 31 bits for representing positive and negative <code>int</code> values. For an unsigned <code>int</code> however, the full 32 bits are used to represent the numerical value.
3.	List the integer types in Java that have support for unsigned integers.  The types in Java that have support for unsigned types are the types <code>int</code> and <code>long</code> . Note that the support is largely implemented through specific methods in the <code>Integer</code> and <code>Long</code> classes; no new primitives or classes have been created to support these types.
4.	What is an “unsigned int” in Java?  An unsigned <code>int</code> in Java is a number which occupies the same space as the integer type <code>int</code> which is a signed type.  Being that the type <code>int</code> occupies 32 bits, of which 31 are used for the value and the highest bit for the sign of the number, an unsigned <code>int</code> , not needing a bit for its sign is free to use the full 32 bits to represent a number.
5.	What is an unsigned long in Java?  An unsigned long in Java is a number which occupies the same space as the <code>long</code> type which is a signed type, however as with an unsigned <code>int</code> , the unsigned long does not use the uppermost bit as a sign bit and therefore it can use the full space allocated to a <code>long</code> to represent a number.
6.	How does Java store unsigned int values?  Interestingly, Java uses the <code>int</code> primitive to store unsigned <code>int</code> values, since the <code>int</code> primitive does indeed have 32 bits which is the space that an unsigned <code>int</code> requires. However, Java provides specific methods to use for storing unsigned numbers into <code>int</code> primitive variables.
7.	What is the numerical range supported by an unsigned int in Java?  $0 \text{ to } 2^{32} - 1 = 0 \text{ to } 4,294,967,295$
8.	What is the numerical range supported by an unsigned long in Java?  $0 \text{ to } 2^{64} - 1 = 0 \text{ to } 1,8446,744,073,709,551,615$
9.	Explain the result of attempting to compile the following code:  <code>public class PracticeYourJava {     public static void main(String[] args) {         int int01 = 4294967290;     } }</code>  The code will <i>not</i> compile, because we attempted to assign a value greater than <code>Integer.INT_MAX</code> to an <code>int</code> .

## Practice Your Java Level 1

10.	<p>I want to represent the number 50 as an unsigned integer. Explain why is it invalid to write the following code:</p> <pre>int uint01 = 50;</pre>
	<p>The reason why this is invalid for unsigned integers is because in Java, when a number is assigned directly to an <code>int</code> type using the assignment operator (<code>=</code>), Java sees such a number as a signed integer and represents it as such internally. For this reason, we need another operator or method which understands that we are trying to use that space for an unsigned integer to set the internal representation of the number appropriately.</p> <p>Note that many other programming languages have an explicit unsigned integer type (named <code>uint</code> in some cases).</p>
11.	<p>Explain the result of compiling the following code. Explain what the code means.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("3147483647");     } }</pre>
	<p>The code compiles and runs successfully.</p> <p>This code uses the <code>Integer.parseUnsignedInt</code> method to have Java store an unsigned <code>int</code> in the same space as an <code>int</code>.</p>
12.	<p>Enhance the code in the preceding exercise to print out the unsigned integer that is in the <code>int</code> variable <code>uint01</code>.</p>
	<p><i>Hint: Integer.parseUnsignedInt, Integer.toUnsignedString</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("3147483647");         System.out.println(Integer.toUnsignedString(uint01));     } }</pre>
13.	<p>Add the following two unsigned integers; 3294967294 and 45. Print the result out.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("3294967294");         int uint02 = Integer.parseUnsignedInt("45");         int uint03 = uint01 + uint02;         System.out.println(Integer.toUnsignedString(uint03));     } }</pre>
14.	<p>Subtract the second of the following unsigned integers from the first; 4,294,967,294 and 3,333,967,295. Print the result out.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("4294967294");         int uint02 = Integer.parseUnsignedInt("3333967295");         int uint03 = uint01 - uint02;         System.out.println(Integer.toUnsignedString(uint03));     } }</pre>
15.	<p>Multiply the following two unsigned integers: 2,094,967,294 and 2.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("2094967294");         int uint02 = Integer.parseUnsignedInt("2");         int uint03 = uint01 * uint02;         System.out.println(Integer.toUnsignedString(uint03));     } }</pre>
16.	<p>Given the following two unsigned integers, 4,294,967,295 and 2,147,483,648. Divide the first unsigned integer by the second. Print out the quotient of the result, ignoring the remainder.</p>
	<p><i>Hint: Integer.divideUnsigned</i></p> <pre>public class PracticeYourJava {</pre>

	<pre> public static void main(String[] args) {     int uint01 = Integer.parseUnsignedInt("4294967295");     int uint02 = Integer.parseUnsignedInt("3147483647");     int uint03 = Integer.divideUnsigned(uint01, uint02);     System.out.println(Integer.toUnsignedString(uint03)); } } </pre>
17.	<p>Divide the first unsigned integer by the second. Print out both the quotient and the remainder of the result.  <i>Hint: Integer.divideUnsigned, Integer.remainderUnsigned</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("4294967295");         int uint02 = Integer.parseUnsignedInt("3147483647");         int quotient = Integer.divideUnsigned(uint01, uint02);         int remainder = Integer.remainderUnsigned(uint01, uint02);         System.out.println("The quotient = " + Integer.toUnsignedString(quotient));         System.out.println("The remainder = " + Integer.toUnsignedString(remainder));     } } </pre>
18.	<p>Using unsigned integer arithmetic, multiply the following unsigned integer values 2,394,967,294 and 5,000 and print out the result.  <i>Hint: Long.parseLongLong, Long.toUnsignedString</i></p> <p>The multiplication of these numbers will overflow an unsigned integer, so we use unsigned long variables instead.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         long ulong01 = Long.parseLongLong("2394967294");         long ulong02 = Long.parseLongLong("5000");         long ulong03 = ulong01 * ulong02;         System.out.println(Long.toUnsignedString(ulong03));     } } </pre>
19.	<p>Write code to compare two unsigned integers, printing out which is larger. Test your code with the values 3,394,967,294 and 3,400,001,003.  <i>Hint: Integer.compareUnsigned</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("3394967294");         int uint02 = Integer.parseUnsignedInt("3400001003");          int result = Integer.compareUnsigned(uint01, uint02);         if(result &lt; 0)             System.out.println(Integer.toUnsignedString(uint01) + " is larger");         else if(result &gt; 0)             System.out.println(Integer.toUnsignedString(uint02) + " is larger");         else             System.out.println("The numbers are equal!");     } } </pre>
20.	<p>Left-shift the unsigned integer value 2,094,634,885 by 1 and print out the results.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("2094634885");         int uint02 = uint01 &lt;&lt; 1;         System.out.println("The result = " + Integer.toUnsignedString(uint02));     } } </pre>
21.	<p>Right-shift the unsigned integer value 4,194,634,885 by 3 and print out the results.</p> <pre> public class PracticeYourJava { </pre>

## Practice Your Java Level 1

	<pre> public static void main(String[] args) {     int uint01 = Integer.parseUnsignedInt("4194634885");     int uint02 = uint01 &gt;&gt;&gt; 3; // NOTE THE new &gt;&gt;&gt; operator for right-shifting                                 // unsigned integer values!     System.out.println("The result = " + Integer.toUnsignedString(uint02)); } } </pre>
22.	<p>Print the unsigned integer value 4,194,634,885 as a binary number.  <i>Hint: Integer.toUnsignedString(number, base)</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          int uint01 = Integer.parseUnsignedInt("4194634885");         String str01 = Integer.toUnsignedString(uint01, 2);         System.out.println(str01);     } } </pre>
23.	<p>Reprint the number in the preceding exercise in the following number bases: 16, 8 and 7. Ensure that the alphabetic values in the hexadecimal number are in uppercase.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          int uint01 = Integer.parseUnsignedInt("4194634885");         String str01 = Integer.toUnsignedString(uint01, 16);         String str02 = Integer.toUnsignedString(uint01, 8);         String str03 = Integer.toUnsignedString(uint01, 7);         System.out.println("The result(Base 16) = " + str01.toUpperCase());         System.out.println("The result(Base 8) = " + str02);         System.out.println("The result(Base 7) = " + str03);     } } </pre>
24.	<p>I have the value 50 in a <code>short</code> variable <code>int01</code>. Add the value in <code>int01</code> to the unsigned integer value 4,194,634,885 and print out the results.  <i>Hint: Short.toUnsignedInt</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          short int01 = 50;         int uint02 = Integer.parseUnsignedInt("4194634885");         int uint03 = Short.toUnsignedInt(int01); //Method to convert short to unsigned int         int sum = uint02 + uint03;         String str01 = Integer.toUnsignedString(sum);         System.out.println("result(Base 8) = " + str01);     } } </pre>
25.	<p>I have the unsigned integer value 4,194,634,885 represented in a variable <code>uint01</code>. Add the value in <code>uint01</code> to the unsigned long 400,194,634,234 and print out the results.  <i>Hint: Long.toUnsignedInt</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          int uint01 = Integer.parseUnsignedInt("4194634885");         long long01 = Long.parseLong("400194634234");          //Now convert that unsigned integer representation in uint01 to an unsigned long         //We do this by converting it to a String and then converting that String to a long         String str01 = Integer.toUnsignedString(uint01);         long long02 = Long.parseLong(str01); //Now we've put it in a long         long result = long01 + long02;         str01 = Long.toUnsignedString(result);         System.out.println("result = " + str01);     } } </pre>

26.	<p><i>Overflow/Underflow handling in unsigned math</i></p> <p>Add the following two unsigned int values: 4,294,967,295 and 15. Print out the result and comment on it.</p>
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         int uint01 = Integer.parseUnsignedInt("4294967295");         int uint02 = Integer.parseUnsignedInt("15");         int result = uint01 + uint02;         String str01 = Integer.toUnsignedString(result);         System.out.println("result = " + str01);     } }</pre> <p><b>Result:</b> 14</p> <p><b>Comments:</b> The value is clearly wrong. What has occurred here is that the computation has overflowed.</p>
27.	<p>How do we detect overflow of unsigned integer computations?</p> <p>At present, there is no built-in method at present in Java to do so (i.e. nothing like the <code>Integer.addExact</code> method for the <code>int</code> type). Therefore, you will have to implement your own overflow/underflow detection code for Java unsigned number math.</p>
E1	<p>In addition to the support Java provides for unsigned int values, it also provides support for unsigned long values. The unsigned arithmetic methods in the class <code>Integer</code> are replicated in function in the class <code>Long</code>. This being the case, solve the following questions; all the numbers in these questions are to be represented as unsigned long integers.</p> <ul style="list-style-type: none"> <li>(a) <math>50 * 1000</math></li> <li>(b) <math>10,446,744,073,709,551,615 + 8,046,744,073,709,551,181</math></li> <li>(c) <math>9000 + 10000</math></li> <li>(d) Write a program to determine which of two unsigned long variables is greater; test the program using the values 760,194,634,234 and 760,193,637,777.</li> <li>(e) Left-shift the unsigned long value 4,046,744,073,709,551,181 by two spaces.</li> <li>(f) Right-shift the unsigned long value 18,446,744,073,709,551,615 (remember the <code>&gt;&gt;&gt;</code> operator).</li> </ul>



# Chapter 15. Date/Time Handling – Foundation

An important part of any computer language is the ability to determine and measure time, including the determination of dates and times (and sub-elements thereof such as the year, month, day, hour, minute, second), past, present or future, time zones, measurement of date/time differences as well as the important topic of formatting date/time output. This chapter presents exercises on each of these topics. Also, introductory exercises on the measurement of the running time of code segments are presented.

Java introduced a new date/time handling API in Java 8; the exercises in this chapter are based on this new API. This API is very extensive.

The key classes and enumerations utilized in this chapter include the classes `LocalDateTime`, `LocalDate`, `LocalTime`, `Year`, `YearMonth`, `Month`, `ZonedDateTime`, `ZoneOffset`, `ZoneId`, `Instant`, `Duration`, `DateTimeFormatter`, `DayOfWeek` and `ChronoUnit`.

Class <code>LocalDateTime</code>	
1.	<p>Write a program which prints out the current date &amp; time in the default format. <i>Hint: </i><code>LocalDateTime.now</code></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentDateTime = LocalDateTime.now();         System.out.println("The current date &amp; time is " + currentDateTime);     } }</pre>
2.	<p>Print each of the following sub-elements of the <code>LocalDateTime</code> of the preceding exercise out separately: Year, Month (both in word form and in numerical form), Day of Month, Day of Week, Hour, Minute, Seconds and Nanoseconds.</p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentDateTime = LocalDateTime.now();         System.out.println("Year      = " + currentDateTime.getYear());         System.out.println("Month     = " + currentDateTime.getMonth());         System.out.println("MonthValue = " + currentDateTime.getMonthValue());         System.out.println("DayOfMonth = " + currentDateTime.getDayOfMonth());         System.out.println("DayOfWeek  = " + currentDateTime.getDayOfWeek());         System.out.println("Hour       = " + currentDateTime.getHour());         System.out.println("Minute     = " + currentDateTime.getMinute());         System.out.println("Seconds    = " + currentDateTime.getSecond());         System.out.println("Nano-seconds = " + currentDateTime.getNano());     } }</pre>
3.	<p>What day of the year is today? <i>Hint: </i><code>LocalDateTime.getDayOfYear</code></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentDateTime = LocalDateTime.now();         System.out.printf("Today is day %d of the year\n", currentDateTime.getDayOfYear());     } }</pre>

## Practice Your Java Level 1

4.	Set a <code>LocalDateTime</code> object to the 1 <sup>st</sup> day of the current year. <i>Hint: LocalDateTime.of</i> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentTime = LocalDateTime.of(LocalDateTime.now().getYear(), 1, 1, 0, 0);     } }</pre>
5.	Print out the date and time it was 2 years ago from now. <i>Hint: LocalDateTime.minusYears</i> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentTime = LocalDateTime.now();         LocalDateTime twoYearAgo = currentTime.minusYears(2);         System.out.println("The date and time 2 years ago was: " + twoYearAgo);     } }</pre>
6.	Print out the date and time it was seven months ago. <i>Hint: LocalDateTime.minusMonths</i> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentTime = LocalDateTime.now();         LocalDateTime monthsAgo = currentTime.minusMonths(7);         System.out.println("The date and time 7 months ago was: " + monthsAgo);     } }</pre>
7.	Print out the date and time it was six weeks ago. <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentTime = LocalDateTime.now();         LocalDateTime weeksAgo = currentTime.minusWeeks(6);         System.out.println("The date and time 6 months ago was: " + weeksAgo);     } }</pre>
8.	Print out the date and time it was 15 days ago. <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentTime = LocalDateTime.now();         LocalDateTime daysAgo = currentTime.minusDays(15);         System.out.println("The date and time 15 days ago was: " + daysAgo);     } }</pre>
9.	Print out the date and time it was 9 hours ago. <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentTime = LocalDateTime.now();         LocalDateTime hoursAgo = currentTime.minusHours(9);         System.out.println("The date and time 9 hours ago was: " + hoursAgo);     } }</pre>

10.	Print out the date and time it was 54 minutes ago.
	<pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentDateTime = LocalDateTime.now();         LocalDateTime minsAgo = currentDateTime.minusMinutes(54);         System.out.println("The date and time 54 minutes ago was: " + minsAgo);     } }</pre>
11.	Print out the date and time it was 1000 seconds ago.
	<pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentDateTime = LocalDateTime.now();         LocalDateTime secondsAgo = currentDateTime.minusSeconds(1000);         System.out.println("The date and time 1000 seconds ago was: " + secondsAgo);     } }</pre>
12.	Print out the following dates & times:
	<ul style="list-style-type: none"> <li>• 5 years in the future</li> <li>• 9 months in the future</li> <li>• 3 weeks in the future</li> <li>• 14 days in the future</li> <li>• 2 hours in the future</li> <li>• 15 minutes in the future</li> <li>• 1000 seconds in the future</li> </ul>
	<p><i>Hint: LocalDateTime.plusYear, plusMonths, etc</i></p>
	<pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         System.out.printf("date/time in 5 years will be %s\n", LocalDateTime.now().plusYears(5));         System.out.printf("date/time in 9 months will be %s\n", LocalDateTime.now().plusMonths(9));         System.out.printf("date/time in 3 weeks will be %s\n", LocalDateTime.now().plusWeeks(3));         System.out.printf("date/time in 14 days will be %s\n", LocalDateTime.now().plusDays(14));         System.out.printf("date/time in 2 hours will be %s\n", LocalDateTime.now().plusHours(2));         System.out.printf("date/time in 15 minutes will be %s\n",                             LocalDateTime.now().plusMinutes(15));         System.out.printf("date/time in 1000 seconds will be %s\n",                             LocalDateTime.now().plusSeconds(5));     } }</pre>
13.	Determine which day is the 200 <sup>th</sup> day of the current year and print out the month and day thereof.
	<p><i>Hint: Determine the 1<sup>st</sup> day and add 199. Also see LocalDateTime.plusDays</i></p>
	<pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime currentDateTime = LocalDateTime.of(LocalDateTime.now().getYear(), 1, 1, 0, 0);         LocalDateTime day200 = currentDateTime.plusDays(199);         System.out.printf("The 200th day of the year is %s %s\n", day200.getMonth(),                             day200.getDayOfMonth());     } }</pre>

## Practice Your Java Level 1

Class <code>LocalDate</code>	
14.	Determine the local date (not the time) and print it out in the default format. <i>Hint: <code>LocalDate.now</code></i> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDate currentDate = LocalDate.now();         System.out.println("The date is " + currentDate);     } }</pre>
15.	Print the following sub-elements of the <code>LocalDate</code> of the preceding exercise out separately: Year, Month (both in word form and in numerical form), Day of Month and Day of Week. <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDate currentDate = LocalDate.now();         System.out.println("Year      = " + currentDate.getYear());         System.out.println("Month     = " + currentDate.getMonth());         System.out.println("MonthValue = " + currentDate.getMonthValue());         System.out.println("DayOfMonth = " + currentDate.getDayOfMonth());         System.out.println("DayOfWeek   = " + currentDate.getDayOfWeek());     } }</pre>
16.	Print out the latest and earliest possible dates that can be represented in Java. <i>Hint: the constants in the class <code>LocalDate</code></i> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDate maxDate = LocalDate.MAX;         System.out.println("The maximum date supported is: " + maxDate);         LocalDate minDate = LocalDate.MIN;         System.out.println("The minimum date supported is: " + minDate);     } }</pre>
17.	Print out how many days there are in the current month. <i>Hint: <code>LocalDate.lengthOfMonth</code></i> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDate currentDate = LocalDate.now();         int daysInMonth = currentDate.lengthOfMonth();         System.out.println("Number of days in this month = " + daysInMonth);     } }</pre>
18.	Print out the date for the following dates relative to the current date: <ul style="list-style-type: none"><li>• 14 years ago</li><li>• 9 months ago</li><li>• 4 weeks ago</li><li>• 7 days ago</li></ul> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {</pre>

	<pre> LocalDate currentDate = LocalDate.now(); System.out.printf("date/time 14 years ago = %s\n", LocalDateTime.now().minusYears(14)); System.out.printf("date/time 9 months ago = %s\n", LocalDateTime.now().minusMonths(9)); System.out.printf("date/time 4 weeks ago = %s\n", LocalDateTime.now().minusWeeks(4)); System.out.printf("date/time in 7 days ago = %s\n", LocalDateTime.now().minusDays(7)); } } </pre>
19.	<p>Print out the date for the following dates relative to the current date:</p> <ul style="list-style-type: none"> <li>• 5 years in the future</li> <li>• 9 months in the future</li> <li>• 3 weeks in the future</li> <li>• 14 days in the future</li> </ul> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDate currentDate = LocalDate.now();         System.out.printf("date/time in 5 years will be %s\n", LocalDateTime.now().plusYears(5));         System.out.printf("date/time in 9 months will be %s\n", LocalDateTime.now().plusMonths(9));         System.out.printf("date/time in 3 weeks will be %s\n", LocalDateTime.now().plusWeeks(3));         System.out.printf("date/time in 14 days will be %s\n", LocalDateTime.now().plusDays(14));     } } </pre>
20.	<p>Print out how many days that there are in each of the years from the year 2010 to the year 2030. At the same time, print out whether each of the years is a leap year or not.</p> <p><i>Hint: LocalDate.of, LocalDate.isLeapYear</i></p> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         for(int i=2010; i &lt;= 2030; i++)         {             LocalDate dt01 = LocalDate.of(i,1,1); //go to the 1st of each year             int numDays = dt01.lengthOfYear();             System.out.printf("There are %d days in year %d\n", numDays, i);              if(dt01.isLeapYear())                 System.out.printf("Year %d is a leap year\n", i);             else                 System.out.printf("Year %d is NOT a leap year\n", i);         }     } } </pre>
21.	<p>What date is the 300<sup>th</sup> day of this year?</p> <p><i>Hint: get the current year from the current LocalDate (now) and then use LocalDate.ofYearDay</i></p> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         int dayInQuestion = 300;         int iYear = LocalDate.now().getYear();         LocalDate dateInQuestion = LocalDate.ofYearDay(iYear, dayInQuestion);         System.out.printf("The 300th day of the year is %s\n", dateInQuestion);     } } </pre>
22.	<p>In computer terms, what is “epoch time”?</p> <p>Epoch time is a system for describing instants in time, in seconds, that have elapsed since the time point called the “epoch”, that time point being Thursday, 1<sup>st</sup> January, 1970 UTC at 00:00:00 hours.</p>

## Practice Your Java Level 1

23.	<p>Today is how many days since the epoch?</p> <p><i>Hint: LocalDate.toEpochDay</i></p> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDate currentDate = LocalDate.now();         long daysSinceEpoch = currentDate.toEpochDay();         System.out.printf("Today is %d days since the epoch.\n", daysSinceEpoch);     } }</pre>
24.	<p>What was the 5000<sup>th</sup> day after the epoch?</p> <p><i>Hint: LocalDate.ofEpochDay</i></p> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         int daysInQuestion = 5000;         LocalDate dateInQuestion = LocalDate.ofEpochDay(daysInQuestion);         System.out.printf("The date %d days from the epoch = %s\n", daysInQuestion, dateInQuestion);     } }</pre>
25.	<p>I have the following <code>LocalDate</code> object: <code>LocalDate date01 = LocalDate.now();</code> Modify the year, month and day field of this object to 2020, 10 and 15 respectively. Print the resulting <code>LocalDate</code> object out.</p> <p><i>Hint: LocalDate.withYear, withMonth, withDayOfMonth</i></p> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDate date01 = LocalDate.now();         date01 = date01.withYear(2020);         date01 = date01.withMonth(10);         date01 = date01.withDayOfMonth(15);          System.out.printf("The new DateTime = %s\n", date01);     } }</pre>
26.	<p>I have the following <code>LocalDate</code> object: <code>LocalDate date01 = LocalDate.now();</code> Modify this object to the 121<sup>st</sup> date of the year. Print the resulting <code>LocalDate</code> out.</p> <p><i>Hint: LocalDate.withDayOfYear</i></p> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDate date01 = LocalDate.now();         date01 = date01.withDayOfYear(121);          System.out.printf("The new DateTime = %s\n", date01);     } }</pre>
27.	<p>I have the following <code>LocalDate</code> object: <code>LocalDate date01 = LocalDate.now();</code> Copy it to a <code>String</code> variable and print the <code>String</code> variable out.</p> <pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDate date01 = LocalDate.now();         String s01 = date01.toString();          System.out.printf("The DateTime = %s\n", s01);     } }</pre>

	}
28.	Initialize a <code>LocalDate</code> object to your birthday. Print the resulting object out. <i>Hint: LocalDate.of</i>
	<pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalDate birthDate = LocalDate.of(1970, 03, 01); //for example 1st March, 1970         System.out.println("My birthdate is " + birthDate);     } }</pre>
<b>Class <code>LocalTime</code></b>	
30.	Determine the current local date (not the time) and print it out in the default format. <i>Hint: LocalTime.now</i>
	<pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalTime currentTime = LocalTime.now();         System.out.println("The time is " + currentTime);     } }</pre>
31.	Print the following sub-elements of the <code>LocalTime</code> of the preceding exercise out separately: Hour, Minute, Seconds and Nanoseconds.
	<pre>import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {         LocalTime currentTime = LocalTime.now();         System.out.println("Hour      = " + currentTime.getHour());         System.out.println("Minute    = " + currentTime.getMinute());         System.out.println("Seconds   = " + currentTime.getSecond());         System.out.println("Nano-seconds = " + currentTime.getNano());     } }</pre>
32.	Compute and print out the following times: <ul style="list-style-type: none"> <li>• 27 hours ago</li> <li>• 24 minutes ago</li> <li>• 51 seconds ago</li> </ul>

## Practice Your Java Level 1

	<pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime currentTime = LocalTime.now();         System.out.printf("time 27 hours ago was %s\n", currentTime.minusHours(27));         System.out.printf("time 24 minutes ago was %s\n", currentTime.minusMinutes(24));         System.out.printf("time 51 seconds ago was %s\n", currentTime.minusSeconds(51));     } }</pre>
33.	<p>Compute and print out what the time will be in:</p> <ul style="list-style-type: none"> <li>• 3 hours</li> <li>• 45 minutes</li> <li>• 100 seconds</li> </ul> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime currentTime = LocalTime.now();         System.out.printf("time in 3 hours      = %s\n", currentTime.plusHours(3));         System.out.printf("time in 45 minutes   = %s\n", currentTime.plusMinutes(45));         System.out.printf("time in 100 seconds = %s\n", currentTime.plusSeconds(100));     } }</pre>
34.	<p>What was the time 2.5 hours ago?</p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime time = LocalTime.now();         time = time.minusHours(2);         time = time.minusMinutes(30);         System.out.printf("The time 2.5 hours ago was %s\n", time);     } }</pre>
35.	<p>What time of the day is the 49000<sup>th</sup> second of the day?  <i>Hint: LocalTime.ofSecondOfDay</i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         long secondsToConvert= 49000L;         LocalTime currentTime = LocalTime.ofSecondOfDay(secondsToConvert);         System.out.printf("The %d second is %s\n", secondsToConvert, currentTime);     } }</pre>
36.	<p>What time of the day is the 55897558989000<sup>th</sup> nanosecond of the day?</p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         long nsecondsToConvert= 55897558989000L;         LocalTime currentTime = LocalTime.ofNanoOfDay(nsecondsToConvert);         System.out.printf("The %d second is %s\n", nsecondsToConvert, currentTime);     } }</pre>
37.	<p>Initialize a LocalTime object to the time 3:30:35 pm.</p> <pre>import java.time.*; public class PracticeYourJava {</pre>

	<pre>public static void main(String[] args) {     LocalTime time = LocalTime.of(15, 30, 35);     System.out.printf("The time is %s", time); }</pre>
38.	Initialize a <code>LocalTime</code> object to the time 16:30. <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime time = LocalTime.of(16, 30);         System.out.printf("The time is %s", time);     } }</pre>
39.	Modify the hour field of the resultant <code>LocalTime</code> object of the solution to the preceding exercise to 17. <i>Hint: LocalTime.of, LocalTime.withHour</i> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime time = LocalTime.of(16, 30);         time = time.withHour(17);         System.out.printf("The time is %s", time);     } }</pre>
40.	Modify the minute and seconds fields in the <code>LocalTime</code> object in the preceding exercise to 37 and 24 respectively. Also set the nanoseconds field to a value of 0. <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime time = LocalTime.of(16, 30);         time = time.withMinute(37);         time = time.withSecond(24);         time = time.withNano(0);         System.out.printf("The time is %s\n", time);     } }</pre>
41.	Convert the current time to seconds and nanoseconds. Print the values out. <i>Hint: LocalDate.toSecondOfDay, toNanoOfDay</i> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime currentTime = LocalTime.now();         long seconds = currentTime.toSecondOfDay();         long nseconds = currentTime.toNanoOfDay();          System.out.printf("Current time in seconds = %d\n", seconds);         System.out.printf("Current time in nano-seconds = %d\n", nseconds);     } }</pre>
42.	Store the current time in a <code>String</code> field <code>s01</code> . <i>Hint: LocalTime.toString</i> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime currentTime = LocalTime.now();</pre>

## Practice Your Java Level 1

	<pre>         String s01 = currentTime.toString();     } } </pre>
43.	<p>I have the following objects:</p> <pre> LocalDate date01 = LocalDate.of(2025, 12, 15); LocalTime time01 = LocalTime.of(15, 25, 17); </pre> <p>Combine these two to form a single <code>LocalDateTime</code> object and then print out the resulting object.</p> <p><i>Hint: LocalTime.atDate or LocalDate.atTime</i></p> <pre> import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDate date01 = LocalDate.of(2025, 12, 15);         LocalTime time01 = LocalTime.of(15, 25, 17);          LocalDateTime dt01 = time01.atDate(date01);         //OR LocalDateTime dt01 = date01.atTime(time01);          System.out.printf("The combined object is %s\n", dt01);     } } </pre>
44.	<p>Given the two <code>LocalTime</code> objects below, determine whether they are equal or not. If they are, indicate so, otherwise print out which of them is earlier.</p> <pre> LocalTime time01 = LocalTime.of(8, 30, 35); LocalTime time02 = LocalTime.of(14, 42, 9); </pre> <p><i>Hint: LocalTime.isBefore</i></p> <pre> import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime time01 = LocalTime.of(19, 30, 35);         LocalTime time02 = LocalTime.of(14, 42, 9);          if(time01.equals(time02))         {             System.out.println("They are the same time value!");             return;         }         else         {             System.out.println("They are NOT the same time value!");             if(time01.isBefore(time02))                 System.out.println("The 1st time is before the 2nd. ");             else                 System.out.println("The 2nd time is before the 1st");         }     } } </pre> <p><b>Note(s):</b> Also look at the <code>isAfter</code> method.</p>

### Computing time differences using the `ChronoUnit` enumeration and the `until` methods

45.	<p>Determine the difference in seconds between the following <code>LocalTime</code> objects:</p> <pre> LocalTime time01 = LocalTime.of(8, 30, 35); LocalTime time02 = LocalTime.of(14, 42, 9); </pre> <p><i>Hint: LocalTime.until</i></p> <pre> import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalTime time01 = LocalTime.of(8, 30, 35);         LocalTime time02 = LocalTime.of(14, 42, 9);         long diff = time01.until(time02, ChronoUnit.SECONDS);     } } </pre>
-----	---

	<pre>         System.out.printf("The difference is %d seconds\n", diff);     } } </pre>
46.	<p>Print out your age in days.  <i>Hint: LocalDate.until. Difference between your birthday and now.</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          LocalDate birthDate = LocalDate.of(1970,03,01); //for example 1st March, 1970         LocalDate currentDate = LocalDate.now();         long numberofDays = birthDate.until(currentDate, ChronoUnit.DAYS);         System.out.println("I am " + numberofDays + " days old.");     } } </pre>
47.	<p>Print out your age in hours.  <i>Hint: You cannot apply a ChronoUnit.HOURS enumeration to a LocalDate object, but to a LocalDateTime object.</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          LocalDateTime birthDateTime = LocalDateTime.of(1970,03,01,0,0,0);         //for example 1st March, 1970, we presume midnight         LocalDateTime currentDateTime = LocalDateTime.now();         long numberofHours = birthDateTime.until(currentDateTime, ChronoUnit.HOURS);         System.out.println("I am " + numberofHours + " hours old.");     } } </pre>
48.	<p>Calculate how many days there are to your next birthday.  <i>Hint: First determine whether your birthday has already passed this year</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         LocalDate birthDate = LocalDate.of(1970, 03, 01); //for example 1st March, 1970         LocalDate nextBirthDay = LocalDate.of(LocalDate.now().getYear(), birthDate.getMonth(),   birthDate.getDayOfMonth()); // my birthday this year         //Let's see whe if we've passed it or not already         long numberofDays = nextBirthDay.until(LocalDate.now(), ChronoUnit.DAYS);         if(numberofDays &lt; 0) {             nextBirthDay.plusYears(1);             numberofDays = LocalDate.now().until(nextBirthDay, ChronoUnit.DAYS);         }         System.out.println("It is " + numberofDays + " days till my next birth day");     } } </pre>

**Classes and Enums Year, Month, YearMonth, MonthDay and DayOfWeek**

49.	<p>Instantiate a Year object with the value 2025.</p> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {          Year year01 = Year.of(2025);         System.out.printf("The year is %s\n", year01);     } } </pre>
50.	<p>Determine and print out whether the Year object in the preceding exercise is a leap year or not.</p> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {          Year year01 = Year.of(2025);         if(year01.isLeap())             System.out.printf("The year %s is a leap year!\n", year01);     } } </pre>

## Practice Your Java Level 1

	<pre>         else             System.out.printf("The year %s is NOT a leap year!\n", year01);     } } </pre>
51.	<p>Given the <code>Year</code> object below, put its value into an <code>int</code>:</p> <pre> Year year01 = Year.of(2025);  import java.time.*; </pre>
52.	<p>I have two <code>Year</code> objects defined. Write a program which will determine which of these <code>Year</code> objects is before the other and writes this out, otherwise, if they are equal state so. Test your program with values of your choosing.  <i>Hint: Year.isBefore</i></p> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {          Year year01 = Year.of(2025);         Year year02 = Year.of(2050);         if(year01.equals(year02))         {             System.out.println("They are the same year!");             return;         }         else         {             System.out.println("They are NOT the same time value!");             if(year01.isBefore(year02))                 System.out.println("The 1st year is before the 2nd. ");             else                 System.out.println("The 2nd year is before the 1st");         }     } } </pre>
53.	<p>Given a <code>Year</code> object with a value of 2025 and an <code>int</code> with a value of 8 (representing the month August), combine the <code>Year</code> object and the integer value for the month to form a <code>YearMonth</code> object. Print the <code>YearMonth</code> object out.  <i>Hint: Year.atMonth</i></p> <pre> import java.time.*;  public class PracticeYourJava {     public static void main(String[] args) {          Year year01 = Year.of(2025);         int month = 8;         YearMonth ym01 = year01.atMonth(month);         System.out.printf("%s\n", ym01);     } } </pre>
54.	<p>Instantiate a <code>Month</code> enumeration with the value <code>Month.FEBRUARY</code> (representing February). After this, print the following data out:</p> <ol style="list-style-type: none"> <li>the value of the enum constant associated with this month</li> <li>the maximum length (in days) of the month</li> <li>the minimum length (in days) of the month</li> </ol> <p><i>Hint: Month.getValue, minLength, maxLength</i></p> <pre> import java.time.*; </pre>

```

public class PracticeYourJava {
    public static void main(String[] args) {
        Month month01 = Month.FEBRUARY;
        System.out.printf("Enum value = %d\n", month01.getValue());
        System.out.printf("Min #days = %d\n", month01.minLength());
        System.out.printf("Max #days = %d\n", month01.maxLength());
    }
}

```

**Note:** This exercise shows a use of a feature of Java enumeration objects; these enumeration objects have methods.

55. Instantiate a `Month` object with an integer of value 7 (representing the month July). Also, print out the month that is 2 months before this date and the month that is 3 months after it.

```

import java.time.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        Month month01 = Month.of(7);
        System.out.printf("2 months before = %s\n", month01.minus(2));
        System.out.printf("3 months before = %s\n", month01.plus(3));
    }
}

```

56. What is the output of the following code?

```

import java.time.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        Month month01 = Month.valueOf("AUGUST");
        System.out.printf("The month is %s\n", month01);
    }
}

```

The output is: The month is AUGUST

This shows that you can initialize a month object with a `String` that corresponds to the enumeration name of the month in question.

57. What is the length of the month February in a year of your choosing?

*Hint: Month.Length. First check whether the year is a leap year using Year.isLeap*

```

import java.time.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        int specifiedYear = 2024; // just a value for testing purposes
        Month month01 = Month.FEBRUARY;
        int length = month01.length(Year.isLeap(specifiedYear));
        System.out.printf("The length is %d days\n", length);
    }
}

```

58. Initialize a `DayOfWeek` enumeration object to the value `Wednesday` and print out the day 9 days before this one and the day 11 days after it.

*Hint: DayOfWeek\_MINUS, plus*

```

import java.time.*;

public class PracticeYourJava {
    public static void main(String[] args) {
        DayOfWeek day = DayOfWeek.WEDNESDAY;
        System.out.printf("9 days before = %s\n", day.minus(9));
        System.out.printf("12 days after = %s\n", day.plus(11));
    }
}

```

## Practice Your Java Level 1

Taking Time Zones into account: Classes <code>ZonedDateTime</code> , <code>ZoneId</code> , <code>ZoneOffset</code>	
59.	<p>Briefly describe the classes <code>ZonedDateTime</code>, <code>ZoneId</code> and <code>ZoneOffset</code> and the relationship between them.</p> <p>The class <code>ZonedDateTime</code> is used to represent date/time data, specifying the time zone along with the date and time information.</p> <p>Within Java, each time zone has been assigned a name. The class <code>ZoneId</code> represents each of the time zones; as its name implies it has an “id” for a time zone. Now, being that each <code>ZonedDateTime</code> object has a time zone that it is in, that time zone can be extracted from the <code>ZonedDateTime</code> object and assigned to a <code>ZoneId</code> object.</p> <p>The class <code>ZoneOffset</code> represents the offset in hours and minutes of the time zone of a <code>ZonedDateTime</code> object from UTC/GMT.</p> <p><b>Summary:</b> once you have a <code>ZonedDateTime</code> object, you can determine which time zone (this is put into a <code>ZoneId</code> object) it is in and what is the gap in time between this time zone and UTC/GMT (this is put into a <code>ZoneOffset</code> object).</p>
60.	<p>Print out the default time zone of the system on which you are running your programs.</p> <p><i>Hint: <code>ZoneId.systemDefault</code></i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         System.out.printf("The default time zone is %s\n", ZoneId.systemDefault());     } }</pre>
61.	<p>Print out the current date and time, including the current time zone.</p> <p><i>Hint: Use a <code>ZonedDateTime</code> object instead of a <code>LocalDateTime</code> object to determine the current date/time including timezone.</i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zDateTime = ZonedDateTime.now();         System.out.println("The current date &amp; time is " + zDateTime);     } }</pre>
62.	<p>Extract and print out the name of the time zone in which you are from your current <code>ZonedDateTime</code>.</p> <p><i>Hint: <code>ZonedDateTime.getZone</code></i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zDateTime = ZonedDateTime.now();         ZoneId myZone = zDateTime.getZone();         System.out.println("Zone name = " + myZone);     } }</pre>
63.	<p>Extract the <code>ZoneOffset</code> of the zone in which you are from your current <code>ZonedDateTime</code>.</p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zDateTime = ZonedDateTime.now();         ZoneOffset myZoneOffset = zDateTime.getOffset();         System.out.println("Zone offset = " + myZoneOffset);     } }</pre> <p><b>Notes:</b> Observe that the class <code>ZonedDateTime</code> has “get” methods for the same date/time fields as <code>LocalDateTime</code>. We have explicitly extracted only the time zone name and time zone offset in this and the preceding exercise.</p>
64.	<p>Write a program which will determine whether the time zone you are in is behind, ahead or in UTC.</p> <p><i>Hint: <code>ZonedDateTime.getOffset</code>, <code>ZoneOffset.getTotalSeconds</code></i></p> <pre>import java.time.*;</pre>

```

public class PracticeYourJava {
    public static void main(String[] args) {
        ZonedDateTime zDateTime = ZonedDateTime.now();
        ZoneOffset myZoneOffset = zDateTime.getOffset();
        int behindOrBefore = myZoneOffset.getTotalSeconds();
        if(behindOrBefore < 0)
            System.out.println("We are behind GMT.");
        else if(behindOrBefore > 0)
            System.out.println("We are before GMT.");
        else
            System.out.println("We are in GMT.");
    }
}

```

65. Run the program below to observe the list of available time zone names. Comment on the format of the output.

```

import java.time.*;
public class PracticeYourJava {
    public static void main(String[] args) {
        String[] zoneArray = ZoneId.getAvailableZoneIds().toArray(new String[0]);
        for(int i=0; i < zoneArray.length; i++)
            System.out.println(zoneArray[i]);
    }
}

```

Some samples of the time zone names that are output are as follows:

```

Africa/Nairobi
Africa/Cairo
America/Belize
America/New_York
America/Indiana/Petersburg
America/Argentina/San_Juan
Asia/Aden
Asia/Pontianak
Asia/Kuching
CET
Etc/GMT+9
Etc/GMT-1
Europe/London
Europe/Brussels
GMT
Libya

```

It can be observed that the time zone names are presented in one of the following formats:

```

<X>/<Y>
<X>
<X>/<Y>/<Z>

```

- When it is in the format <X>/<Y>, X is often the name of a continent and Y is the name of a city in the continent in question.
- When it is in the format <X>, these are generally the well-known time zones, such as GMT, CET (Central European Time), or even just the name of a country which has a single time zone (for example Libya).
- When it is in the format <X>/<Y>/<Z>, X may represent a continent, Y a country and Z a particular region in the country, or it might represent a country, a state/province therein and then a city/town.

In general each time zone in the world is represented in this list.

66. Explain the relationship of the time zone names to `ZoneId` objects.

The time zone names are textual string representations/descriptions of the different time zones that exist. A `ZoneId` object however is an object which is used to represent a given time zone name; the `ZoneId` object knows the time offset of the zone that it is representing.

67. Obtain a `ZoneId` object for each of the following zone names:

1. Africa/Nairobi

## Practice Your Java Level 1

	<p>2. GMT 3. Europe/London</p> <p><i>Hint: ZoneId.of</i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZoneId zid01 = ZoneId.of("Africa/Nairobi");         ZoneId zid02 = ZoneId.of("GMT");         ZoneId zid03 = ZoneId.of("Europe/London");     } }</pre>
68.	<p>Create a ZonedDateTime object that represents the 15<sup>th</sup> of June, 2019, at 1:15 am in New York, USA.</p> <p><i>Hint: ZonedDateTime.of</i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zdt01 = ZonedDateTime.of(2019,6,15,1,15,0,0, ZoneId.of("America/New_York"));     } }</pre>
69.	<p>If it is 5pm in London, UK now, what is the time in Moscow, Russia?</p> <p><i>Hint: Get a ZonedDateTime object for London, UK and set it to 5pm. Then using the method withZoneSameInstant with a ZoneId of Europe/Moscow, determine the equivalent time in Moscow, Russia.</i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zdt01 = ZonedDateTime.now(ZoneId.of("Europe/London"));         zdt01.withHour(17); //set it to 5:00:00 pm         zdt01.withMinute(0);         zdt01.withSecond(0);         ZonedDateTime zdt02 = zdt01.withZoneSameInstant(ZoneId.of("Europe/Moscow"));         System.out.println("The corresponding time in Moscow = " + zdt02);     } }</pre>

Date/Time Formatting	
70.	<p>I have the following ZonedDateTime object:</p> <pre>ZonedDateTime zdt01 = ZonedDateTime.of(2017,2,9,8,6,4,512, ZoneId.of("America/New_York"));</pre> <p>Print it out in the following ways:</p> <ol style="list-style-type: none"> <li>1. In each of the built-in date/time formats available.</li> <li>2. Only the date portion in each of the built-in date formats available.</li> <li>3. Only the time portion in each of the built-in time formats available.</li> </ol> <p><i>Hint: DateTimeFormatter constants</i></p> <pre>import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zdt01 = ZonedDateTime.of(2017,2,9,8,6,4,512, ZoneId.of("America/New_York"));         System.out.println(zdt01.format(DateTimeFormatter.ISO_LOCAL_DATE_TIME));         System.out.println(zdt01.format(DateTimeFormatter.ISO_OFFSET_DATE_TIME));         System.out.println(zdt01.format(DateTimeFormatter.ISO_ZONED_DATE_TIME));         System.out.println(zdt01.format(DateTimeFormatter.ISO_DATE_TIME));         System.out.println(zdt01.format(DateTimeFormatter.RFC_1123_DATE_TIME));         System.out.println(zdt01.format(DateTimeFormatter.ISO_INSTANT));         System.out.println(zdt01.format(DateTimeFormatter.BASIC_ISO_DATE));         System.out.println(zdt01.format(DateTimeFormatter.ISO_LOCAL_DATE));     } }</pre>

	<pre>         System.out.println(zdt01.format(DateTimeFormatter.ISO_OFFSET_DATE));         System.out.println(zdt01.format(DateTimeFormatter.ISO_DATE));         System.out.println(zdt01.format(DateTimeFormatter.ISO_LOCAL_TIME));         System.out.println(zdt01.format(DateTimeFormatter.ISO_OFFSET_TIME));         System.out.println(zdt01.format(DateTimeFormatter.ISO_TIME));     } } </pre>
71.	<p>I have the following <code>ZonedDateTime</code> object:</p> <pre>ZonedDateTime zdt01 = ZonedDateTime.of(2017,2,9,8,6,4,512, ZoneId.of("America/New_York"));</pre> <p>Write a program to output all of the different ways in which <u>each of the components</u> of this object can be written with the <u>custom</u> date/time format specifiers. The components thereof are the following: year, era (AD/BC), month, day (in month), day of week, hour (1-12, 24 hour format, 0-11 format), AM/PM, minute, seconds, milliseconds and time zone.</p> <p><i>Hint: <code>DateTimeFormatter.ofPattern</code></i></p>
	<pre> import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         LocalDateTime dt01 = LocalDateTime.of(2017, 2, 9, 8, 6, 4, 512);         ZonedDateTime zdt01 = ZonedDateTime.of(dt01,ZoneId.of("America/New_York"));         System.out.println("Year= " + zdt01.format(DateTimeFormatter.ofPattern("yyyy yyy yy y")));         System.out.println("Era= " + zdt01.format(DateTimeFormatter.ofPattern("GG G")));         System.out.println("Month= " + zdt01.format(DateTimeFormatter.ofPattern("MMM MM M")));         System.out.println("Day= " + zdt01.format(DateTimeFormatter.ofPattern("dd d")));         System.out.println("Day of week= " + zdt01.format(DateTimeFormatter.ofPattern("EE E")));         System.out.println("Hour(12 hr format(1-12) = "                 + zdt01.format(DateTimeFormatter.ofPattern("hh h")));         System.out.println("Hour(24 hr format) = "                 + zdt01.format(DateTimeFormatter.ofPattern("HH H")));         System.out.println("Hour(12 hr format(0-11) = "                 + zdt01.format(DateTimeFormatter.ofPattern("KK K")));         System.out.println("AM/PM = " + zdt01.format(DateTimeFormatter.ofPattern("a")));         System.out.println("Minute = " + zdt01.format(DateTimeFormatter.ofPattern("mm m")));         System.out.println("Seconds = " + zdt01.format(DateTimeFormatter.ofPattern("ss s")));         System.out.println("Milliseconds = " + zdt01.format(DateTimeFormatter.ofPattern("SS S")));         System.out.println("Time Zone = " + zdt01.format(DateTimeFormatter.ofPattern("zzzz z")));         System.out.println("RFC 1123 Time Zone = "                 + zdt01.format(DateTimeFormatter.ofPattern("ZZZZ Z")));     } } </pre>
72.	<p>For the <code>ZonedDateTime</code> object in the preceding exercise, print out the following information: what day of the year it corresponds to, what week of the year it falls in, what week of the month it is and what occurrence of the day of the week it is this month (for example, determining that it is the 3<sup>rd</sup> Wednesday of this month).</p> <pre> import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         ZonedDateTime zdt01 = ZonedDateTime.of(2017,2,9,8,6,4, 512,ZoneId.of("America/New_York"));         System.out.println("Day in year = " + zdt01.format(DateTimeFormatter.ofPattern("D")));         System.out.println("The date is in week = "                 + zdt01.format(DateTimeFormatter.ofPattern("w")));         System.out.println("The week of the month is = "                 + zdt01.format(DateTimeFormatter.ofPattern("W")));         System.out.println("It is the nth day of the week this month = "                 + zdt01.format(DateTimeFormatter.ofPattern("F")));     } } </pre>
73.	Write a program which prints out the date 1 <sup>st</sup> March, 2019, 2:15:07 am in the following formats:

## Practice Your Java Level 1

- |  |  |
|--|--|
|  | <ol style="list-style-type: none"><li>1. 2019/3/1 02:15:07</li><li>2. 20190301 02:15:07</li><li>3. 20190301-02:15:07</li></ol> |
|--|--|

```
import java.time.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {  
        LocalDateTime dt02 = LocalDateTime.of(2019, 3, 1, 2, 15, 7);  
        System.out.println(dt02.format(DateTimeFormatter.ofPattern("yyyy/M/d hh:mm:ss")));  
        System.out.println(dt02.format(DateTimeFormatter.ofPattern("yyyyMMdd hh:mm:ss")));  
        System.out.println(dt02.format(DateTimeFormatter.ofPattern("yyyyMMdd'-'hh:mm:ss")));  
    }  
}
```

### Duration & Instant

- |     |  |
|-----|--|
| 74. | We want to measure the time that it takes to sum the values 1 to 1,000,000,000, adding these to an object of class Long. Add these up using a for loop and measure the duration of the addition using objects of the classes Instant and Duration. Print the duration out. |
|-----|--|

Repeat the exercise using the long primitive instead of the class Long.

*Hint: Instant.now, Duration*

```
import java.time.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {  
        int maxNum = 1_000_000_000;  
        Long sum01 = 0L;  
        Instant startInstant = Instant.now();  
        for(int i=0; i < maxNum; i++){  
            sum01+=i;  
        }  
        Instant finishInstant = Instant.now();  
        Duration duration01 = Duration.between(startInstant,finishInstant);  
        System.out.println("The duration is: " + duration01);  
  
        //Now doing it with the primitive long  
        long sum02 = 0;  
        startInstant = Instant.now();  
        for(int i=0; i < maxNum; i++){  
            sum02+=i;  
        }  
        finishInstant = Instant.now();  
        Duration duration02 = Duration.between(startInstant,finishInstant);  
        System.out.println("The duration is: " + duration02);  
    }  
}
```

- |     |   |
|-----|---|
| 75. | We want to determine which is more efficient; the Arrays.sort method or the Arrays.parallelSort method. To wit, write a program which will fill an int array with 20,000,000 random numbers between 1 and 10,000. Sort the array using each of the methods noted above, measuring and outputting the time that it takes to perform each sort. |
|-----|---|

*Remember:* After the array is created, remember to make a copy of it in its original state for the second algorithm to use so that they are sorting the same data.

```
import java.time.*;  
  
public class PracticeYourJava {  
    public static void main(String[] args) {  
  
        //measuring duration of array sorting, using Instant & Duration  
        int numElements = 20_000_000;  
        int[] array01 = new int[numElements];  
        int[] array02; //we will copy array01 to this after we've filled array01 with random numbers
```

	<pre> for(int i=0;i&lt;array01.length;i++)     array01[i] = (int)Math.ceil(Math.random() * 10000);  array02 = Arrays.copyOf(array01, array01.length); //make a copy to use for the other algorithm  // The 2nd sort, using Arrays.Sort Instant startInstant = Instant.now(); Arrays.sort(array01); Instant finishInstant = Instant.now(); Duration duration01 = Duration.between(startInstant,finishInstant); System.out.println("Using Arrays.sort: time to sort: " + numElements + " = " + duration01);  // The 2nd sort, using Arrays.parallelSort startInstant = Instant.now(); Arrays.parallelSort(array02); finishInstant = Instant.now(); duration01 = Duration.between(startInstant,finishInstant); System.out.println("Using Arrays.parallelSort: time to sort: " + numElements + " = " + duration01); } } </pre>
E1	Write a program to calculate and print out how old you are in years, months and days (for example, “35 years, 11 months and 3 days old”). <i>Hint: Investigate the class Period.</i>
E2	Repeat exercise 74 using the <code>System.nanoTime</code> method to measure time differences.



# Chapter 16. Exceptions

The concept of exceptions is a very important one in Java as it provides a means with which to gracefully handle abnormal situations that potentially might occur during program execution. This chapter presents exercises which aid in understanding the concept of exceptions and also help in clarifying the appropriate, necessary and sometimes mandatory placement of exception handling within Java programs.

1.	<p>What is an <i>exception</i>?</p> <p>An exception is an alarm notification raised by the Java Virtual Machine, third party libraries, or even your own application code due to unacceptable situations/conditions in any of the JVM, the third party libraries, your application or even the computer itself. Examples of unacceptable conditions include but are not limited to: attempted division by 0, attempted access beyond array bounds, attempted access of non-existent files/directories, program threads waking up, as well as out of memory errors.</p> <p>Exceptions were designed to ensure that you the programmer handle aberrant situations gracefully.</p> <p>When such conditions arise, the running of the program <u>will actually be aborted</u> by the JVM unless you the programmer have written code to “catch” the exception.</p> <p>In order to “catch” exceptions, the code that is liable to “throw” exceptions must be contained in what is called a <b>try</b> block, as exceptions are only caught when you actually <i>try</i> to catch them.</p> <p>You should also note that exceptions are actually objects (with all the attendant properties of objects) of classes descended from the class <b>Exception</b>.</p>
2.	<p>Why should you make sure that you catch every exception that might be thrown?</p> <ol style="list-style-type: none"><li>One key reason for catching every exception is that the program stops running when an exception is thrown but not explicitly caught! Abnormal program termination is not acceptable to end users.</li><li>Catching exceptions ensures that the application has an opportunity to recover from errors gracefully, ensuring that the end user at worst suffers only minimal loss due to the exception condition(s).</li></ol>
3.	<p>Run the program below and observe the exception that it will produce.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int a = 5;         int b = 0;         System.out.println(a / b);         System.out.println("After the calculation\n");     } }</pre> <p>This will produce a <code>java.lang.ArithmaticException</code> exception. The JVM will also state that the actual issue is “/ by zero”.</p> <p>The program will not even get to the line which outputs the string “<i>After the calculation</i>”.</p>
4.	<p>Modify the program in the preceding exercise to catch the exception that was observed. Your exception <b>catch</b> block should print out the statement “<i>I caught the exception!</i>”</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int a = 5;         int b = 0;         try{             System.out.println(a / b);         }         catch(ArithmaticException e){             System.out.println("I caught the exception!");         }         System.out.println("After the calculation\n");     } }</pre>

## Practice Your Java Level 1

	}
5.	<p>What is the resulting exception from this code that is trying to access an uninitialized object reference?</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         String s = null;         int length = s.length();     } }</pre> <p>The exception thrown is a <a href="#">java.lang.NullPointerException</a> exception.</p>
6.	<p>Modify the program in the preceding exercise to catch the exception in question and also to print out the name of the caught exception.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         try{             String s = null;             int length = s.length();         }         catch(NullPointerException e){             System.out.println("Exception caught:" + e);         }     } }</pre>
7.	<p>State which exception is produced by the code shown below.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int x1 = Integer.MAX_VALUE;         int x2 = Integer.MAX_VALUE - 1000;         int x3 = Math.addExact(x1, x2);     } }</pre> <p>The exception thrown is a <a href="#">java.lang.ArithmaticException</a> exception, the JVM also stating that it is an “integer overflow”.</p>
8.	<p>Modify the program in the preceding exercise to catch the thrown exception and print out the text “<i>I caught the overflow exception!</i>” in the exception handler.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         try{             int x1 = Integer.MAX_VALUE;             int x2 = Integer.MAX_VALUE - 1000;             int x3 = Math.addExact(x1, x2);         }         catch(ArithmaticException e){             System.out.println("I caught the overflow exception!");         }     } }</pre>
9.	<p>What exception does the following code trying to access an out of bounds array index throw?</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 14;         int[] iArray = new int[5];         iArray[arrayIndex] = 1500;         System.out.println("After the array index access attempt");     } }</pre>

	<pre>}</pre> <p>This will throw a <code>java.lang.ArrayIndexOutOfBoundsException</code> exception.</p>
10.	<p>Modify the program in the preceding exercise to catch the exception and print out the text “<i>I caught the exception!</i>” in the exception handler.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 14;         int[] iArray = new int[5];          try{             iArray[arrayIndex] = 1500;         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println("I caught the exception!");         }         System.out.println("After the array index access attempt");     } }</pre>
11.	<p>Modify the exception handler in the solution to the preceding exercise to print out the following information about the exception from the exception object:</p> <ol style="list-style-type: none"> <li>1. Exception message</li> <li>2. Exception cause</li> <li>3. Stack Trace at the point of the exception</li> </ol> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 14;         int[] iArray = new int[5];          try{             iArray[arrayIndex] = 1500;         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println(e.getMessage());             System.out.println(e.getCause());             e.printStackTrace();         }     } }</pre>

12.	<p><i>Deliberately throwing an exception</i></p> <p>Modify the program in exercise 10 to throw the exception <code>ArrayIndexOutOfBoundsException</code> whenever the user tries to access the array position 2. The exception message should be “<i>Attempt to access array index 2</i>”.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 2; //set it to 2 for our test         int[] iArray = new int[5];          if(arrayIndex == 2) {             throw new ArrayIndexOutOfBoundsException("Attempt to access array index 2");         }         try{             iArray[arrayIndex] = 1500;         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println("I caught the exception!");         }         System.out.println("After the array index access attempt");     } }</pre>
-----	---

## Practice Your Java Level 1

13.	<p><i>User-defined exceptions: creating your own exception class</i></p> <p>Java provides the facility for you to define your own custom exception classes. These look like any other exception class, in that they must be derived from the class <code>Exception</code> or any subclass thereof. With that in mind, we have the following exercise:</p> <p>Create a custom exception named <code>DivideBy2Exception</code>, of superclass <code>Exception</code>. (Do not add any substantive code to the methods therein).</p> <p><i>Note:</i> If you are using the Eclipse IDE, it is easiest to do this using the class creation template, simply specifying in the window that the superclass is <code>Exception</code>; the IDE will present the stubs of the inherited methods to you.</p> <pre>public class DivideBy2Exception extends Exception {     public DivideBy2Exception() {}     public DivideBy2Exception(String message) {         super(message);     }     public DivideBy2Exception(Throwable cause) {         super(cause);     }     public DivideBy2Exception(String message, Throwable cause) {         super(message, cause);     }     public DivideBy2Exception(String message, Throwable cause,         boolean enableSuppression, boolean writableStackTrace) {         super(message, cause, enableSuppression, writableStackTrace);     } }</pre>
14.	<p><i>Throwing a user-defined exception</i></p> <p>Write a program which divides a given <code>float</code> value by another (hardcode the values for testing). Throw your earlier created exception <code>DivideBy2Exception</code> when the divisor is divisible by 2.</p> <p>Have the <code>catch</code> block for the exception print out the following:</p> <ol style="list-style-type: none"><li>the <code>Message</code> field of the exception</li><li>the phrase “Divisor divisible by 2 exception”</li></ol> <pre>public class ExceptionTest {     public static void main(String[] args) {         float a = 10f, b = 2f;         float c;         try{             if ((b % 2) == 0){                 throw new DivideBy2Exception("Divisor divisible by 2 exception");             }             c = a / b;             System.out.println(c);         }         catch (DivideBy2Exception e){             System.out.println(e);         }         catch(ArithmaticException e){             System.out.println(e);         }     } }</pre>

<i>Checked Exceptions (Mandatory-catch exceptions)</i>	
15.	<p>Explain the concept of “checked exceptions”.</p> <p>There are certain scenarios where Java insists that you make provision for catching any exceptions that may be thrown, otherwise the compiler will not even compile your code. File handling is one key area of Java programming where you are mandated to catch for exceptions.</p>
16.	<p>Explain why the code below which attempts to create a symbolic link to a non-existent file will not compile and state what is required for it to compile.</p> <pre>import java.io.IOException; import java.nio.file.Files; import java.nio.file.Path; import java.nio.file.Paths;  public class ExceptionTest {      public static void A(){         String fileName      = "ThISfileDoesNotExxist";         String symLinkName  = "lnk01";          Path targetPath   = Paths.get(System.getProperty("user.home"), "Desktop", fileName);         Path symlinkPath = Paths.get(System.getProperty("user.home"), "Desktop", symLinkName);          Files.createLink(symlinkPath, targetPath);     }      public static void main(String[] args) {         A();     } }</pre> <p>The code will not compile because it contains methods whose potential exceptions Java demands that we handle. I/O operations fall into that category of “you must catch” exceptions; Java calls these “checked exceptions”. There are two ways to handle them and these are:</p> <ol style="list-style-type: none"> <li>1. standard try/catch handling</li> <li>2. the method in which the methods that can throw these “checked exceptions” appear must explicitly have a <code>throws</code> statement in its method declaration. For example, the header of the method <code>A()</code> above in which the method <code>Files.createLink</code> which can throw such an exception is present would be modified as follows:</li> </ol> <pre>public static void A() throws IOException {</pre> <p>If this mechanism is chosen, it implies that somewhere in the call stack for method <code>A()</code> the exception(s) in question will be handled. In this particular example, the exception <u>must</u> be handled in the calling method, <code>main()</code>. Note that that it is still valid to apply <code>try/catch</code> blocks to catch the same exception directly within the method.</p> <p>The benefit of this mechanism is that it facilitates exception handling in a central place, without <code>try</code> blocks scattered all over the code. (Exercise 22 shows the catching of an exception in a given method by an exception handler in its calling method).</p> <p><b>Notes</b></p> <ol style="list-style-type: none"> <li>1. Restatement: Even with this second mechanism, it is still valid to apply <code>try/catch</code> blocks within the method which uses this mechanism.</li> <li>2. If <code>main()</code> is the method with the <code>throws &lt;exception&gt;</code> clause in its header, then clearly if the exception is not handled within <code>main()</code> itself, the program will halt running when the exception occurs.</li> </ol>

## Practice Your Java Level 1

17.	<p>Explain the compilation error from the following code:</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 10;         int[] iArray = new int[5];         try{             iArray[arrayIndex] = 1500;         }         catch(Exception e){             System.out.println("I caught the exception!");         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println("I caught the exception!");         }         System.out.println("After the array index access attempt");     } }</pre> <p>The compiler will let you know that the 2<sup>nd</sup> exception handler cannot be reached due to the fact that there is a preceding <b>catch</b> that is capable of handling what the 2<sup>nd</sup> exception handler can catch (due to the fact that the exception class of the 1<sup>st</sup> catch block is a superclass to the exception class of the 2<sup>nd</sup> catch block), thus making the 2<sup>nd</sup> exception catcher “unreachable code”.</p>
18.	<p>Exchange the position of the exception handlers above and explain the result of the compilation.</p> <p>The code compiles and runs without any problem. The reason for this is that no longer is the exception handler that handles the exception <b>ArrayIndexOutOfBoundsException</b> unreachable code.</p> <p>Now, does this mean that the exception handler that handles the exception <b>Exception</b> is unreachable code? No, because at compile time the compiler does not know what exceptions might result from the <b>try{} block</b> and thus does not know whether the catch block that catches <b>ArrayIndexOutOfBoundsException</b> will catch every exception that might result. And since the catch block that catches <b>ArrayIndexOutOfBoundsException</b> will not cause the code block that catches <b>Exception</b> to be unreachable, the compiler has nothing to complain about.</p> <p>Also it can be seen from this example that multiple exception catchers for different possible exceptions can be applied to a single <b>try</b> block.</p>
19.	<p><b>finally</b></p> <p>Describe what a <b>finally{}</b> code block does with respect to exception handling.</p> <p>A <b>finally{}</b> block is a block of code which is run after a series of <b>try/catch</b> blocks, whether or not the contents of the <b>try/catch</b> blocks throw exceptions or not.</p> <p>The <b>finally</b> block is a good place to put resource cleanup code, for example, code for releasing file handles, code for releasing network connections and suchlike.</p>
20.	<p>Modify the code below by adding a <b>finally</b> block to it to print out the date and time at which the code was run.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 10;         int[] iArray = new int[5];         try{             iArray[arrayIndex] = 1500;         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println("I caught the exception!");         }     } }</pre> <pre>public class ExceptionTest {     public static void main(String[] args) {         int arrayIndex = 10;         int[] iArray = new int[5];         try{             iArray[arrayIndex] = 1500;         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println("I caught the exception!");         }     } }</pre>

	<pre>         }         catch(ArrayIndexOutOfBoundsException e){             System.out.println("I caught the exception!");         }         finally{             System.out.println(LocalDateTime.now());         }     } } </pre> <p><b>Note:</b> You can modify the array index to bring it within bounds and rerun the code to confirm that indeed the <code>finally</code> block is run whether or not the <code>try/catch</code> block was invoked.</p>
21.	<p>Predict and explain the output of the program below.</p> <pre> public class ExceptionTest {     public static void A(){         int a = 10, b = 0;         int result;          try{             result = a / b;         }         catch (ArithmaticException e){             System.out.println("A divide by zero situation has occurred. Rethrowing this exception!");             throw(e);         }     }      public static void main(String[] args) {         try{             A();         }         catch (ArithmaticException e){             System.out.println("In main...exception caught.");         }     } } </pre> <p><b>Output</b>  A divide by zero situation has occurred. Rethrowing this exception!  In main...exception caught.</p> <p><b>Observation/Explanation</b>  The call to method <code>A()</code>, called from <code>main()</code> has a <code>try</code> block in <code>main</code> around it. When <code>A()</code> catches an exception and “rethrows” it using the method <code>throw</code>, the <code>try</code> block in <code>main</code> recatches the same exception.  From this exercise the following can be observed:</p> <ol style="list-style-type: none"> <li>1. <code>try/catch</code> blocks that catch rethrown exceptions (or in fact non-rethrow exceptions, see next exercise) do not have to be local to the method in which the rethrow occurs.</li> <li>2. When designing a program, you can have a <code>try</code> block in your method <code>main</code> that can serve as a catch-all for any exceptions not caught elsewhere in your program.</li> </ol>
22.	<p>Predict and explain the output of the program below.</p> <pre> public class ExceptionTest {     public static void A() {         int a = 10, b = 0;         int result;          result = a / b;     }      public static void main(String[] args) {         try{             A();         }     } } </pre>

## Practice Your Java Level 1

	<pre>        catch (ArithmException e){             System.out.println("In main...exception caught.");         }     } } </pre> <p><b>Output</b> In main...exception caught.</p> <p><b>Observation/Explanation</b> An exception condition was triggered in the method A(), but caught in another method, main(), which had invoked A() from within a try block. This scenario further emphasizes the fact as displayed in the preceding exercise that a calling method can actually catch uncaught exceptions in the methods that it calls.</p>
23.	<p>Predict and explain the output of the program below.</p> <pre>public class ExceptionTest {     public static void B() {         int a = 10, b = 0;         int result;          result = a / b;     }      public static void A() {         B();     }      public static void main(String[] args) {         try{             A();         }         catch (ArithmException e){             System.out.println("In main...exception caught.");         }     } } </pre> <p><b>Output</b> In main...exception caught.</p> <p><b>Observation/Explanation</b> An exception condition was triggered in the method B() but caught in another method, main(), which had invoked A() which then called B() from within a try block. This is another exercise that shows that a try block can be placed anywhere in the call stack of the exception generating method and the try block will catch the exception.</p>
24.	<p><i>Rethrowing exceptions (again)</i></p> <p>Predict the output of the program below, then run it and explain the reason for the output.</p> <pre>public class ExceptionTest {     public static void main(String[] args) {         try{             int a = 10, b = 0;             int result;              try{                 result = a / b;             }             catch (ArithmException e) {                 System.out.print("Divide by zero situation has occurred. ");                 System.out.println("Rethrowing this exception.");                 throw(e);             }         } // end of outer try block     } } </pre>

	<pre>         catch (ArithmaticException e) {             System.out.println("Outer try block...exception caught.");         }     } }  <b>Output</b> Divide by zero situation has occurred. Rethrowing this exception! Outer try block...exception caught. </pre> <p><b>Explanation</b></p> <p>This code shows the concept of “rethrowing” an exception and how this works in conjunction with nested <code>try</code> blocks. The following points are noted in this code:</p> <ol style="list-style-type: none"> <li>1. The inner <code>try</code> block has a corresponding <code>catch</code> block that rethrows the caught exception simply by using the keyword <code>throw</code>.</li> <li>2. The outer <code>try</code> block, which catches the rethrown exception.</li> </ol>
25.	<p>Modify the catch for the outer <code>try</code> block in the exercise above to catch exceptions of class <code>Exception</code>. Run the modified code and explain the results.</p> <p>The output is the same as in the preceding solution. The reason for this is that <code>Exception</code> (also known as <code>System.Exception</code>) is the parent exception class and therefore its handler can act as a catch-all for whatever is not caught.</p>
26.	<p>Give a scenario in which you might want to implement the rethrowing and catching of an exception.</p> <p>A scenario where this might occur would be in a case where you have local handling for the exception in the method where it occurs, but then at a higher level you have more general handling for the exception in question.</p>
27.	<p>Give a reason/scenario where nested <code>try</code> blocks would be of value to your code.</p> <p>We might, for example, have inner <code>try</code> blocks in order to catch all exception scenarios, however perhaps not all exception scenarios have been foreseen. An outer <code>try</code> block that catches the parent exception <code>Exception</code> will help your application to handle any such failure gracefully.</p>
28.	<p>Write a program which does the following:</p> <ol style="list-style-type: none"> <li>1. Gets the current time</li> <li>2. Sleeps for 10 seconds</li> <li>3. Rereads the time</li> <li>4. Prints the difference in milliseconds between the 1<sup>st</sup> time reading and the 2<sup>nd</sup> time reading</li> </ol> <p><i>Hint: DateTime, Thread.Sleep, ChronoUnit.MILLIS, exception handling</i></p> <pre> import java.time.*; public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Testing waiting in a program");         System.out.println ("Will sleep for 10secs (10000msecs)");         LocalDateTime dtBefore = LocalDateTime.now();          try{             Thread.sleep(10000);         }         catch(InterruptedException e){         }         LocalDateTime dtAfter = LocalDateTime.now();         System.out.println("Done. Elapsed milliseconds=" +                            dtBefore.until(dtAfter,ChronoUnit.MILLIS));     } } </pre>
29.	<p><i>Try with resources</i></p> <p>Note: The reader who is not familiar with File I/O should still read this example, as the use of the <code>try with resources</code> feature will become clearer on using the classes that have implemented the interface <code>AutoCloseable</code> of package <code>java.lang</code>. (We discuss File I/O and interfaces in later chapters).</p> <p>Rewrite the following code to use a <code>try with resources</code> statement:</p> <pre> import java.io.BufferedReader; import java.io.IOException; import java.nio.file.Files; import java.nio.file.Path; import java.nio.file.Paths; </pre>

## Practice Your Java Level 1

```
public class ExceptionTest {  
    public static void main(String[] args){  
        String fileToRead = "rhyme.txt";  
        Path sourcePath = Paths.get(System.getProperty("user.home"), "Desktop", fileToRead);  
        String nextLine;  
        BufferedReader nbr;  
  
        try{  
            nbr = Files.newBufferedReader(sourcePath);  
            while((nextLine = nbr.readLine()) != null){  
                System.out.println(nextLine);  
            }  
            nbr.close();  
        }  
        catch(IOException e){  
            System.out.println("IOException = " + e);  
        }  
    }  
}  
  
public class ExceptionTest {  
    public static void main(String[] args){  
        String fileToRead = "rhyme.txt";  
        Path sourcePath = Paths.get(System.getProperty("user.home"), "Desktop", fileToRead);  
        String nextLine;  
        try(BufferedReader nbr = Files.newBufferedReader(sourcePath)){  
            while((nextLine = nbr.readLine()) != null){  
                System.out.println(nextLine);  
            }  
        }  
        catch(IOException e){  
            System.out.println("I/O Exception noted");  
        }  
    }  
}
```

### Notes

The differences between the code in the question and the code in the solution are:

1. The instantiation of the `BufferedReader` object was done directly within the `try` block.
2. Also, there is no longer any need for to explicitly close the `BufferedReader` resource using its `close` method.

The reason why it is possible to apply a “try with resources” statement to a `BufferedReader` object is because the `BufferedReader` class implements the interface `AutoCloseable`. Any class that implements the interface `AutoCloseable` is a candidate for a try with resources statement.

# Chapter 17. Console Handling II: Console Input

There are times when console applications require user interaction, this via keyboard input. This necessitates our proper understanding of how to request and interpret data that is input at the keyboard. The data read at the keyboard is always received by the computer as a string of characters (whether the data is numeric or not) and thus it is necessary to know how to convert non-string input (*numerical input in particular*) to their proper type for use. The `valueOf` and `parse<type>` methods that are present in each of the numerical wrapper classes provide means with which to implement this conversion.

The exercises in this chapter call for the use of either of the two classes `Console` or `Scanner` to receive user input.

The exercises also leverage and further your skills in conditional processing and string and number handling.

**Note:** The exercises in this chapter must be run directly from the command line and not from within an IDE.

1.	Request and receive from the user, using a <code>Console</code> object, their first name and then their last name. Afterwards print out the following: “ <i>Your name is &lt;firstName&gt; &lt;lastName&gt;</i> ”. <i>Hint: System.console of package java.io.Console.</i>
	<pre>import java.io.Console;  public class ConsoleHandling {     public static void main(String[] args){         Console consoleObject = System.console();         if(consoleObject != null){             System.out.print("Please enter your first name: ");             String firstName = consoleObject.readLine();             System.out.print("Please enter your last name: ");             String lastName = consoleObject.readLine();             System.out.printf("\nYour name is %s %s", firstName, lastName);         }         else             System.out.println("A console object was not obtained successfully.");             System.out.println("Ensure that this program is being run directly on the command line.");     } }</pre>
2.	Using a <code>Console</code> object to receive the entered data, request and receive a first and then a second string from the user. If the strings are the same, output the statement “ <i>The strings are the same!</i> ”, otherwise output the statement “ <i>The strings are different!</i> ”.

*Hint: String.contentEquals*

```
import java.io.Console;

public class ConsoleHandling {
    public static void main(String[] args){
        Console consoleObject = System.console();
        if(consoleObject==null){
            System.out.println("A console object was not obtained successfully");
            System.exit(0);
        }

        System.out.print("Please enter your the 1st String: ");
        String str01 = consoleObject.readLine();
        System.out.print("Please enter your the 2nd String: ");
        String str02 = consoleObject.readLine();

        boolean comparisonValue;
        comparisonValue = str01.contentEquals(str02);
```

## Practice Your Java Level 1

	<pre> if(comparisonValue == false){     System.out.println("The Strings are different!"); } else     System.out.println("The Strings are the same!"); } } </pre>
3.	<p>Redo the preceding exercise, this time ignoring the case of the entered strings.  <i>Hint: String.equalsIgnoreCase</i></p> <p>Simply replace the following line in the solution to the preceding exercise:</p> <pre>comparisonValue = str01.contentEquals(str02);</pre> <p>with the following line:</p> <pre>comparisonValue = str01.equalsIgnoreCase(str02);</pre>
4.	<p>Repeat the preceding exercise, modifying it in the following fashion: If the strings are the same, but of different case state, “<i>They are equal, although the case is different!</i>”, otherwise if the strings are the same and the same case state, “<i>They are equal and the case is the same!</i>”.</p> <pre> import java.io.Console;  public class ConsoleHandling {     public static void main(String[] args){         Console consoleObject = System.console();         if(consoleObject==null){             System.out.println("A console object was not obtained successfully");             System.exit(0);         }         System.out.print("Please enter your the 1st String: ");         String str01 = consoleObject.readLine();         System.out.print("Please enter your the 2nd String: ");         String str02 = consoleObject.readLine();          boolean comparisonValue;         boolean caseCheck;         caseCheck      = str01.contentEquals(str02); //Will tell us if they are different         comparisonValue = str01.equalsIgnoreCase(str02);         if(comparisonValue == false){             System.out.println("The Strings are different!");             System.exit(0);         }         if(caseCheck == true)             System.out.println("They are equal and the case is the same!");         else             System.out.println("The Strings equal although case is the differemt!");     } } </pre>
5.	<p>Have the user enter a first and then a second string. Print out which appears first alphabetically. If they are both the same string, then inform the user as to that fact. Ignore the case of the strings.  <i>Hint: String.compareToIgnoreCase</i></p> <pre> import java.io.Console;  public class ConsoleHandling {     public static void main(String[] args){         Console consoleObject = System.console();         if(consoleObject==null){             System.out.println("A console object was not obtained successfully");             System.exit(0);         }         System.out.print("Please enter your the 1st String: ");         String str01 = consoleObject.readLine();         System.out.print("Please enter your the 2nd String: ");         String str02 = consoleObject.readLine();     } } </pre>

	<pre> int comparisonValue; comparisonValue = str01.compareToIgnoreCase(str02); if(comparisonValue &lt; 0)     System.out.printf("%s is before %s alphabetically",str01, str02); else if(comparisonValue &gt; 0)     System.out.printf("%s is before %s alphabetically ",str02, str01); else     System.out.println("The Strings are the same!"); } } </pre>
6.	<p>What data type is all console input data interpreted as and what is the implication for receiving and processing numerical input from the console?</p> <p>All console input is received and interpreted as string data. Even when you type a number, for example if you type the number “22”, it is seen and received as a string of two contiguous characters with no semantic meaning! The implications of the console receiving and interpreting numerical data as string data is that the data has to be explicitly converted to a numerical type. There are methods in Java which provide this conversion, for example the method <code>Integer.valueOf</code> which converts integer values in string form to <code>int</code> values, <code>Float.valueOf</code> which does the same for <code>float</code> values, as well as the methods <code>Long.valueOf</code> and <code>Double.valueOf</code> that apply to their respective primitive types.</p>
7.	<p>Using a <code>Console</code> object to receive the entered data, ask the user in succession for two integer values, then add them and print out the sum. If the entered data are not integers the calculation cannot be done, therefore in such a case output a statement indicating which number/numbers is/are in error and then exit the program</p> <p><i>Hint: use the <code>Integer.valueOf</code> method to convert the numbers entered at the console to integers</i></p> <pre> import java.io.Console;  public class ConsoleHandling {     public static void main(String[] args){         Console consoleObject = System.console();         if(consoleObject==null){             System.out.println("A console object was not obtained successfully...exiting");             System.exit(0);         }          System.out.print("Please enter the 1st integer: ");         String sValue01 = consoleObject.readLine();         System.out.print("Please enter the 2nd integer: ");         String sValue02 = consoleObject.readLine();          int value01=0, value02=0;         boolean isValidNum01 = true;         boolean isValidNum02 = true;          try{             value01 = Integer.valueOf(sValue01);         }         catch(NumberFormatException e){             isValidNum01 = false;         }          try{             value02 = Integer.valueOf(sValue02);         }         catch(NumberFormatException e){             isValidNum02 = false;         }          if(isValidNum01 != true){             System.out.println("The 1st value entered is not an integer");         }         if(isValidNum02 != true){             System.out.println("The 2nd value entered is not an integer");         }     } } </pre>

## Practice Your Java Level 1

	<pre> if((isValidNum01==false)  isValidNum02==false){     System.out.println("...therefore cannot do the computation. Exiting.");     System.exit(0); } System.out.printf("The sum of the entered numbers is: %d", (value01+value02)); } } </pre>
8.	<p>We would like to convert a user entered temperature from Celsius to Fahrenheit or vice-versa. Ask the user to enter a temperature in the format &lt;temperature&gt;&lt;F C&gt;, then convert the entered value to the other temperature scale and print it out. For example, if the user enters 20C, detect that it is in Celsius and convert it to its equivalent in Fahrenheit, stating “Equivalent temp = 68F”.</p> <pre> import java.io.Console;  public class ConsoleHandling {     public static void main(String[] args){         Console consoleObject = System.console();         if(consoleObject==null){             System.out.println("A console object was not obtained successfully...exiting");             System.exit(0);         }          String tempScale="";         System.out.print("Enter temperature:");         String sInput = consoleObject.readLine();         sInput = sInput.toLowerCase();         if(sInput.contains("f"))             tempScale = "f";         else if (sInput.contains("c"))             tempScale = "c";         else{             System.out.print("You have to specify C or F");             System.exit(0);         }          //Now extract only the number value from the temperature         //do that by getting rid of "f", "c" and " "         sInput = sInput.replace("f","");         sInput = sInput.replace("c","");         sInput = sInput.replace(" ", "");         double d01=0;          try{             d01 = Double.valueOf(sInput);         }         catch(NumberFormatException e){             System.out.println("The value entered is not a valid numerical value...exiting...");             System.exit(0);         }          if (tempScale.contentEquals("f")){             System.out.printf("Equivalent temp = %fC", (d01 - 32) * ((float)5 / (float)9));         }         else             System.out.printf("Equivalent temp = %fF", (d01 * (float)9 / (float)5) + 32);     } } </pre>

### Input using the class Scanner

An alternative to using objects of class `Console` for console input is the class `Scanner`. Some exercises that use this class follow.

9.	<p>Explain what each of the numbered lines of code in the following example is doing:</p> <pre> import java.util.*; public class ConsoleHandling {     public static void main(String[] args){ </pre>
----	---

	<pre> System.out.println("Enter a String: "); #1: Scanner scanner01 = new Scanner(System.in); #2: String s = scanner01.nextLine();     System.out.println(s); #3: scanner01.close(); } } </pre> <p><b>#1:</b> A Scanner object <code>scanner01</code> has been created and is pointed to the object <code>System.in</code>, which by default refers to the keyboard.</p> <p><b>#2:</b> In this line we are requesting that our <code>Scanner</code> object read a line from the keyboard and that that line be put into the <code>String</code> variable <code>s</code>.</p> <p><b>#3:</b> We must apply the <code>close</code> method to the <code>Scanner</code> object; if not it is still “connected” to <code>System.in</code> and receiving keyboard input.</p>
10.	<p>Explain what each of the numbered sections of code in the following example is doing:</p> <pre> import java.util.Scanner; public class ConsoleHandling {     public static void main(String[] args){         String s = "Hello how are you?";         #1: Scanner scanner01 = new Scanner(s);         #2: while(scanner01.hasNext()){             String token = scanner01.next();             System.out.println(token);         }         #3: scanner01.close();     } } </pre> <p><b>#1:</b> We obtain a <code>Scanner</code> object <code>scanner01</code> using the <code>String s</code> as its input. Comparing this to the previous exercise, we see that we can obtain a <code>Scanner</code> against different types of input.</p> <p><b>#2:</b> A <code>Scanner</code> object is capable of breaking input data into tokens according to what you the user specify. Its default delimiter is whitespace. We can then go through the tokens one by one, using a <code>while</code> loop and the <code>hasNext</code> method of the <code>Scanner</code> class. In this example we are looping through the tokens while there is yet another one to obtain. We then print out each token.</p> <p><b>#3:</b> As stated in the previous exercise, we must apply the <code>close</code> method to the <code>Scanner</code> object; if not, it is still “connected” to the <code>String s</code>. Even though in this particular exercise there are no deleterious effects to not closing the <code>Scanner</code> object, we should do so in order to release resources.</p> <p>The reader is encouraged to read the documentation on the <code>Scanner</code> class in order to see the different methods that can be used to access <code>Scanner</code> tokens.</p>
11.	<p>Ask the reader for a set of tokens and then, using a <code>Scanner</code> object, extract each of the tokens and print them out one by one.</p> <pre> import java.util.*; public class ConsoleHandling {     public static void main(String[] args){         System.out.println("Enter a String: ");         Scanner scanner01 = new Scanner(System.in);         String s = scanner01.nextLine();         Scanner scanner02 = new Scanner(s); //use the line of tokens received as the input for  //a new scanner object         while(scanner02.hasNext()){             String z = scanner02.next();             System.out.println(z);         }         scanner01.close();         scanner02.close();     } } </pre>

## Practice Your Java Level 1

12. Ask the user to enter a set of integers (space separated) and then add them up. Ensure that you skip over non-integers in the entered data. Use a `Scanner` object to receive the data from the user.

*Hint: `Scanner.nextInt`*

```
import java.util.*;  
  
public class ConsoleHandling {  
    public static void main(String[] args){  
        System.out.println("Enter a set of integers to be added up: ");  
        Scanner scanner01 = new Scanner(System.in);  
        String s = scanner01.nextLine();  
        Scanner scanner02 = new Scanner(s);  
        int scannedInt = 0;  
        long sum=0;  
        while(scanner02.hasNext()){  
            try{  
                scannedInt = scanner02.nextInt();  
                sum+=scannedInt;  
            }  
            catch(InputMismatchException e){  
                //skip the unwanted token  
                scanner02.next();  
            }  
            catch(IllegalStateException e){  
                break;  
            }  
        }  
        scanner01.close();  
        scanner02.close();  
        System.out.println("Total = " + sum);  
    }  
}
```

13. Enhance the solution to the preceding exercise, this time, performing an infinite loop requesting for the numbers until the word “quit” in any case is seen in the input.

```
import java.util.*;  
  
public class ConsoleHandling {  
    public static void main(String[] args){  
        Scanner scanner01 = new Scanner(System.in);  
        while(true){  
            System.out.print("Enter a set of integers to be added up: ");  
            String s = scanner01.nextLine();  
            String s_lower_case = s.toLowerCase();  
            if(s_lower_case.contains("quit")){  
                System.out.println("\\"Quit" detected in input...bye bye!");  
                break;  
            }  
            Scanner scanner02 = new Scanner(s);  
            int scannedInt = 0;  
            long sum=0;  
            while(scanner02.hasNext()){  
                try{  
                    scannedInt = scanner02.nextInt();  
                    sum+=scannedInt;  
                }  
                catch(InputMismatchException e){  
                    //skip the unwanted token  
                    scanner02.next();  
                }  
                catch(IllegalStateException e){  
                    break;  
                }  
            }  
        }  
    }  
}
```

	<pre>         }         System.out.println("Total = " + sum);         scanner02.close();     }     scanner01.close(); } </pre>
14.	<p>Ask the user for their first name and then for their last name and respond with a statement stating “Your full name is &lt;last name (only 1<sup>st</sup> letter in uppercase)&gt;, &lt;first name (only 1<sup>st</sup> letter in uppercase)&gt;”.</p> <p>For example, if the user enters a first name of “jay” and a last name of “doe”, print “Your full name is Doe, Jay”.</p> <p><i>Hint: Identify the substring of each entry that you want as uppercase and lowercase respectively (String.substring)</i></p> <pre> import java.util.*;  public class ConsoleHandling {     public static void main(String[] args) {          String firstName = ""; // declare a String that will receive the value         String lastName = ""; // declare a String that will receive the value         String modifiedFirstName, modifiedLastName;         System.out.print("What is your first name please? ");          Scanner in = new Scanner(System.in);          firstName = in.next();         modifiedFirstName = firstName.substring(0,1).toUpperCase() +                            firstName.substring(1,firstName.length()).toLowerCase();         System.out.print("What is your last name please? ");         lastName = in.next();         modifiedLastName = lastName.substring(0,1).toUpperCase() +                            lastName.substring(1, lastName.length()).toLowerCase();         System.out.printf("Your full name is %s, %s", modifiedLastName, modifiedFirstName);         in.close();     } } </pre>
15.	<p>Ask the user to input a string of data, where the tokens are any of the following data types: boolean, short integer, integer, long integer, float, double, decimal, big integer, big decimal or string.</p> <p>Determine what type each of the tokens are and print the type out.</p> <p><i>Note:</i> Pay careful attention to the order of handling for floating point numbers, as, for example, on seeing a value that should legitimately be a <code>BigDecimal</code>, <code>hasNextFloat</code> assumes that it has seen a <code>float</code> of infinite value.</p> <pre> import java.util.*;  public class ConsoleHandling {     public static void main(String[] args){          Scanner scanner01 = new Scanner(System.in);          System.out.println("Please enter a set of tokens");         String s = scanner01.nextLine();         System.out.println("The input String = " + s);          Scanner scanner02 = new Scanner(s); // apply a scanner to the String          while(scanner02.hasNext()){             if(scanner02.hasNextBoolean())                 System.out.println(scanner02.nextBoolean() + " = boolean");             else if(scanner02.hasNextByte())                 System.out.println(scanner02.nextShort() + " = byte");             else if(scanner02.hasNextShort())                 System.out.println(scanner02.nextShort() + " = short");             else if(scanner02.hasNextInt())                 System.out.println(scanner02.nextInt() + " = integer");             else if(scanner02.hasNextLong())                 System.out.println(scanner02.nextLong() + " = long");         }     } } </pre>

## Practice Your Java Level 1

```

        else if(scanner02.hasNextBigInteger())
            System.out.println(scanner02.nextBigInteger() + " = BigInteger");
        else if(scanner02.hasNextFloat()){
            String str = scanner02.next();
            float x      = Float.parseFloat(str);
            double y     = Double.parseDouble(str);

            if((x!= Float.POSITIVE_INFINITY) && (x != Float.NEGATIVE_INFINITY) && (x != Float.NaN))
                System.out.println(x + " = float");
            else if((y!=Double.POSITIVE_INFINITY) && (y != Double.NEGATIVE_INFINITY) &&
                    (y != Double.NaN))
                System.out.println(y + " = double");
            else
                System.out.println(str + " = BigDecimal");
        }
        else
            System.out.println(scanner02.next() + " = String");
    }
    scanner01.close();
    scanner02.close();
}
}

```

16. Explain your understanding of the objects `System.in` and `System.out`.
- In the simplest of terms, `System.in` and `System.out` refer to the keyboard and to the console respectively. There are methods which are applicable to each; for example, we have seen the extensive use of the methods `System.out.println` and `System.out.printf`. `System.in` has, among other methods, the method `read` which is used to read single character input from the keyboard.
- In a more detailed sense, `System.in` is an `InputStream` object and `System.out` is a `PrintStream` object. The interested reader is encouraged to study more on these classes.

### Inputting a single keypress from the keyboard

17. Write a program, which, using `System.in.read`, asks the user for a single keypress and on receiving it prints out the key received and exits.

```

import java.util.*;

public class ConsoleHandling {
    public static void main(String[] args){
        System.out.println("Enter a single character");

        try {
            c = (char)System.in.read();
            System.out.println("The character read was: " + c);
        } catch (IOException e) {
            System.out.println("Couldn't read from the keyboard successfully.");
        }
    }
}

```

18. Write a program which uses a `Scanner` object to request and receive the first character of user input.

```

import java.util.*;

public class ConsoleHandling {
    public static void main(String[] args) {
        char c01;
        System.out.print("Enter a character followed by the return key please: ");
        Scanner in = new Scanner(System.in);
        c01 = in.next().charAt(0); //take the 1st character of whatever the person enters.
        if (in != null)
            in.close(); // close the scanner when done
        System.out.printf("The character you typed in is: %c\n",c01);
    }
}

```

# Chapter 18. Console Handling III: Command Line Input

There are a significant number of programs in existence which are run directly from the command line or are invoked by other programs without any direct user input. A number of these programs require that parameters be passed to them on the command line (or by their calling program) at the time of their invocation. This chapter presents exercises which ensure understanding of how programs receive and process command line input that is passed to them on invocation.

**Note:** The exercises in this chapter must be run directly from the command line and not from within an IDE.

1.	What does the information in the brackets in the following method declaration mean?  <code>static void main(String[] args)</code>
	It means that this method is expecting to be passed an array of strings. This being the method <code>main</code> which in effect is invoked by the operating system, it means that this method is expecting that an array of type <code>String</code> may be passed to it by the operating system (or by whatever program invokes this program directly). It is acceptable that an empty array be passed to it, in which case the number of elements in the <code>String</code> array <code>args</code> will be 0.
2.	What is the type that command line input is interpreted as?  Command line input is always interpreted as a <code>String</code> . This implies that if we are expecting numerical input, we have to use the <code>Integer.parseInt</code> , <code>Long.parseLong</code> and suchlike methods to convert the strings in question to numbers.
3.	Write and test a program which can receive any number of command line arguments. Print out the number of command line arguments received and also print each argument value out in the following format:  <code>argument[&lt;x&gt;] = &lt;argument value&gt;</code>  <code>public class ConsoleHandling {     public static void main(String[] args){         System.out.printf("Number of args=%d\n", args.length);         for (int i = 0; i &lt; args.length; i++)             System.out.printf("argument[%d] = %s\n", i, args[i]);     } }</code>
4.	Write a program which receives the first name and last name of a person, in that order, on the command line. Print these out to the console in the format <code>&lt;last name&gt;, &lt;first name&gt;</code> . If anything other than two arguments are entered, output the statement “Please pass exactly 2 arguments.” and then exit the program. Remember to run the program from command line.  <code>public class ConsoleHandling {     public static void main(String[] args){         if(args.length !=2){             System.out.println("Please pass exactly 2 arguments.");             System.exit(0);         }         System.out.printf("%s, %s", args[1], args[0]);     } }</code>
5.	Write a program which takes an unlimited number of integers from the command line and does the following: <ol style="list-style-type: none"><li>1. Adds them up and prints out the summation of the integers presented on the command line. If any non-integers are presented, it ignores them.</li><li>2. Prints out how many integers were passed on the command line.</li><li>3. Prints out how many arguments were presented on the command line.</li></ol> <i>Hint: Long.parseLong</i>

## Practice Your Java Level 1

```
public class ConsoleHandling {  
    public static void main(String[] args){  
        long sum = 0;  
        int numOfIntegers = 0;  
  
        if(args.length == 0 ){  
            System.out.println("No arguments passed. Exiting");  
            System.exit(0);  
        }  
        for (int i = 0; i < args.length; i++){  
            long potentialInteger;  
            try{  
                potentialInteger = Long.parseLong(args[i]);  
                sum+=potentialInteger;  
                numOfIntegers++;  
            }  
            catch(NumberFormatException e){  
                //Ignore the non-numbers  
            }  
        }  
        System.out.printf("Total = %d\n", sum);  
        System.out.printf("Number of integers = %d\n", numOfIntegers);  
        System.out.printf("Number of args      = %d\n", args.length);  
    }  
}
```

# Chapter 19. The `char` primitive & the `Character` Wrapper Class

Java presents the ability to manipulate single characters, whether they are of type `char`, or individual characters in a `String`. This chapter presents exercises that pertain to various aspects of the `char` type and its related wrapper class `Character`. Also exercises on the relationship between the `char` type and integers and individual elements of `String` objects are presented.

The methods which operate on the `char` primitive are found in its corresponding wrapper class `Character`; this interrelationship is woven throughout the exercises in this chapter.

1.	What is Unicode?
	Unicode is a convention by which characters from various languages are assigned a particular positive numerical value. The Unicode character range exceeds one million, however, most of the characters that are in use are in the range <code>0000(hex)</code> to <code>FFFF(hex)</code> (0 to 65535 in number base 10), a range which can be represented in 2 bytes, which is the size of the <code>char</code> primitive.
2.	What is the <code>char</code> primitive type?
	The <code>char</code> type is a Java type that is used to represent Unicode characters.
3.	Explain briefly the concept of Unicode encoding schemes.
	It was stated earlier that in Unicode, characters have been assigned a numerical value, ranging from a value of 0, to a value that is greater than 1,000,000. The concept of a Unicode encoding scheme refers to the different ways in which Unicode values are represented; in Unicode there are a number of different encoding schemes, each with a different emphasis. The schemes UTF-8, UTF-16 and UTF-32 exist; each one encodes the characters in a different way. For example, UTF-32 encodes the characters in a fixed 4 bytes per character (which takes up a lot of space for characters which don't really need up to 4 bytes for representation), UTF-8 encodes the characters in either of 1,2,3 or 4 bytes per character (which while it saves space for characters at the lower end of the Unicode space, it adds programming complexity). UTF-16 encodes characters in either 2 or 4 bytes.
4.	What is the default Unicode encoding used for characters in Java?
	The particular Unicode encoding known as UTF-16. It uses either 2 or 4 bytes to represent characters.
5.	How many bytes are used to represent a <code>char</code> in Java?
	2 bytes, which represents a range of 65536 numbers. Given that the <code>char</code> type is represented by positive integer values, the range of the numbers that these 2 bytes represent is 0 to 65535. Note however though, that there are certain Unicode characters that require 4 bytes for representation; these characters are represented in Java by 2 chars.
6.	What numerical type would correspond most directly to the numerical range that a <code>char</code> will fit into?
	An unsigned short integer, which has a range of 0...65535 ( <i>Java doesn't have an unsigned short type though</i> ).
7.	Write a line of code which will initialize the <code>char</code> variable <code>firstChar</code> to the value 'x'.
	<code>char firstChar = 'x';</code>
8.	Run the following program and explain its output.
	<pre>public class PracticeYourJava {     public static void main(String[] args) {         char char01 = (char)100;         System.out.println(char01);     } }</pre>
	<b>Output:</b> The program will output the following: d The reason why we can cast an integer value to a <code>char</code> is because every character has been mapped to a numerical equivalent in Unicode and its various encoding schemes. In this example, we see that the value 100

## Practice Your Java Level 1

	<p>maps to the character d.</p> <p>The reader is urged to review the Unicode UTF-16 character encodings for the first 127 characters and to compare these values to the first 127 values of an ASCII table.</p>
9.	<p>Run the following program and explain its output.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         char char01 = 'A';         int num = char01; //built-in implicit conversion from char to integer         System.out.println(num);     } }</pre> <p><b>Output:</b> The code will output the following value: 65</p> <p>The reason for this is because the Unicode UTF-16 mapping of the character 'A' is the numeric value 65. This exercise and the preceding one are here to show the relationship between the <code>char</code> type and integer values, moving in-between them using the required cast.</p>
10.	<p>Using a <code>for</code> loop, print out the equivalent characters of the number range 65 to 150.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         for (int j = 65; j &lt;= 150; j++)             System.out.println(j + " is equivalent to " + (char)j);     } }</pre> <p><b>Notes:</b> You will note that above a certain integer value the console window will show a question mark for each character. This does not mean that each of those values necessarily represents the question mark, but rather that the console cannot display those characters (they can be displayed properly in a GUI) due to the way in which the console displays items.</p>
11.	<p>Given the following array of <code>char</code> (see below), print out the numerical value of each <code>char</code> in the array.</p> <pre>char[] charArray01 = new char[]{'?', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', 'A', 'a'};</pre> <pre>public class PracticeYourJava {     public static void main(String[] args) {         char[] charArray01 = new char[] { '?', '1', '2', '3', '4', '5', '6', '7', '8',  '9', '0', 'A', 'a' };         for (int j = 0; j &lt; charArray01.length; j++)             System.out.println(charArray01[j] + " is equivalent to " + (int)charArray01[j]);     } }</pre>
12.	<p><i>Char &amp; Hexadecimal</i></p> <p>Write a program which assigns the hexadecimal value 2A to a <code>char</code> variable. Print the variable out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         char c01 = '\u002A'; // the 'u' indicating Unicode         System.out.println(c01);     } }</pre>
13.	<p>Run and explain what the code below is doing. Please note that this exercise is pertinent to the next few exercises.</p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         char c01;         System.out.print("Enter a character followed by the return key please: ");         Scanner in = new Scanner(System.in);         c01 = in.next().charAt(0); //take the 1st character of whatever the person enters.         if (in != null)             in.close(); // close the scanner when done         System.out.printf("The character you typed in is: %c\n", c01);     } }</pre>

```

    }
}
```

**Summary**

The summary is that this code is being used to read the first character input at the keyboard and put it into the `char c01`.

**Details**

The program acquired a reference to a `Scanner` object which receives line input from the input stream object denoted by `System.in`; this happens to be, by default, the keyboard. The object `System.in` receives input until the Enter key is pressed.

The line input having been received and stored in the `Scanner` object, our code, requests the input line as a `String` in question from the `Scanner` object via the command `in.next()`, however, really what we want in this case is the first character in that input string and so we request the first character at the same time that we request the `String` as follows: `in.next().charAt(0)`.

We can then use that character as we want.

Of course we must remember to close that input stream, by using its `close` method.

14. Write a program which requests a single character from a user. Analyze the character and state whether it is a letter or not.

*Hint: Use the `Character.isLetter` method.*

```

import java.util.*;
public class PracticeYourJava {
    public static void main(String[] args) {
        char c01;

        System.out.print("Enter a character followed by the return key please: ");
        Scanner in = new Scanner(System.in);
        c01 = in.next().charAt(0); //take the 1st character.
        boolean boolIsLetter = Character.isLetter(c01);
        if (in!=null) in.close();
        if(boolIsLetter==true)
            System.out.println("The character you entered is a letter\n");
        else
            System.out.println("The character you entered is not a letter\n");
    }
}
```

15. Have the user enter an alphabetic character. Determine and print out the case of the character and then print out the character in the opposite case to which it was originally entered. For example if an uppercase character is entered, print out the following: *"It is uppercase. Its lowercase equivalent is <lower case equivalent>."* If it is not an alphabetic character state so.

*Hint: `Character.isLetter`, `Character.isUpperCase`, `Character.toLowerCase`*

```

import java.util.Scanner;
public class PracticeYourJava {
    public static void main(String[] args) {
        char c01;
        char oppositeCase;

        System.out.print("Enter a character followed by the return key please: ");
        Scanner in = new Scanner(System.in);
        c01 = in.next().charAt(0); //take the 1st character.
        boolean boolIsLetter = Character.isLetter(c01);
        if (in!=null) in.close();

        if(!Character.isLetter(c01)) {
            System.out.println("The character you entered is not a letter. Exiting.\n");
            System.exit(0);
        }
        if(Character.isUpperCase(c01)) {
            oppositeCase = Character.toLowerCase(c01);
            System.out.printf("It is uppercase. The opposite case of %c = %c\n",c01,oppositeCase);
        }
    }
}
```

## *Practice Your Java Level 1*

	<pre>         }     else{         //It is lower case         oppositeCase = Character.toUpperCase(c01);         System.out.printf("It is lowercase. The opposite case of %c = %c\n",c01,oppositeCase);     } } </pre>
16.	<p>Write a program which requests a character from a user. Analyze it and state to the user whether or not it is a numerical character.</p> <p><i>Hint: Character.isDigit</i></p> <pre> import java.util.Scanner;  public class PracticeYourJava {     public static void main(String[] args) {         char c01;          System.out.print("Enter a character followed by the return key please: ");         Scanner in = new Scanner(System.in);         c01 = in.next().charAt(0); //take the 1st character.         boolean boolIsNumeric = Character.isDigit(c01);         if (in!=null) in.close();         if(boolIsNumeric == true)             System.out.println("The character you entered is a numeric character.\n");         else             System.out.println("The character you entered is not a numeric character.\n");     } } </pre>
17.	<p>Write a program which requests a character from a user. Analyze the character and inform the user as to whether or not it is an alphanumeric character.</p> <p><i>Hint: Character.isAlphabetic, Character.isDigit</i></p> <pre> import java.util.Scanner;  public class PracticeYourJava {     public static void main(String[] args) {         char c01;          System.out.print("Enter a character followed by the return key please: ");         Scanner in = new Scanner(System.in);         c01 = in.next().charAt(0); //take the 1st character.         boolean boolIsAlphaNumeric = (Character.isAlphabetic(c01)    Character.isDigit(c01));         if (in!=null) in.close();         if(boolIsAlphaNumeric == true)             System.out.println("The character you entered is an alphanumeric character.\n");         else             System.out.println("The character you entered is not an alphanumeric character.\n");     } } </pre>
18.	<p>Write a program which requests a single character from a user. Analyze the character and inform the user as to whether or not it is a whitespace character.</p> <p><i>Hint: Character.isWhiteSpace</i></p> <pre> import java.util.Scanner;  public class PracticeYourJava {     public static void main(String[] args) {         char c01;          System.out.print("Please enter a character: ");         Scanner in = new Scanner(System.in);         char c01 = in.next().charAt(0); //take the 1st character.          if (reader != null) reader.close();         boolean boolIsWhitespace = Character.isWhitespace(c01);     } } </pre>

	<pre> if(boolIsWhitespace == true)     System.out.println("The character you entered is a whitespace character\n"); else     System.out.println("The character you entered is not a whitespace character\n"); } } </pre>
19.	<p>I have two <code>char</code> variables, <code>c01</code> and <code>c02</code> (for testing purposes hardcode values for them). Write a program which determines if they contain the same value and print out your findings as to whether or not they are equal.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         char c01 = 'x';         char c02 = 'x';          if (c01 == c02)             System.out.println("The characters are the same!");         else             System.out.println("The characters are different!");     } }  OR  public class PracticeYourJava {     public static void main(String[] args) {         char c01 = 'x';         char c02 = 'x';          if (Character.compare(c01, c02))             System.out.println("The characters are the same!");         else             System.out.println("The characters are different!");     } } </pre>
<b>String/Char relationship</b>	
20.	<p>What is the relationship between the <code>char</code> primitive and the type <code>String</code>?</p> <p>A <code>String</code> is a datatype which might <i>have the appearance</i> of being a sequence/array of <code>char</code>, however it is not so. We restate the above as follows: Despite the similarity in the content, a <code>char[]</code> is not equivalent to a <code>String</code>. There are however methods which convert between both types and also a method that facilitates the extraction of individual characters in a <code>String</code> as the type <code>char</code>.</p>
21.	<p>State what the difference between "<code>x</code>" and '<code>x</code>' is.</p> <p>"<code>x</code>" refers to a <code>String</code> object. '<code>x</code>' refers to a <code>char</code>.</p>
22.	<p>Write a program which converts the <code>char 'x'</code> to a <code>String</code> which you should then print out.</p> <p><i>Hint: Character.toString or String.valueOf</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         char c01 = 'x';         String s01 = Character.toString(c01);         System.out.printf("%s\n", s01);     } }  OR  public class PracticeYourJava {     public static void main(String[] args) {         char c01 = 'x';         String s01 = String.valueOf(c01);         System.out.printf("%s\n", s01);     } } </pre>
23.	<p>Write a program which converts the string "<code>Xavier</code>" to an array of type <code>char</code>.</p> <p>Loop through the resulting <code>char</code> array and write out each character one after the other.</p> <p><i>Hint: String.toCharArray</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01;         String s1 = "Xavier";         cArray01 = s1.toCharArray();         for (int j = 0; j &lt; cArray01.length; j++)             System.out.print(cArray01[j]);     } } </pre>

## Practice Your Java Level 1

		}
24.	I have the following String: s1="Xavier". Copy the 2 <sup>nd</sup> character in s1 to the char variable c01 and print c01 out to the console. <i>Hint: String.charAt</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01;         String s1 = "Xavier";         c01 = s1.charAt(1); // The 2nd character.         // Accessing a String as if it is an array of char.         System.out.println(c01);     } }</pre>
25.	Convert the following char array to a String: char[] cArray01 = {'A','B','C','D','E','F','G'};	<pre>public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01 = {'A','B','C','D','E','F','G'};         String s1 = new String(cArray01);         System.out.println(s1);     } }</pre> <p><b>OR</b></p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01 = {'A','B','C','D','E','F','G'};         String s1 = String.valueOf(cArray01);         System.out.println(s1);     } }</pre>
26.	Convert the 2 <sup>nd</sup> up to the last character in the following char array to a single String:	<pre>char[] cArray01 = {'A','B','C','D','E','F','G'};</pre> <pre>public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01 = {'A','B','C','D','E','F','G'};         String s1 = String.valueOf(cArray01, 1, cArray01.length - 1);         System.out.println(s1);     } }</pre>
27.	I have a String variable which contains the following: "Hello how are you?" Write a program which reverses this string. Print out the resulting string. <i>Hint: One way in which this can be done is to first convert the String to an array of char. Then loop through the array of char backwards, adding each character to the new String that represents the reversed String (you add a char to a String by first converting the char to a String using the ToString method of char).</i> <i>Note: Another solution is to use the Array.Reverse method.</i>	<pre>public class PracticeYourJava {     public static void main(String[] args) {         String s1 = new String("Hello how are you?");         char[] cArray01 = s1.toCharArray();         String reversed="";         for (int index = cArray01.length - 1; index &gt;= 0; index--){             reversed += Character.toString(cArray01[index]);         }         //Not the most efficient code, but it suffices         System.out.println(reversed);     } }</pre>

# Chapter 20. Random Numbers

The ability to produce random numbers is an important feature in such scenarios as the generation of random data for statistical purposes, for adding randomness to computer games and particularly for cryptographic applications. This chapter presents exercises that ensure your basic ability to perform random number generation in Java.

The key class on which exercises are presented in this chapter is the class `Random`.

1.	<p><i>Initialization of the random number generator</i></p> <p>What is the difference between the following two initialization methods for the random number generator?</p> <ol style="list-style-type: none"><li>1. Using a user supplied seed value</li><li>2. Using the built-in time-based seed for the generator</li></ol>
	<p>In the first case, you the user supply the seed value. However if you use the same seed value again the random number generator will simply generate the same random value.</p> <p>In the second case, you are entrusting randomness to the built-in seed generator, which indeed uses the time-honored method of using the system clock as one of the inputs in generating the seed.</p>
2.	<p>Letting the JVM determine the seed, instantiate a random number generator object and print out the first 10 <code>int</code> values that it returns. Use the method <code>Random.nextInt()</code>.</p>
	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {          Random randomGenerator = new Random();         int nextRandomValue;         for(int count=0; count &lt; 10; count++){             nextRandomValue = randomGenerator.nextInt();             System.out.println(nextRandomValue);         }     } }</pre>
	<p><b>Notes:</b> Observe that the returned values can be both negative and positive and span the range of an <code>int</code>.</p>
3.	<p>Letting the JVM determine the seed, instantiate a random number generator object and print out the first 10 <code>long</code> values that it returns. Use the method <code>Random.nextLong()</code>.</p>
	<pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {          Random randomGenerator = new Random();         long nextRandomValue;         for(int count=0; count &lt; 10; count++){             nextRandomValue = randomGenerator.nextLong();             System.out.println(nextRandomValue);         }     } }</pre>
	<p><b>Notes:</b> Observe that the returned values can be both negative and positive and span the range of a <code>long</code>.</p>
4.	<p>What is the range of values that the method <code>Random.nextDouble()</code> spans?</p>
	<p>The range of values is <code>0</code> to <code>1.0</code>.</p>
5.	<p>Letting the JVM determine the seed, instantiate a random number generator and print out the first 10 <code>double</code> values that it returns. Use the method <code>Random.nextDouble()</code>.</p>
	<pre>import java.util.*;  public class PracticeYourJava {</pre>

## Practice Your Java Level 1

	<pre>public static void main(String[] args) {     Random randomGenerator = new Random();     double nextRandomValue;     for(int count=0; count &lt; 10; count++){         nextRandomValue = randomGenerator.nextDouble();         System.out.println(nextRandomValue);     } }</pre>
6.	<p>Using an <code>int</code> value of your choosing, seed the random number generator with it and then print out the first 10 random numbers of type <code>int</code> returned by the random generator. Take record of the numbers generated. Then, rerun the exercise and take note of the numbers generated. Comment on the results.</p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         Random randomGenerator = new Random(1234312);         int nextRandomValue;         for(int count=0;count&lt;10;count++){             nextRandomValue = randomGenerator.nextInt();             System.out.println(nextRandomValue);         }     } }</pre> <p><b>Comment:</b> The output is the same, since the seed is the same. There are times though, that we do want such “controlled randomness” and so supply the seed ourselves. For example, we might use such a seed when we want to administer an online exam and want the questions on a given day to be put in the same “random” order to the individual exam takers taking it that day (so that they cannot share the question order with others taking the exam on subsequent days).</p>
7.	<p><i>Limiting the output range</i>  Generate 10 random <code>int</code> values between the value of 0 and 20.  <i>Hint: overloaded Random.nextInt(&lt;value&gt;) which generates only positive numbers</i></p> <p>When an <code>int</code> upper bound value is specified to the random number generator, that upper bound value is actually not included as a potential number, but rather <i>upper bound - 1</i> is really the upper limit of numbers that will be generated! Therefore if the intent is to generate numbers between 0 and z, z included, then really we have to specify to the generator function that the upper bound = z + 1.</p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int upperBound = 20;         Random randomGenerator = new Random();         int nextRandomValue;         for(int count=0; count &lt; 10; count++){             nextRandomValue = randomGenerator.nextInt(upperBound + 1);             System.out.println(nextRandomValue);         }     } }</pre>
8.	<p>Generate 10 random <code>int</code> values between the value of <u>1</u> and 20.</p> <p>The random number generator supports an upper bound being set (see the description of how it is set in the solution to the preceding exercise), however the generator always has a lower bound of 0, therefore, we have to compensate for this in our code.</p> <p>One means of compensation is to add the value of our intended lower bound —in this exercise the value 1— to the generated number; that guarantees a lower bound of 1.</p> <p>Also, we have stated that in this exercise we want an upper bound of 20; therefore to effect this, we pass to the</p>

	<p>generator the stated upper bound of 20 instead of <math>20 + 1</math> as in the preceding exercise, knowing the following two facts: (1) the generator will generate a maximum value of <math>20 - 1 = 19</math> and (2) we are adding 1 to whatever the generator outputs and that will give us a range of 1 to <math>(19+1) = 20</math>. We restate the above as follows: The generator will generate 0 to 19. Adding 1 to the generated value will shift this range to 1 to 20.</p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         int upperBound = 20;         Random randomGenerator = new Random();         int nextRandomValue;         for(int count=0; count &lt; 10; count++){             nextRandomValue = 1 + randomGenerator.nextInt(upperBound);             System.out.println(nextRandomValue);         }     } }</pre>
9.	<p>Write a program which will generate 10 random <code>double</code> numbers, Gaussian distributed, with mean 0 and standard deviation 1.0.</p> <p><i>Hint: randomGenerator.nextGaussian()</i></p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         Random randomGenerator = new Random();         double nextRandomValue;         for(int count=0; count &lt; 10; count++){             nextRandomValue = randomGenerator.nextGaussian();             System.out.println(nextRandomValue);         }     } }</pre>
E1	Study the random number generation class <code>SecureRandom</code> which is a more suitable generator for cryptography purposes.



# Chapter 21. StringBuffer & StringBuilder

This chapter presents exercises that pertain to the `StringBuffer` and `StringBuilder` classes, which are mutable counterparts of the `String` class. The programming exercises in this chapter will focus on the `StringBuffer` class, which while fully equivalent in outward functionality to the older `StringBuilder` class is a safer class to use.

1.	Describe the <code>StringBuffer</code> class and how it relates to and differs from the <code>String</code> class.  The <code>StringBuffer</code> class is a class that supports the same content as the <code>String</code> class, however objects of class <code>StringBuffer</code> are mutable, as opposed to the <code>String</code> class whose objects are immutable (the immutability of <code>String</code> objects has been discussed in Chapter 5. <i>String: Basic Handling</i> ). Due to this mutability, the <code>StringBuffer</code> class manipulates string data faster than the <code>String</code> class. Also, because <code>StringBuffer</code> objects are mutable, when a <code>StringBuffer</code> object is passed from one method to another, the receiving method can change the contents of the <code>StringBuffer</code> object and this will reflect in the calling method; this cannot happen with <code>String</code> objects. We will see examples of this in Chapter 23. It should be noted that despite the speed improvement, it is still very acceptable to use the <code>String</code> class for string operations; the speed advantage of the <code>StringBuffer</code> class is made more apparent in programs which need to do a very high amount of string manipulation.
2.	When a <code>StringBuffer</code> object is instantiated, what is its default initial capacity?  The initial capacity is 16 characters.
3.	What happens if you attempt to put into a <code>StringBuffer</code> object more characters than its current capacity?  The <code>StringBuffer</code> objects' capacity is automatically resized by the system to at least the necessary capacity.
4.	Write a program which will instantiate a <code>StringBuffer</code> object named <code>sb01</code> . Print out its capacity as well. <i>Hint: &lt;StringBuffer&gt;.capacity</i> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer();         System.out.println("The capacity = " + sb01.capacity());     } }</pre>
5.	Write a program which will instantiate a <code>StringBuffer</code> object <code>sb02</code> with an initial capacity for 1000 characters. <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb02 = new StringBuffer(1000);     } }</pre>
6.	Given the <code>StringBuffer</code> object declared above, enhance the solution to the preceding exercise to ensure that the <code>StringBuffer</code> is set to a minimum capacity of 5000 <i>after</i> its initial creation with a size of 1000. Print out the new capacity to confirm that it indeed has been modified. <i>Hint: StringBuffer.ensureCapacity</i> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb02 = new StringBuffer(1000);         sb02.ensureCapacity(5000);         System.out.println("Capacity = " + sb02.capacity());     } }</pre>
7.	Explain the use of the <code>trimToSize</code> method of class <code>StringBuffer</code> .  There are times when we want the size of an oversized <code>StringBuffer</code> to exactly match the length of the contents. The method <code>trimToSize</code> effects this.
8.	What value does the <code>length</code> method of the <code>StringBuffer</code> class represent?

## Practice Your Java Level 1

	The <code>length</code> method represents the length of whatever string/character sequence is currently assigned to the <code>StringBuffer</code> object in question, <u>not</u> the capacity of the <code>StringBuffer</code> .
9.	<p>Print out the length of the <code>StringBuffer</code> object in question 5 above.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb02 = new StringBuffer(1000);         System.out.println("The length = " + sb02.length());     } }</pre> <p><b>Output:</b> 0, as expected, because no string/character sequence has been assigned yet to the <code>StringBuffer</code>.</p>
10.	<p>Given the <code>String s01 = "Mary had a little lamb"</code>, initialize a <code>StringBuffer</code> object <code>sb01</code> with the <code>String s01</code>. Print out the length of <code>s01</code> and <code>sb01</code>. Also, print out the capacity of <code>sb01</code>.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         String s01 = "Mary had a little lamb";         StringBuffer sb01 = new StringBuffer(s01);         System.out.println("The length of the String      = " + s01.length());         System.out.println("The length of the StringBuffer = " + sb01.length());         System.out.println("THe capacity of the StringBuffer = " + sb01.capacity());     } }</pre> <p><b>Comments:</b> The length of the <code>String</code> and <code>StringBuffer</code> objects match.</p>
11.	<p>I have the following character array: <code>char[] cArray01 = new char[]{'M','a','r','y'}</code>; Initialize a <code>StringBuffer</code> object with this <code>char[]</code>. Print out the contents of the <code>StringBuffer</code> object.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01 = new char[]{'M','a','r','y'};         // Have to convert it to a String first because a StringBuffer cannot be initialized         // directly with a char[]         String s01 = new String(cArray01);         StringBuffer sb01 = new StringBuffer(s01);         System.out.println(sb01);     } }</pre> <p><i>Or, interestingly,</i></p> <pre>import java.util.*;  public class PracticeYourJava {     public static void main(String[] args) {         char[] cArray01 = new char[]{'M','a','r','y'};         StringBuffer sb01 = new StringBuffer(); // get an empty StringBuffer         sb01.append(cArray01); //you can append a char[] to an existing StringBuffer object         System.out.println(sb01);     } }</pre>
12.	<p>I have a <code>StringBuffer</code> object with the contents <code>"Mary had a little lamb"</code>. Copy the contents of this object into a <code>String</code> object <code>s01</code>.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         String s01 = sb01.toString();     } }</pre>
13.	<p>I have a <code>StringBuffer</code> object with the contents <code>"Mary had a little lamb"</code>. Append the contents of a <code>String s01</code> that contains <code>"\nlittle lamb\nlittle lam"</code> (<i>the "b" is missing on purpose</i>) to the <code>StringBuffer</code> object. Print out</p>

	<p>the resulting <code>StringBuffer</code> object.</p> <p><i>Hint: method <code>append</code></i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         String s01 = "\nlittle lamb\nlittle lam";         sb01.append(s01);         System.out.println(sb01);     } }</pre>
14.	<p>Extend the solution to the preceding exercise to append the character 'b' to the end of the <code>StringBuffer</code> object in the previous question. Print the resulting <code>StringBuffer</code> out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         String s01 = "\nlittle lamb\nlittle lam";         sb01.append(s01);          char missingCharacter = 'b';         sb01.append(missingCharacter);         System.out.println(sb01);     } }</pre>
15.	<p>Put the following values, comma delimited, into a <code>StringBuffer</code> object <code>sb01</code>:  <code>double d01=7.5943e12, float f01=1.6E14f, int int01=823</code>. Print <code>sb01</code> out and compare its contents to what was input.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         double d01 = 7.5943e12; float f01=1.6E14f; int int01=823;         StringBuffer sb01 = new StringBuffer();         sb01.append(d01); sb01.append(',');         sb01.append(f01); sb01.append(',');         sb01.append(int01);         System.out.println(sb01);     } }</pre> <p><b>Comments:</b> The purpose of this exercise was for you to see that the string representation of numerical types can be put into <code>StringBuffer</code> objects.</p>
16.	<p>I have the following two <code>StringBuffer</code> objects:</p> <pre>StringBuffer sb01 = new StringBuffer("Mary had a little lamb"); and StringBuffer sb02 = new StringBuffer("\nlittle lamb\nlittle lamb");</pre> <p>Concatenate the contents of <code>sb02</code> to <code>sb01</code>. Print <code>sb01</code> out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         StringBuffer sb02 = new StringBuffer("\nlittle lamb\nlittle lamb");         sb01.append(sb02);         System.out.println(sb01);     } }</pre>
17.	<p>Copy the first four characters of the resulting <code>StringBuffer</code> <code>sb01</code> in the preceding solution into a <code>String</code> <code>s01</code>. Print <code>s01</code> out.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb\nlittle lamb\nlittle lamb");</pre>

## Practice Your Java Level 1

	<pre>         String s01 = sb01.substring(0,4);         System.out.println(s01);     } } </pre>
18.	<p>What is the position of the first occurrence of the word “little” in the <code>StringBuffer sb01</code> in the preceding exercise?</p> <p><i>Hint: <code>StringBuffer.indexOf</code></i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         String searchWord = "little";         StringBuffer sb01 = new StringBuffer("Mary had a little lamb\nlittle lamb\nlittle lamb");         int position = sb01.indexOf(searchWord);         System.out.printf("The position of %s is %d\n", searchWord, position);     } } </pre>
19.	<p>What is the position of the 2<sup>nd</sup> occurrence of the word “little” in the <code>StringBuffer sb01</code> in the preceding exercise?</p> <p><i>Hint: <code>StringBuffer.indexOf(String str, int fromIndex)</code></i></p> <p><i>Determine the position of the preceding occurrences, then start searching from there.</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         String searchWord = "little";         StringBuffer sb01 = new StringBuffer("Mary had a little lamb\nlittle lamb\nlittle lamb");         int position = sb01.indexOf(searchWord); //that's the first occurrence         position = sb01.indexOf(searchWord, position+searchWord.length()); //we started searching after the first occurrence         System.out.printf("The position is %d\n", searchWord, position);     } } </pre>
20.	<p>What is the position of the last occurrence of the word “little” in the <code>StringBuffer sb01</code> in the preceding exercise?</p> <p><i>Hint: <code>StringBuffer.lastIndexOf</code></i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         String searchWord = "little";         StringBuffer sb01 = new StringBuffer("Mary had a little lamb\nlittle lamb\nlittle lamb");         int position = sb01.lastIndexOf(searchWord);         System.out.printf("The last index of %s is %d\n", searchWord, position);     } } </pre>
21.	<p>What is the position of the <i>n-th</i> occurrence of the word “little” in the <code>StringBuffer sb01</code> in the preceding exercise?</p> <p><i>Hint: Similar to the preceding exercise, however use a loop</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         String searchWord = "little";         StringBuffer sb01 = new StringBuffer("Mary had a little lamb\nlittle lamb\nlittle lamb");         int occurrence = 3; // We'll just choose the 3rd one here. You can change it.         int occurrenceCounter=0;         int found=0;         int position=0;         do{             occurrenceCounter++;             found = sb01.indexOf(searchWord, position);             if((found &gt;= 0) &amp;&amp; (occurrenceCounter &lt; occurrence))                 position=found + searchWord.length();         }while((occurrenceCounter &lt; occurrence) &amp;&amp; (found &gt;= 0));     } } </pre>

	<pre>         if(found &gt;= 0)             System.out.printf("The position is %d\n", position);         else             System.out.printf("There are not up to %d occurrences of %s here\n", occurrence, searchWord);     } } </pre>
22.	<p>Given the same <code>StringBuffer</code> object as in the preceding exercise, modify the word “Mary” at the beginning of the <code>StringBuffer</code> object to “John &amp; Mary”. Print out the resulting <code>StringBuffer</code>.</p> <p><i>Hint: insert</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb\nlittle lamb\nlittle lamb");         sb01.insert(0, "John &amp; ");         System.out.println(sb01);     } } </pre>
23.	<p>Replace the string “John &amp; Mary” at the beginning of the resultant <code>StringBuffer</code> <code>sb01</code> of the preceding exercise with the string “David”. </p> <p><i>Hint: StringBuffer.replace</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          String oldStr = "John &amp; Mary";         String newStr = "David";         StringBuffer sb01 = new StringBuffer("John &amp; Mary had a little lamb\nlittle lamb\nlittle lamb");         sb01.replace(0, oldStr.length()-1, newStr);         System.out.println(sb01);     } } </pre>
24.	<p><i>How to reverse a String</i></p> <p>I have the following <code>String</code> object: <code>String original = "Hello how are you?"</code>; Using the <code>reverse</code> method of the class <code>StringBuffer</code>, reverse this <code>String</code> and print it out.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          String original = "Hello how are you?";         StringBuffer sb01 = new StringBuffer(original);         sb01.reverse();         String reversed = sb01.toString();         System.out.println(reversed);     } } </pre>
25.	<p><i>StringBuffer to Char array</i></p> <p>Given a <code>StringBuffer</code> object <code>sb01</code> with the value “<i>Mary had a little lamb</i>”, copy the first 4 letters in it to the first 4 elements of a <code>char[]</code> named <code>cArray01</code>. Print <code>cArray01</code> out.</p> <p><i>Hint: StringBuffer.getChars</i></p> <pre> public class PracticeYourJava {     public static void main(String[] args) {          StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         char[] cArray01 = new char[4];         sb01.getChars(0, 4, cArray01, 0);         System.out.println(cArray01);     } } </pre>
26.	<p>Remove the substring “had a little lamb” from the <code>StringBuffer</code> object in the preceding exercise.</p> <p><i>Hint: find the position of the desired substring and then apply the method StringBuffer.delete</i></p>

## Practice Your Java Level 1

	<pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         String strToRemove = " had a little lamb";         //First find its position in the String         int position = sb01.indexOf(strToRemove);         if(position &gt;=0)             sb01.delete(position, position + strToRemove.length());         System.out.println(sb01);     } }</pre>
27.	<p>Given a <code>StringBuffer</code> object <code>sb01</code> with the value “<i>Mary had a little lamb</i>”, print out the 7<sup>th</sup> character therein.  <i>Hint:</i> <code>StringBuffer.charAt</code></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         char char01 = sb01.charAt(6);         System.out.println("The character is " + char01);     } }</pre>
28.	<p>Given a <code>StringBuffer</code> object <code>sb01</code> with the value “<i>Mary had a little lamb</i>”, empty <code>sb01</code> of its contents.  <i>Hint:</i> <code>StringBuffer.setLength</code></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuffer sb01 = new StringBuffer("Mary had a little lamb");         sb01.setLength(0);         System.out.println(sb01);     } }</pre>
29.	<p><i>StringBuffer vs String performance</i></p> <p>We want to assess the performance of <code>String</code> vs <code>StringBuffer</code>.</p> <p>To this end, write a program which does the following:</p> <ol style="list-style-type: none"> <li>Creates an object <code>sbTest</code> of class <code>StringBuffer</code>. Using a <code>for</code> loop, the program appends the <code>String</code> “A” to it 100,000 times.</li> <li>Using <code>Instant</code> and <code>Duration</code> objects, measures how long it takes for the code to run and prints this time out to the console.</li> <li>Creates a <code>String sTest</code> and appends the <code>String</code> “A” to it 100,000 times using a <code>for</code> loop.</li> <li>Measures the running time as with the <code>StringBuilder</code> code and prints out the time out to the console.</li> </ol> <p>Note the difference in times.</p> <p>Next: increase the loop counter to 250,000 and note the difference in time again.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         //measuring durations, using Instant &amp; Duration         System.out.println("\nStart: Testing StringBuffer concatenation time");         StringBuffer sbTest = new StringBuffer("A");         Instant startInstant = Instant.now();         long numElements = 100_000L;         for(long counter=0L; counter&lt; numElements; counter++)             sbTest.append("A");         Instant finishInstant = Instant.now();         Duration duration01 = Duration.between(startInstant,finishInstant);         System.out.println("Time (StringBuffer) " + numElements + " characters = " + duration01);         String a="A";         startInstant = Instant.now();         for(long counter=0L; counter &lt; numElements; counter++)     } }</pre>

```

        a+="A";
        finishInstant = Instant.now();
        duration01 = Duration.between(startInstant,finishInstant);
        System.out.println("Time (String) " + numElements + " characters = " + duration01);
    }
}

```

**Notes:** It can be seen clearly that the `StringBuffer` concatenates far faster than the `String`. In scenarios where a limited amount of string manipulation is done, a `String` is an acceptable variable choice. (One thing though; if the object reference of the object in which you are storing the string data is of importance to you and you want it constant throughout the program, then use a `StringBuffer` object).

30. In general terms what are the similarities and differences between the `StringBuffer` class and the `StringBuilder` class? Which one is preferred and why?
- The `StringBuffer` class and `StringBuilder` class are essentially the same in functionality, however the `StringBuffer` class is a more modern version of `StringBuilder` in its internal functioning. In particular `StringBuffer` provides better support for threading. As a result, `StringBuffer` is the preferred class to use of the two. As can be seen in the exercises in this chapter, we used the more modern one, `StringBuffer` for our exercises.



# Chapter 22. Classes I

Java is an object-oriented language and thus the concept of classes and objects is central to it. This chapter is a very important one as it presents exercises which cover the concepts of classes (top-level classes in this chapter) and types thereof, class instances (objects), class/object member accessibility, constructors, initializers, inheritance, polymorphism and other key related topics.

Please note that the exercises in this chapter in particular should be completed in sequential order, as often through this chapter subsequent exercises build on preceding ones.

Given the number of topics and sub-topics covered in this chapter, a brief summary of the topics covered is provided after each section in this chapter.

The reader should note that this is the first of two chapters in this book that address the topic of classes.

1.	What is a class in Java?
	A class is a data structure that can combine Java members (i.e. constants, fields, methods, constructors, initializers, other classes as well as implemented interfaces) in a single unit. A class is a “template” for dynamically created <b>instances</b> of the class; these instances are called <b>objects</b> . Classes support the concept of <b>inheritance</b> , a mechanism whereby <b>derived classes</b> obtain the functionality of the class they inherit from (known as a “superclass” of the class which is doing the inheriting) and can enhance the inherited functionality as desired.
2.	In Java, what is a package?
	Simply put, a package is a grouping of types (usually any of classes, enumerations and interfaces). You the programmer determine the types that you want to group together in a package (See exercise 2 in Chapter 1).
3.	<p>Describe the potential members of a class.</p> <ol style="list-style-type: none"><li>1. Constants – constant values associated with the class (for example <code>Math.PI</code>), or with instances thereof</li><li>2. Fields – variables of the class or its instances</li><li>3. Methods – distinct named code blocks that belong to the class (static methods) or its instances</li><li>4. Constructors – special methods which are invoked on the instantiation of instances of the class</li><li>5. Static initialization blocks – blocks of code which initialize class variables</li><li>6. Instance initialization blocks – blocks of code that can be used to initialize instance variables (they work in conjunction with constructors)</li><li>7. Types – other classes, as well as implemented interfaces</li></ol> <p><b>Note:</b> Not all of these members are required to be present in a given class at once.</p>
4.	In Java, can a class be defined within another class?
	Yes. (In the solution to exercise 3 above it was stated that a class can have other classes defined within it).
5.	<p>Explain the concept of a “top-level” class.</p> <p>A top-level class is simply a class that is <i>not</i> defined <u>within</u> another class. (In the solution to exercise 3 above it was stated that a class can have other classes defined within it).</p> <p>All the classes defined in this chapter are top-level classes.</p>
6.	<p>What is a nested class?</p> <p>A brief description of a nested class is “a class which is defined within another class”. We look more closely at the different kinds of nested classes in Chapter 24 entitled <i>Classes II</i>.</p>
7.	<p><i>top-level class access modifiers</i></p> <p>Explain the meaning of each of the following top-level class access modifiers:</p> <ol style="list-style-type: none"><li>1. <code>public</code></li><li>2. <code>package-private</code></li></ol> <p>1. <code>public</code> – the class in question can be accessed by any other class anywhere</p>

## Practice Your Java Level 1

	<p>2. <b>package-private</b> – the class can only be accessed by other items defined in the same package as itself  <b>Note:</b> “<b>package-private</b>” is not explicitly written in our code, rather, if <b>public</b> is not specified then the default class access level for the class in question is package-private. Conclusion: if you want the class to be package-private then don’t add any access modifier to its definition.</p>
8.	<p><i>class member access modifiers</i></p> <p>Describe what each of the following class member access modifiers means:</p> <ol style="list-style-type: none"> <li>1. <b>public</b></li> <li>2. <b>protected</b></li> <li>3. <b>no modifier specified</b></li> <li>4. <b>private</b></li> </ol> <p>1. <b>public</b> – the class member in question can be accessed directly by any other class anywhere, be it a subclass (we look at the concept of subclasses later), other types in the package of which its containing class is a part, or types in any other package. In summary, anything anywhere can access a member with the access level <b>public</b>.</p> <p>2. <b>protected</b> – can be accessed by members of its class, subclasses, or other members of the package of which it is a part (contrast this with <b>public</b>).</p> <p>3. <b>no modifier specified</b> – the class member in question can only be accessed by members of its class and its package; even subclasses of the class with the member with this unspecified permission cannot access such members unless the subclass itself is in the same package! This access level is called “<i>package-private</i>”. This is the default member access level if no other access level is explicitly specified.</p> <p>4. <b>private</b> – the member is directly accessible only by elements of the very class of which it is a part.</p> <p>For ease of recollection, note that the access levels listed are in ascending order of restriction.</p> <p>It should be understood that the member access level is subordinate to the access level of its class. For example, if a class member has an access level of <b>public</b> but its containing class has an access-level of package-private, the access level of the class member really is limited to package-private.</p>

*Summary of concepts just practiced: Definition of class. Definition of package. Potential class members. Class access modifiers. Class member access modifiers.*

9.	<p>Create the outline of a <b>public</b> class named <b>Vehicle</b>.</p> <pre><b>public class Vehicle{</b>     //The basic outline of a public class named Vehicle }</pre>
10.	<p>Add the following fields to the previously created <b>public</b> class <b>Vehicle</b>:</p> <ol style="list-style-type: none"> <li>1. A <b>private String</b> named <b>license</b></li> <li>2. A <b>private String</b> named <b>chassisNumber</b></li> <li>3. A <b>private enum</b> field <b>vehicleType</b> of type <b>VehicleTypes</b>. This <b>enum</b> is defined as follows:</li> </ol> <pre><b>public enum VehicleTypes {</b>     BICYCLE, MOTORBIKE,     CAR, TRUCK, BOAT,     TRAIN, TRAILER,     AIRPLANE; }</pre> <ol style="list-style-type: none"> <li>4. A <b>private String</b> named <b>color</b></li> <li>5. A <b>private short</b> field named <b>numberOfWheels</b></li> </ol> <pre><b>public class Vehicle {</b>     private String license;     private String chassisNumber; // Vehicle identification number     private VehicleTypes vehicleType;     private String color;     private short numberOfWheels; }</pre>

11.	<p>State why no other class can access the fields of objects of the class <b>Vehicle</b> that we just defined.</p> <p>The reason for such non-access is because every field that we have defined for this class has an access level of <b>private</b>, therefore the fields of any given object of class <b>Vehicle</b> are accessible only by the other members of the object itself.</p>
12.	<p>What is the default value assigned to fields in an object in Java?</p> <p>The default value for numerical primitives is <b>0</b>.</p> <p>The default value for object reference fields is <b>null</b>.</p>
13.	<p>What is the title of the special method which when present in any class is automatically invoked on creation of an instance of the class?</p> <p>It is called a <i>constructor</i>.</p>
14.	<p>What is the function of a constructor method?</p> <p>A constructor is primarily used to perform initialization actions on newly instantiated objects. Initialization parameters can be passed to the constructor at the time of instantiation of an object. For example, if you have a class <b>Person</b> which represents a person and thus has fields for a person's last name and first name, your constructor might receive values for these fields on object instantiation to initialize the object with.</p> <p>Beyond initializing the newly instantiated object, a constructor is free to perform any desired actions on any other aspect of the program at the time of object instantiation if so desired; its function is not limited to the initialization of objects.</p>
15.	<p>At what point in the lifetime of an object is the constructor invoked?</p> <p>It is invoked automatically on instantiation of the object.</p>
16.	<p>What is the minimum number of constructors a class can have in Java?</p> <p>The minimum number of constructors a class can have is 1, as every class must have at least one constructor. If you the programmer do not define any constructor for the class, the compiler will create one automatically; the compiler created constructor is called the <u>default constructor</u>, as it is created by default. It takes no parameters.</p>
17.	<p>Can constructors be overloaded?</p> <p>Yes constructors can be overloaded like any other method.</p>
18.	<p>Explain the need for overloaded constructors.</p> <p>This is like overloading any method; you would want an overloaded constructor in order to accommodate the different input values and input value types that you choose to initialize your objects with at instantiation time. For example, if I have a class <b>Person</b> that has fields named <b>lastName</b>, <b>firstName</b> and <b>age</b>, I can, for example, have the following constructors:</p> <pre>public Person(String lastName, String firstName); public Person(String lastName, int age); public Person(String lastName, String firstName, int age);</pre>
19.	<p>For the earlier defined class <b>Vehicle</b>, what would be the name of any of its constructor methods?</p> <p>The name of the constructor method of a class named <b>Vehicle</b> is <b>Vehicle</b>.</p> <p>Constructor methods <u>always</u> have the same name as the class itself. Constructors are the only methods that have the same name as the class in which they appear.</p>
20.	<p>What is the return type of a constructor?</p> <p>Constructors do not have a return type at all (not even <b>void</b>).</p>
21.	<p>What is a <i>parameterless</i> constructor?</p> <p>A parameterless constructor is a constructor that has no parameters passed to it. It is the constructor that is invoked when an object is instantiated without any parameters. For example we can have the following:</p> <pre>Vehicle v01 = new Vehicle(); // Example of instantiation using a parameterless constructor;                            // No parameters passed in the brackets.</pre> <p>The programmer can create the parameterless constructor, or leave the construction thereof to the compiler to do implicitly (which Java will do if the programmer has not defined any other constructor). For example, because we have not explicitly defined any constructor for our class <b>Vehicle</b>, Java has actually already created a parameterless constructor, which it (Java) will do only if the programmer has not defined any constructor for the class in question.</p>

## Practice Your Java Level 1

22.	<p><b>(IMPORTANT)</b> Describe the actions undertaken by a constructor when it is invoked.</p>
	<p>In Java, the creation of an object provokes its constructor to, as its first order of business, invoke a designated constructor in its superclass (which is the parameterless constructor by default) <i>before</i> its own body runs. The constructor of the parent will in turn invoke a designated constructor in its own superclass (the parameterless constructor is the default), before its own body runs, and it a designated constructor in its own parent, before its own body runs, etc, all the way to the top of the inheritance hierarchy at which is the constructor of the class <b>Object</b> which is the ultimate ancestor class of all Java classes. In short, a constructor of each ancestor class runs, starting from the body of the constructor at the top of the inheritance hierarchy of the class in question (which is always the class <b>Object</b>), ending with the body of the constructor of the object being instantiated.</p>
	<p>If the immediate superclass does not have a parameterless constructor defined and neither have you specified in your constructor that your constructor should invoke a different constructor in the superclass, your code will not compile.</p>
23.	<p>Describe what a default constructor is, its nature (<i>number of parameters, accessibility level</i>) and its actions when invoked.</p> <p>Also, describe what happens to the default constructor once you yourself have created/defined a constructor for your class.</p>
	<p>As previously stated, this is a constructor that is created by the compiler automatically, only if the class does not have any programmer defined constructor. Using as example the class <b>Vehicle</b> which we created earlier, we did not define any constructor for it and since we didn't define one, the compiler created one for the class implicitly and so we are thus able to instantiate objects of the class <b>Vehicle</b>. The characteristics/behavior of the default constructor are as follows:</p>
	<ol style="list-style-type: none"> <li>1. It does not take any parameters</li> <li>2. It has an accessibility level of <b>public</b></li> <li>3. It will invoke the parameterless constructor of its immediate superclass, as is the default action of all constructors, before its own body runs, as stated in the solution to preceding exercise.</li> </ol>
	<p>Once you yourself define a constructor for a given class, Java expects that any object created for the class in question will need to invoke one of the programmer defined constructors. For example if you have only the following constructor defined for the class <b>Vehicle</b>:</p>
	<pre><b>public</b> Vehicle(String vehicleColor){     color = vehicleColor; }</pre>
	<p>Then you <i>cannot</i> do the following object instantiation:</p>
	<pre><b>Vehicle</b> vehicle01 = <b>new</b> Vehicle();</pre>
	<p>The reason why you cannot do this is because once you yourself have defined at least one constructor for a class, Java will no longer create a default one (the default constructor) for you. However you are free to create your own parameterless constructor.</p>
24.	<p>With respect to the class <b>Vehicle</b> created earlier, if you have not defined a constructor for the class, what values would the object fields <b>numberOfWheels</b> and <b>color</b> of a newly instantiated <b>Vehicle</b> instance have?</p>
	<p><b>numberOfWheels</b> would have a value of <b>0</b>  <b>color</b> is initialized to <b>null</b>.</p>
	<p>The reason for these values is because field values are set to the respective default for their type.</p>
25.	<p>Write the outline for a <b>public</b> parameterless constructor for the class <b>Vehicle</b>. This constructor needn't perform any actions.</p>
	<pre><b>public</b> Vehicle(){ }</pre>
	<p>As can be seen, no return type is specified for constructors.</p>
26.	<p><i>The keyword this</i>  What does the keyword <b>this</b> mean and how is it used?  The keyword <b>this</b> means “this object with which we are dealing at present”. It is used within instance methods to refer to the instance associated with the method. Here are a few scenarios in which <b>this</b> would be used:</p>

**Scenario 1: field vs. parameter disambiguation**

If for example I have a class `Car` that has a field `color` defined as follows:

```
class Car {
    private String color;
}
```

If the class has a method `setColor(String color){...}` whose function it is to set the field `color` of a `Car` object, a problem arises because the parameter name in the method declaration matches the field name in the class itself. You see we can't say:

```
setColor(String color) {
    color = color;
    // Our intent is that the 1st color here is the object field color and the 2nd color
    // the parameter passed to the method. The compiler however can't tell which is which.
}
```

Such a statement is confusing to the compiler as it is not sure which field or variable `color` is being referenced on either side of the equation as it has to ask the question "*which one am I dealing with, the field in the object or the parameter in the method declaration?*".

Therefore, we use `this` to effect a disambiguation and instead rewrite the method as follows:

```
setColor(String color) {
    this.color = color; //says object field color (on the left) = value of passed parameter color
}
```

**Scenario 2: passing an object from a method of its own class**

Sometimes an instance method needs to pass the very object to which it pertains to another method. Recall that the object at hand refers to itself as `this`. The way in which the object passing is effected can be seen in the sample code snippet below:

```
class Car {
    sendMeElsewhere(){
        sendOff(this); // where the method sendoff is a method that takes as
                      // parameter an object of class Car
    }
}
```

**Scenario 3: invoking an overloaded constructor**

There are scenarios where there is a need to access a constructor of a class from within another constructor of the same class. In such cases `this` comes into play. This invoking of a peer constructor is called *explicit constructor invocation*. We will see this concept in action in a later exercise in this chapter.

27. Explain the concept of *explicit constructor invocation*.

*Explicit constructor invocation* is a phrase that is used to describe the invocation of a constructor of a class from another constructor of the same class, i.e. you a constructor are invoking one of your overloaded peer constructor methods directly.

28. Create a constructor that receives and sets values for all the fields of the previously defined class `Vehicle`.

```
public Vehicle(String license, String chassisNum, VehicleTypes vt, String color,
              short numWheels){
    this.license = license; //note the use of the keyword this (explained earlier)
    chassisNumber = chassisNum;
    vehicleType = vt;
    this.color = color;
    numberOfWorkingWheels = numWheels;
}
```

29. Add a `public` method named `setLicense` to the class `Vehicle`. This method should set the field `license`. The return type of the method should be `void`.

```
public class Vehicle{
    // other previously defined fields and code such as the constructor and other methods here...
    public void setLicense(String license){
```

## Practice Your Java Level 1

	<pre>         this.license = license;     } } </pre>
30.	<p>Add a public method named <code>setChassisNumber</code> to the class <code>Vehicle</code>. This method should set the field <code>chassisNumber</code>. The return type of the method should be <code>void</code>.</p> <pre> public void setChassisNumber(String chassisNumber){     this.chassisNumber = chassisNumber; } </pre>
31.	<p>In the parameterless constructor for the <code>Vehicle</code> class, add a line that prints the statement "<i>I am a Vehicle!</i>".</p> <pre> public Vehicle(){     System.out.println("I am a Vehicle!"); } </pre>
32.	<p>In the method <code>main</code> of class <code>Vehicle</code>, create an instance of the class <code>Vehicle</code>, naming it <code>v01</code>. Instantiate it using the parameterless constructor. What is the output to the console? (only method <code>main</code> of class <code>Vehicle</code> shown here):</p> <pre> public static void main(String[] args) {     Vehicle v01 = new Vehicle(); } </pre> <p>The output is: "<i>I am a Vehicle!</i>", which is what the parameterless constructor was coded to do.</p>

*Summary of concepts just practiced: class definition, constructors, constructor behavior, default constructor, parameterless constructor, explicit constructor invocation, methods, the keyword `this`, object instantiation.*

33.	<p>In Java, what is the concept of inheritance?</p> <p>Inheritance is a feature in Java by which a class can be specified to be a descendant of an already existing class (termed its “superclass”). By virtue of inheritance, the subclass “inherits” the components of its superclass. The subclass is free to create any desired extra functionality for itself and also to, within certain constraints, override what is not desired of what it inherited.</p> <p>For example, if you want to declare a class <code>Car</code> as a descendant of the class <code>Vehicle</code>, you define class <code>Car</code> as follows:</p> <pre> public class Car extends Vehicle { } </pre> <p>Observe the use of the keyword <code>extends</code> which is used to effect inheritance.</p> <p><b>Note:</b> Officially in Java it is stated that a subclass does not inherit the private members of its superclass, however in a practical sense it does. The issue really is one of direct access to the private members of its superclass; it indeed cannot directly access these inherited private members except through non-private methods that the superclass has provided for accessing them.</p>
34.	<p>What is the concept of multiple inheritance?</p> <p>Multiple inheritance is a feature in some computer languages by which a given class can have multiple superclasses and thus inherit the members of all of its superclasses.</p>
35.	<p>How is multiple inheritance implemented in Java?</p> <p>Java DOES NOT support multiple inheritance. It only supports single inheritance, that is, a class can only have one direct/immediate superclass.</p> <p>Note though that, the concept of <i>interfaces</i> can seem to have a similar effect as multiple inheritance. We look at the concept of interfaces in a later chapter.</p>
36.	<p>(<i>subclass creation</i>)</p> <p>Create the outline of a <code>public</code> class named <code>Car</code> that is a <u>subclass</u> of the earlier created class <code>Vehicle</code>.</p> <p><i>Hint: keyword extends</i></p> <pre> public class Car extends Vehicle { } </pre>

37.	<p>Create a class <code>Motorcycle</code> as a subclass of the class <code>Vehicle</code>. Give it a parameterless constructor that sets the field <code>vehicleType</code> to the value <code>MOTORBIKE</code>. The constructor should also set the field <code>numberOfWheels</code> to a value of 2.</p> <pre><code>public class Motorcycle extends Vehicle {     public Motorcycle(){ // Constructor         vehicleType = VehicleTypes.MOTORBIKE;         numberOfWheels = 2;     } }</code></pre>
38.	<p>Create a class <code>Train</code> as a subclass of the class <code>Vehicle</code>. Create for it a constructor that sets the field <code>vehicleType</code> to the value <code>TRAIN</code>.</p> <pre><code>public class Train extends Vehicle {     public Train(){ // Constructor         vehicleType = VehicleTypes.TRAIN;     } }</code></pre>

39.	<p><i>Issues in class inheritance</i>  <i>Using the methods of the superclass as your own</i>  Given the classes defined below, predict the output of the code and explain why the output is as it is.</p> <pre><code>public class A {     public void printName(){         System.out.println("I am class A!");     } }  public class B extends A {     public static void main(String args[]){         B obj01 = new B();         obj01.printName();     } }</code></pre> <p><b>Output:</b> I am class A</p> <p><b>Explanation:</b> As can be seen in this example, class <code>B</code> extends class <code>A</code>. When this happens, objects of the subclass inherit the methods of their superclass and use them as their own, that is why we can use the inherited method <code>printName</code> as if it is a direct member of class <code>B</code> in our writing the line of code <code>obj01.printName();</code>.</p>
40.	<p><i>Overriding superclass methods</i>  Given the following classes, predict the output of this code and explain why the output is as it is.</p> <pre><code>public class A {     public void printName(){         System.out.println("I am class A!");     } }  public class B extends A {     public void printName(){         System.out.println("I am class B!");     } }  public static void main(String args[]){     B obj01 = new B();     obj01.printName(); }</code></pre> <p><b>Output:</b> I am class B</p> <p><b>Explanation:</b> As can be seen in this example, class <code>B</code> extends class <code>A</code>. It can also be seen that class <code>B</code> has declared a method <code>printName</code> which is the same name as a method in its superclass.</p>

## Practice Your Java Level 1

	<p>In Java, when a subclass reuses the name of a non-static method in its superclass, this is called <b>overriding</b> the method. Java recognizes the overridden method in class <b>B</b> as the method that pertains to objects of class <b>B</b> and therefore it printed out “I am class <b>B</b>”.</p>
41.	<p><i>Behavior of final components</i></p> <p>Given the following classes, predict the output of this code and explain your observations.</p> <pre>public class A {     public final void printName(){         System.out.println("I am class A!");     } }  public class B extends A {     public printName(){         System.out.println("I am class B!");     }      public static void main(String args[]){         B obj01 = new B();         obj01.printName();     } }</pre> <p><b>Observations:</b> The code will not compile.</p> <p><b>Explanation:</b> We declared the method <b>printName</b> in class <b>A</b> to be a <b>final</b> method. A type declared as <b>final</b> cannot be overridden in a descendant class.</p>

*Summary of concepts just practiced: inheritance, method overriding, keywords extends and final (final methods that is).*

42.	<p>Modify the method <b>main</b> of class <b>Vehicle</b> to instantiate an object named <b>car01</b> of the class <b>Car</b> defined earlier. Observe and explain the output of the code.</p> <pre>public static void main(String[] args) {     Car car01 = new Car(); }</pre> <p><b>Output</b> I am a Vehicle!</p> <p>This output comes from the parameterless constructor of the superclass <b>Vehicle</b> of class <b>Car</b>!</p> <p><b>Explanation</b></p> <p>As previously explained in the solution to exercise 21, whenever an object of any given class is instantiated, the constructor of the class in question will first of all invoke a designated constructor in its superclass (<i>yes, you can choose which of your parents' constructors you want to invoke, we will see this in a later exercise</i>) before its own body runs. If no parent constructor is explicitly designated as the one to invoke, the parameterless constructor of the superclass will be invoked as can be seen in the output of this exercise. Also, as previously stated, whichever parent constructor is invoked will invoke a designated constructor in its own parents' class (the parameterless one is the default) before its own body runs, which will in turn invoke a designated constructor in its own parents' class before its own body runs and so on recursively to the top of the object hierarchy.</p> <p>We look further at this topic in the immediately following exercises.</p>
43.	<p>Predict the output of the following code:</p> <pre>public class X {     public X() {         System.out.println("Class X: Parameterless constructor!");     }     public X(int x) {         System.out.println("Class X: The constructor that takes an integer!");     } }  public class Y extends X {</pre>

	<pre> public Y() {     System.out.println("Class Y: Parameterless constructor!"); }  public static void main(String args[]){     Y obj1 = new Y(); } </pre>
	<p><b>Output</b></p> <pre> Class X: Parameterless constructor! Class Y: Parameterless constructor! </pre> <p>This exercise is similar to the preceding exercise, as it shows that when an object is instantiated, the constructor of the object will by default invoke the parameterless constructor of its parent before it itself runs as can be observed in the order of the output.</p> <p>Note: As previously stated, we can specify which of the constructors of our superclasses that we want to invoke; we will see this in exercise 46.</p>
44.	<p>Predict the output of the following code:</p> <pre> public class W {     public W() {         System.out.println("Class W: Parameterless constructor!");     }     public W(String str01) {         System.out.println("Class W: The constructor that takes a String!");     } }  public class X extends W{     public X() {         System.out.println("Class X: Parameterless constructor!");     }     public X(int x) {         System.out.println("Class X: The constructor that takes an integer!");     } }  public class Y extends X {     public Y() {         System.out.println("Class Y: Parameterless constructor!");     }      public static void main(String args[]){         Y obj1 = new Y();     } } </pre> <p><b>Output</b></p> <pre> W: Parameterless constructor! X: Parameterless constructor! Y: Parameterless constructor! </pre> <p>This exercise shows what was previously stated: that on instantiating an object, a constructor of each of its ancestor classes is invoked, recursively, up the inheritance hierarchy, resulting in a constructor of the highest ancestor/superclass running first and then that of each lower ancestor class in the hierarchy successively, eventually ending with the body of the constructor of the class of which we are instantiating an object. The default constructor that is invoked in the ancestor classes is the parameterless constructor of each ancestor unless otherwise specified.</p> <p><b>Note:</b> You can only specify which constructor of your immediate superclass you want to invoke (using the keyword <code>super</code>), not of classes above it.</p>
45.	<p>(Using the keyword <code>super</code> to invoke a superclass constructor)</p> <p>It has been stated in a previous exercise that a superclass constructor of choice can be invoked by a subclass' constructor. Briefly describe how this invocation is implemented.</p>

## Practice Your Java Level 1

	This invocation is done using the keyword <b>super</b> , passing it the arguments, if any, for the particular superclass constructor of choice.
46.	<p>Predict the output of the following code:</p> <pre>public class X {     public X() {         System.out.println("Class X: Parameterless constructor!");     }     public X(int x) {         System.out.println("Class X: The constructor that takes an integer!");     }     public X(float x) {         System.out.println("Class X: The constructor that takes a float!");     } }  public class Y extends X {     public Y() {         super(5); // Note this line inside the constructor. Calls to super must be the 1<sup>st</sup> line.         System.out.println("Class Y: Parameterless constructor!");     }     public static void main(String args[]){         Y obj1 = new Y();     } }</pre> <p><b>Output</b> <b>Class A: The constructor that takes an integer!</b> <b>Class B: Parameterless constructor!</b></p> <p>The keyword <b>super</b> is used in this context to specify which of the constructors of the superclass should be utilized. In this example, using <b>super</b>, we invoked the particular constructor of the superclass that we wanted; in this case the superclass' constructor that takes an <b>int</b> as input. It should be noted that once we explicitly invoke a constructor in our superclass, the parameterless constructor of the superclass is no longer run.</p> <p>The objective of this exercise is to show how to directly invoke a constructor of choice in the superclass rather than the default choice of the superclass' parameterless constructor.</p>

47.	<p><i>Using <b>super</b> to access overridden methods in the immediate superclass</i></p> <p>Given the class definitions below, state the expected output of the program.</p> <pre>public class Parent{     int x=25;      public void printValue(){         System.out.println("x in the parent is " + x);     } }  public class Child extends Parent{     int x=120;      public void printValue(){         System.out.println("x in the child is " + x);         super.printValue();     } }  public static void main(String args[]){     Child child01 = new Child();     child01.printValue(); }</pre>
-----	---

	<p><b>Output:</b></p> <pre>x in the child is 25 x in the parent is 120</pre> <p>As can be seen from the output of this exercise, the keyword <b>super</b> can be used to access overridden methods of an immediate superclass that have the same name as a method in the subclass from which the method in the superclass is being accessed.</p>
48.	<p>Under what circumstances is the keyword <b>super</b> used?</p> <p><i>Hint: See the immediately preceding exercises.</i></p> <p>The keyword <b>super</b> is used under the following circumstances:</p> <ol style="list-style-type: none"> <li>1. A constructor in a subclass can use it to invoke a constructor of choice in its superclass.</li> <li>2. A method in a subclass can use it to invoke a method in its superclass that it the subclass has hidden through reuse of the same method name.</li> </ol>
49.	<p>What is the ultimate ancestor class of <u>all</u> classes in Java?</p> <p><b>The ultimate ancestor class of all classes in Java is the class Object.</b> This class provides a number of methods that each of its descendant classes inherits. The reader is urged to look at the official Java documentation to see the methods that this class provides, especially the methods <b>toString</b> and <b>getClass</b>. We have used the <b>toString</b> method implicitly in this book; we will use the method <b>getClass</b> later in this chapter.</p>
50.	<p>The code below does not compile. Explain why it does not.</p> <pre>public class A {     public A(int x) {         return;         // Here we have defined a constructor which is not parameterless for this class.         // Therefore, we now only have one constructor for this class, because once we ourselves         // define a constructor, Java will no longer take on the responsibility of creating         // the parameterless one for us.     } }  public class B extends A {     static void main(String[] args){         B obj1 = new B();     } }</pre> <p>(Observe first that class <b>A</b> has only one constructor defined and that this constructor takes arguments and that class <b>B</b> only has as constructor the default constructor).</p> <p>Stated succinctly, the problem is that there is no constructor in the superclass <b>A</b> for the constructor of its subclass <b>B</b> (<b>B</b> itself has a compiler created default constructor) to invoke. A detailed explanation of the problem appears below.</p> <p>It was stated previously that once a constructor is defined for a class by the programmer, the compiler no longer takes on the responsibility of providing a parameterless constructor for the class in question. Thus, class <b>A</b> in this example does not have a parameterless constructor. This is not wrong, but in this example, it is a part of the problem. We also know that when an object of a given class is created, the constructor of the object will by default seek to invoke the parameterless constructor of its superclass. In the code sample given above, the problem is that there is no parameterless constructor in class <b>A</b> to be invoked when an object of class <b>B</b> is instantiated.</p>
51.	<p>What is the solution to the non-compilation of the code in the preceding exercise?</p> <p>Two potential solutions present themselves. These are:</p> <ol style="list-style-type: none"> <li>1. Create a parameterless constructor in the superclass.</li> <li>2. Invoke from the constructor in class <b>B</b> the only constructor that is present in the superclass by using the keyword <b>super</b> (see exercise 45).</li> </ol>

## Practice Your Java Level 1

*Summary of concepts just practiced: object instantiation, more detail on interaction with superclass constructors, overriding methods, the keyword super.*

52.	<p>Write individual <code>public</code> methods for the class <code>Vehicle</code> to return each of the following fields:</p> <ol style="list-style-type: none"> <li>1. <code>license</code></li> <li>2. <code>chassisNumber</code></li> <li>3. <code>vehicleType</code></li> <li>4. <code>color</code></li> <li>5. <code>numberOfWheels</code></li> </ol> <p>Add these methods to the class <code>Vehicle</code>. Name the methods <code>getLicense</code>, <code>getChassisNumber</code>, <code>getVehicleType</code>, <code>getColor</code> and <code>getNumberOfWheels</code> respectively.</p>	<p>The methods are as follows:</p> <pre>public String getLicense() {return(license);} public String getChassisNumber() {return(chassisNumber);} public VehicleTypes getVehicleType() {return (vehicleType);} public String getColor() {return (color);} public short getNumberOfWheels(){return (numberOfWheels);}</pre>
53.	<p>In addition to the methods written in exercises 29 and 30 to set values for the fields <code>license</code> and <code>chassisNumber</code> respectively, write individual <code>public</code> methods for the class <code>Vehicle</code> to also set each of the following fields:</p> <ol style="list-style-type: none"> <li>1. <code>vehicleType</code></li> <li>2. <code>color</code></li> <li>3. <code>numberOfWheels</code></li> </ol> <p>Name the methods <code>setVehicleType</code>, <code>setColor</code> and <code>setNumberOfWheels</code> respectively. They should all have a return type of <code>void</code>.</p>	<pre>public void setVehicleType(VehicleTypes vt) {     vehicleType = vt; }  public void setColor(String color) {     this.color = color; }  public void setNumberOfWheels(short numberOfWheels){     this.numberOfWheels = numberOfWheels; }</pre>
54.	<p>For the earlier created object <code>car01</code> (see exercise 42), enhance the code in method <code>main</code> of class <code>Vehicle</code> to print its fields <code>vehicleType</code>, <code>numberOfWheels</code> and <code>color</code> using the methods defined in exercise 52.</p> <p>Explain the output of the program.</p> <p><i>Add the following lines of code to <code>main</code> in class <code>Vehicle</code>:</i></p> <pre>System.out.println("Vehicle type=" + car01.getVehicleType()); System.out.println("number Of Wheels=" + car01.getNumberOfWheels()); System.out.println("color=" + car01.getColor());</pre> <p><b>Output</b></p> <pre>I am a Vehicle! Vehicle type=null numberOfWheels=0 color=null</pre> <p><b>Explanation</b></p> <p>The reason for the output is as follows:</p> <p><i>Line 1:</i> This is because the parameterless constructor <u>of the superclass</u> is called (This was explained in the solution to exercise 42).</p> <p><i>Line 2:</i> We did not modify this enum field, therefore it still holds its default value of <code>null</code> (recall that enums are objects).</p> <p><i>Line 3:</i> We did not modify this numerical primitive field, therefore it still holds its default value of <code>0</code>.</p> <p><i>Line 4:</i> Similarly, if not explicitly initialized, a <code>String</code> field, being an object reference, is initialized to a value of <code>null</code>.</p> <p><b>Note(s):</b> Again we observe that due to inheritance the object of class <code>Car</code> has inherited the methods of its superclass and is employing them as its own.</p>	

55.	<p>Create a parameterless constructor for the class <code>Car</code>. Have it set the <code>enum</code> field <code>vehicleType</code> to the value <code>CAR</code> and the field <code>numberOfWheels</code> to 4. Then rerun the code of the preceding exercise. Explain the output of the program.</p>
	<pre>public class Car extends Vehicle{     //... other parts of the class here ...     public Car(){ // Constructor         vehicleType = VehicleTypes.CAR;         numberOfWheels = 4;     } }</pre>
	<p>The method <code>main</code> of class <code>Vehicle</code> is as in the preceding exercise.</p>
<p><b>Output</b></p>	<pre>I am a Vehicle! Vehicle type=CAR numberOfWheels=4 color=null</pre>
	<p><b>Explanation</b></p>
	<p>Line 1: Now the parameterless constructor that we defined for the class <code>Car</code> was invoked. However this constructor invokes the parameterless constructor of its superclass <i>before</i> it itself runs. The superclass' constructor prints out the string "<i>I am a Vehicle!</i>".</p>
	<p>Line 2: The field <code>vehicleType</code> was initialized to <code>VehicleTypes.CAR</code> in the parameterless constructor for the class <code>Car</code>.</p>
<p>Line 3: We initialized the <code>numberOfWheels</code> to 4 in our parameterless constructor.</p>	<p>Line 4: Since we did not explicitly initialize it, the <code>String</code> variable <code>color</code> has been assigned the value <code>null</code>.</p>
	<p><b>Overloaded constructor</b> Create a constructor for the class <code>Car</code> which takes as parameters values for the following fields: <code>chassisNumber</code>, <code>license</code> and <code>color</code>. Afterwards, in <code>main</code> of class <code>Vehicle</code>, create an instance <code>car02</code> of the class <code>Car</code> using this new constructor, giving input values of your choice and then, using the methods defined in exercise 52, print out all the fields of object <code>car02</code>.</p>
	<p>Explain the output of the program.</p>
<pre>public class Car extends Vehicle{     // other code...      public Car(String chassisNumber, String license, String color) { // This is the new constructor         this.chassisNumber = chassisNumber;         this.license      = license;         this.color        = color;     } }</pre> <p>In <code>main</code> of class <code>Vehicle</code> we have the following code:</p> <pre>Car car02 = new Car("VIN11111232", "DRV0123A3432", "blue");  System.out.println("vehicle type=" + car02.getVehicleType()); System.out.println("number Of Wheels=" + car02.getNumberOfWheels()); System.out.println("color=" + car02.getColor()); System.out.println("license=" + car02.getLicense()); System.out.println("chassisNumber=" + car02.getChassisNumber());</pre> <p><b>Output</b></p> <pre>I am a Vehicle! vehicleType=null      ← the constructor we just used never initialized this field number Of Wheels=0    ← same reason as above color=blue license=DRV0123A3432 chassisNumber=VIN11111232</pre>	

## Practice Your Java Level 1

57.	<p><i>Calling an overloaded constructor from another constructor for the same class</i></p> <p>In the constructor designed in exercise 55 we set some of the fields of the class <code>Car</code>; in the constructor in exercise 56 we set the rest. Now, modify the constructor for class <code>Car</code> in exercise 56 to set the fields <code>vehicleType</code> and <code>color</code> to the values in the earlier created parameterless constructor (see exercise 55). Effect this by calling the parameterless constructor for the class <code>Car</code> <u>from the constructor being modified</u>, since we know that the parameterless constructor already sets those particular fields. Rerun the program.</p> <p>Explain the output of the program.</p> <p><i>Hint: <code>this();</code></i></p> <pre>public class Car extends Vehicle{     public Car(){ // Constructor         vehicleType = VehicleTypes.CAR;         numberOfWorks = 4;     }     public Car(String chassisNumber, String license, String color) {         this(); // This is invoking an overloaded constructor, which in this case happens to be                 // the parameterless constructor.         this.chassisNumber = chassisNumber;         this.license = license;         this.color = color;     } }</pre> <p>In <code>main</code> we still have:</p> <pre>Car car02 = new Car("VIN11111232", "DRV0123A3432", "blue");  System.out.println("vehicle type=" + car02.getVehicleType()); System.out.println("number Of Wheels=" + car02.getNumberOfWorks()); System.out.println("color=" + car02.getColor()); System.out.println("license=" + car02.getLicense()); System.out.println("chassisNumber=" + car02.getChassisNumber());</pre> <p><b>Output</b></p> <pre>I am a Vehicle! vehicleType=CAR number Of Wheels=4 color=blue license=DRV0123A3432 chassisNumber=VIN11111232</pre> <p>The output is as expected.</p> <p><b>Explanation</b></p> <p>Instead of copying the lines of code:</p> <pre>vehicleType = VehicleTypes.CAR; numberOfWorks = 4;</pre> <p>which are already present in the parameterless constructor to this constructor, we simply invoked the parameterless constructor which already has this code, from the other constructor by enhancing the constructor with the code</p> <pre>this(&lt;constructor parameters&gt;);</pre>
-----	---

*Summary of concepts just practiced: instance methods, overloaded constructors, invoking peer constructors (called explicit constructor invocation), using the keyword `this`.*

58.	<p><u>Class type modifiers</u></p> <p>Explain the meaning of each of the following <u>class type</u> modifiers:</p> <ol style="list-style-type: none"><li>1. <code>abstract</code></li><li>2. <code>final</code></li></ol>
-----	--

	<p>Note: There is another class modifier, <b>static</b>, which we will look at in the chapter on nested classes as it does not apply to top-level classes.</p> <ol style="list-style-type: none"> <li><b>abstract</b> – a class which we desire not be instantiable. Its members themselves may or may not be <b>abstract</b>, however, with this modifier, even the defined members thereof cannot be invoked. Only non-abstract subclasses of abstract classes can be directly instantiated. Note: Any instantiable class (i.e. any non-abstract class is called a “concrete class”).</li> <li><b>final</b> – no other class can be derived from such a class (i.e. it cannot have subclasses).</li> </ol>
59.	<p><u>Class member modifiers</u></p> <p>What does each of these <i>class member</i> modifier keywords mean?</p> <ol style="list-style-type: none"> <li><b>abstract</b></li> <li><b>final</b></li> <li><b>static</b></li> </ol> <ol style="list-style-type: none"> <li><b>abstract</b> – (this applies to methods) an abstract method is a method that is declared but unimplemented. The body of such a method is simply a semicolon, with no surrounding braces.</li> <li><b>final</b> – indicates that the member in question <u>cannot</u> be overridden in descendant classes.</li> <li><b>static</b> – indicates that the member in question belongs to the class itself, not to the instances. For example the method <code>String.equals(String, String)</code> of class <code>String</code> exists relative to the class <code>String</code> and is therefore accessed relative to the class name (see the class name at its beginning) rather than being accessed relative to the name of an object of the class. Another example of a static member is the <b>double</b> field <code>PI</code> in class <code>Math</code>; as has been seen in earlier exercises this field is accessed as <code>Math.PI</code>. Also, you have seen in a previous chapter that the methods of class <code>Math</code> are all accessed as <b>static</b> methods.</li> </ol> <p><b>Note:</b> Not all of these modifiers are valid for every potential member type. They are all valid though for methods.</p>

Summary of concepts just practiced: *class type modifiers (abstract & final), class member modifiers*.

<b>static class members, blocking class instantiation</b>	
60.	<p>What is the difference between an <i>instance member</i> of a class and a <i>static member</i> of a class?</p> <p>An <i>instance member</i> of a class can only be invoked relative to an instance (instantiated object) of a class; a <i>static member</i> however is invoked relative to the class itself; i.e. a static member operates without reference to any object.</p> <p>Let us take examples from the class <code>Float</code> which has both instance and static methods. The static method <code>Float.isFinite(float)</code> is invoked relative to the class, for example as:</p> <pre>boolean trueFalse = Float.isFinite(700.55f);</pre> <p>As you can see, the method is invoked relative to the class name <code>Float</code>.</p> <p>On the other hand, the instance method <code>shortValue()</code> of the same class would have to be called for a given instance as <code>&lt;instance&gt;.shortValue()</code>; that is, this particular method has to be invoked relative to an instance of class <code>Float</code>. For example, to invoke it for an instance <code>float01</code> of class <code>Float</code> we would have to invoke it as <code>float01.shortValue();</code></p> <p>Now, not only can methods be static members as discussed above, but class fields can also be static members. For example in the class <code>Math</code>, the field that represents the mathematical constant <math>\pi</math> is defined as a static field which is accessed as <code>Math.PI</code>.</p> <p>Some very well known static members are the objects <code>in</code> and <code>out</code> of class <code>System</code>; you have used the object <code>out</code> extensively for console output as <code>System.out</code>. (<code>System.out</code> is an instance of class <code>PrintStream</code>, <code>System.in</code> is an instance of class <code>InputStream</code>).</p>
61.	<p>In the class below, identify which are the class variables and which are the instance variables.</p> <pre>public class Store {     public static String storeName;     public static LocalDateTime openingTime;     public static LocalDateTime closingTime;     public LocalDateTime startTime;     public LocalDateTime finishTime; }</pre>

## Practice Your Java Level 1

	<p>The class variables are identified by the keyword <b>static</b> in their declaration. These are the fields <b>storeName</b>, <b>openingTime</b> and <b>closingTime</b>.</p> <p>The instance variables are identified by the lack of the keyword <b>static</b> in their declaration. These are the fields <b>startTime</b> and <b>finishTime</b>.</p>
62.	<p>(Blocking class instantiation)</p> <p>Modify the access level of the two constructors for class <b>Car</b> to an access level of <b>private</b> and then attempt to rerun <b>main</b> of class <b>Vehicle</b>. Present and explain the output.</p> <p><i>After running the program reset the access levels of the constructors back to public.</i></p> <p>The code does not compile. It yields the following error statement “<i>The constructor Car(String, String, String) is not visible</i>”.</p> <p>The access level of <b>private</b> means that only direct members of class <b>Car</b> can access any of its private members; that would now include its constructors which we have just made <b>private</b>. Class <b>Vehicle</b> cannot access these now private constructors of class <b>Car</b> and thus it is impossible for a method in any class other than the class <b>Car</b> itself to instantiate an object of class <b>Car</b>, because class <b>Car</b> is now the only class that can access its now private constructors.</p> <p><b>Summary:</b> One thing we have learned from this is that if you give the constructors of a class an access level of <b>private</b>, you cannot create objects of that class (except a method within the class itself creates an object of that class; for example, you can instantiate an object <b>Car</b> in the method <b>main</b> of class <b>Car</b>; do try to do so).</p> <p>It must be stated though that if there are any non-private <b>static</b> members of a class whose constructors are <b>private</b>, those static members, since they belong to the class and not to instances thereof can be accessed.</p>
63.	<p>Attempt to compile and run the following program. State and explain the outcome.</p> <pre>import java.util.*; public class Experiment{     public static void main(String[] args) {         Math abc = new Math();     } }</pre> <p>The code does <u>not</u> compile. The following is the error message presented “<i>The constructor Math() is not visible</i>”. Clearly this error happened because the parameterless constructor of class <b>Math</b> has an access level modifier of <b>private</b>, thus blocking the ability of any class other than objects of the class <b>Math</b> to instantiate any object of class <b>Math</b> via this constructor; nor does it have any other public constructor; in short, instances of class <b>Math</b> cannot be created except within the class <b>Math</b> itself.</p> <p>We however can access the static members of class <b>Math</b> as we have done in previous chapters.</p> <p><b>Conclusion/Observations:</b> From this and the preceding exercise we can make the following observations/conclusions:</p> <ol style="list-style-type: none"> <li>1. If we do not want a class to be instantiated by any method other than methods in its own class, we simply make all of its constructors <b>private</b>.</li> <li>2. Even if the constructors of a class are <b>private</b>, its non-private static members are still accessible (For example we can still access <b>Math.PI</b> which is <b>public</b>).</li> </ol>
64.	<p>Why would you want to make a class uninstantiable? Use the class <b>Math</b> as an example with which to explain your position.</p> <p>Using the class <b>Math</b> as an example, we see that this class in particular is simply a cornucopia of mathematical functions. We don't need to instantiate an object in order to perform what can be called “standard calculator functions” such as computing a tangent, an exponent or suchlike; all we simply want is the numerical value of the computation. Since objects are not required for such computations, the designers of class <b>Math</b> blocked the creation of objects of class <b>Math</b>.</p> <p>Similarly, we ourselves may very well have a need to create our own classes that are simply repositories of methods for whom the existence of objects is not necessary and thus we too would prevent our class from being instantiated.</p>
65.	<p>Create a class named <b>Volume</b>, with the following <b>static public</b> methods which calculate the volume of their respective namesake geometric object:</p>

- **Sphere** which takes the radius of a sphere as parameter.
- **Cube** which takes the length of a cube as parameter.
- **Cone** which takes the radius and height of a cone as parameters.
- **Cylinder** which takes the radius and height of a cylinder as parameters.

Each of these should return a value of type **double**.

Also, ensure that the class **Volume** cannot be instantiated outside its own class.

```
public class Volume {
    private Volume() {
        // The sole constructor is made private to prevent instances of the
        // class from being initialized.
    }

    static public double Sphere(double radius){
        return((4.0/3.0)*Math.PI*radius*radius);
    }
    static double Cube(double length){
        return(Math.pow(length, 3));
    }
    static double Cone(double radius, double height){
        return((1.0/3.0)*Math.PI*radius*radius*height);
    }
    static double Cylinder(double radius, double height){
        return(Math.PI*radius*radius*height);
    }
}
```

66. Test the above created **static** class **Volume** for the following objects:

1. A sphere of radius 40,000
2. A cube of side 2.5
3. A cone of radius 50 and height 50
4. A cylinder of radius 50 and height 25

Put the code into the method **main** of a class of your choice other than the class **Volume** itself.  
Print the results out.

*We show only main below.*

```
public class PracticeYourJava {
    static void main(String args[]){
        double sphereVolume = Volume.Sphere(40000);
        double cubeVolume = Volume.Cube(2.5);
        double coneVolume = Volume.Cone(50, 50);
        double cylinderVolume = Volume.Cylinder(50, 25);
        System.out.printf("%f\n%f\n%f\n%f\n", sphereVolume, cubeVolume, coneVolume, cylinderVolume);
    }
}
```

67. Create a class named **Circle**, that defines static methods named **Area**, **Diameter** and **Circumference** which calculate their namesake values(that is *area*, *diameter*, *circumference*) for a given radius. Each of these should return a value of type **double**. Also, ensure that the class **Circle** cannot be instantiated outside of its own class.

```
public class Circle{
    public static double Area(double radius) { return (Math.PI * radius * radius); }
    public static double Diameter(double radius) { return (2 * radius); }
    public static double Circumference(double radius) { return (2 * Math.PI * radius); }

    private Circle() {
        // This private constructor ensures that the class cannot be instantiated outside
        // of this class itself.
    }
}
```

68. Using the class **Circle** created above, print the area, circumference and diameter for circles of radius 2 and 12 to two decimal places.

## Practice Your Java Level 1

	<p>We only show the method <code>main</code> here:</p> <pre>public class PracticeYourJava {     public static void main(String[] args){         double r=2; // The radius         System.out.printf("the area,circumference,diameter of a circle of radius %.2f is %.2f, %.2f, %.2f respectively\n", r, Circle.Area(r), Circle.Diameter(r), Circle.Circumference(r));         r=12; // The second radius         System.out.printf("the area,circumference,diameter of a circle of radius %.2f is %.2f, %.2f, %.2f respectively\n", r, Circle.Area(r), Circle.Diameter(r), Circle.Circumference(r));     } }</pre>
69.	<p><i>Singleton pattern</i></p> <p>Analyze and explain the features of the code below:</p> <pre>public class PrinterSpooler{     public static PrinterSpooler ps = new PrinterSpooler();     private PrinterSpooler(){}     public spoolPrintFile(String[] fileToPrint){         //This stub represents a public method that handles print spooling using the object ps         ...     } }</pre> <p>Interestingly, an object of a given class (<code>PrinterSpooler</code>) in this case, is being instantiated right inside the class definition itself! This is valid in Java.</p> <p>We also note that we have a single constructor for this class and this constructor is private. This implies that no one can create another object of class <code>PrinterSpooler</code>; the only instance of it that exists is the one we've already created directly within the class.</p> <p>Now why would you want to do such a thing? The reason is that there are certain times that we want only one instance of a given class to exist, as a single point of contact for certain operations in our code. An example of such a need is in resource control situations; we might want a single <code>PrinterSpooler</code> object in our code to manage spooling data to the printer across all of the threads in our program.</p> <p>This program design of limiting instantiation of an object to a single object for a given class has a formal name: it is called a “<i>singleton pattern</i>”.</p>

*Summary of concepts just practiced: static class members, blocking class instantiation, singleton pattern.*

70.	<p>Modify the previously created class <code>Circle</code> class so that it is now an instantiable class and its methods are no longer static.</p> <p>It should also have the following modifications:</p> <ol style="list-style-type: none"> <li>1. A constructor which saves the passed radius value into a private field of type <code>double</code> named <code>radius</code>.</li> <li>2. It should have a method named <code>getRadius</code> which returns the value of the field <code>radius</code>.</li> <li>3. It should have a method named <code>setRadius</code> which sets the value of the field <code>radius</code>.</li> <li>4. Modify the existing methods of the class so that they get the radius from the private field <code>radius</code>, rather than that value being passed in the call to the methods of the class.</li> </ol> <pre>public class Circle {     private double radius;      public double Area() {return (Math.PI * radius * radius);}     public double Diameter() {return (2 * radius);}     public double Circumference() {return (2 * Math.PI * radius);}     public void setRadius(double r) {radius = r; }     public double getRadius() {return (radius); }     public Circle(double r) {radius = r; } //Constructor }</pre>
71.	<p>Instantiate an object of the modified class <code>Circle</code> with a radius of 2 and another one with a radius of 4. Print the area, circumference and diameter of these objects to two decimal places.</p>

We only show the method `main` here:

```
public static void main(String[] args) {
    Circle circle01 = new Circle(2);
    Circle circle02 = new Circle(4);

    System.out.printf("The area,diameter,circumference of a circle of radius %f is %f, %f, %f
respectively\n", circle01.getRadius(), circle01.area(), circle01.diameter(),
circle01.circumference());

    System.out.printf("The area,diameter,circumference of a circle of radius %f is %f, %f, %f
respectively\n", circle02.getRadius(), circle02.area(), circle02.diameter(),
circle02.circumference());
}
```

### Abstract classes

72.	<p>What is an abstract class?  <i>Hint: See exercises 58 and 59.</i></p> <p>An abstract class is one of the following:  A class where some of the expected implementation thereof (methods and suchlike) is not done/complete and thus the class itself and these incomplete components have been explicitly denoted as being <b>abstract</b>. Such abstract components are merely placeholders and it is intended that they be completed in a derived class.  OR  A class which it is desired that it itself not be instantiated at all but that only classes derived from it be instantiable (note that all of the components of the class may be completely developed, however we can still choose to denote the class as being <b>abstract</b>).  In either case, an abstract class cannot be instantiated, only a non-abstract subclass of such can be.  Once <i>any</i> component of a class is denoted as <b>abstract</b>, the class itself must be marked <b>abstract</b>.</p>
73.	<p>What is the procedure for instantiating an object of an abstract class?</p> <p>An abstract classes <i>cannot</i> have instances.  Only a non-abstract descendant class (which by definition will not have any abstract components) of the abstract class can be instantiated.</p>
74.	<p>Why are abstract classes useful?</p> <p>An abstract class is a class which demands that a certain mandatory set of components (that is, all of the components tagged as <b>abstract</b> in the class) be implemented in its subclasses before the subclass is usable. By doing this, the abstract class forces a certain minimum implementation for its concrete subclasses.  For example, if I had a class named <code>Shape</code> and wanted to derive classes from it representing the shapes square, circle, triangle, pentagon and hexagon and wanted to ensure that the programmer provided methods for computing the perimeter and area of the derived subclasses, defining abstract methods in the class <code>Shape</code> for computing the perimeter and area would compel any non-abstract class that derives from <code>Shape</code> to implement the methods that compute perimeter and area.</p>
75.	<p>Write a line of code that defines an abstract method named <code>square</code> that takes a <code>float</code> for input and returns a <code>float</code>.</p> <pre>public abstract float square(float x);</pre> <p>Observe that an abstract method does not have a body like other methods, rather a semi-colon is all that is written after the method declaration.</p>
76.	<p>Define a <code>public abstract</code> class named <code>Shape</code>. We are going to use it as a superclass for subclasses that represent equilateral polygons. Ensure that the class has the following features:</p> <ol style="list-style-type: none"> <li>1. A <code>private</code> field named <code>length</code> of type <code>double</code> to store the side-length/radius of equilateral shapes.</li> <li>2. A <code>public</code> method named <code>getSideLength</code> that returns the value of the stored <code>length</code> field.</li> <li>3. A constructor that simply stores a length/radius into the field <code>length</code>.</li> <li>4. A <code>public abstract</code> method named <code>area</code> that supports the computation of area.</li> <li>5. A <code>public abstract</code> method named <code>perimeter</code> that supports the computation of the perimeter/circumference.</li> </ol>

## Practice Your Java Level 1

```
public abstract class Shape {
    private double length;
    public abstract double area();
    public abstract double perimeter();
    public double getSideLength() {return(length);}
    public Shape(double length){ // constructor
        this.length = length;
    }
}
```

77. Define the following non-abstract subclasses of the previously created class `Shape` (the formulas for the area and perimeter of each is given for your convenience):

1. `Circle`: (Area =  $\pi r^2$ ; Perimeter =  $2\pi r$ )
2. `Triangle`: (Area of equilateral triangle =  $\sqrt{3}/4 * \text{length}^2$ ; Perimeter =  $\text{length} * 3$ )
3. `Square`: (Area =  $\text{length}^2$ ; Perimeter =  $\text{length} * 4$ )
4. `Pentagon`: (Area =  $1.720477401 * \text{length}^2$ ; Perimeter =  $\text{length} * 5$ )
5. `Hexagon`: (Area =  $2.598076211 * \text{length}^2$ ; Perimeter =  $\text{length} * 6$ )

The constructors of the respective classes should simply invoke the constructor of their superclass (*Hint: super*).

```
public class Circle extends Shape
{
    public double area(){
        return (Math.PI * Math.pow(length, 2));
    }
    public double perimeter(){
        return(this.circumference());
    }
    public double circumference(){
        return (2 * Math.PI * length);
    }
    public Circle(double x){super(x);}
}

public class Square extends Shape
{
    public double area() {
        return (length * length);
    }
    public double perimeter() {
        return (length * 4);
    }
    public Square(double x){
        super(x);
    }
}

public class Hexagon extends Shape
{
    public double area(){
        return(2.598076211 * Math.pow(length, 2));
    }
    public double perimeter(){
        return (length * 6);
    }
    public Hexagon(double x){
        super(x);
    }
}

public class Triangle extends Shape
{
    public double area(){
        return((Math.sqrt(3)/(float)4) *
               length*length);
    }
    public double perimeter(){
        return (length * 3);
    }
    public Triangle(double x){
        super(x);
    }
}

public class Pentagon extends Shape
{
    public double area(){
        return(1.720477401 * Math.pow(length,2));
    }
    public double perimeter(){
        return (length * 5);
    }
    public Pentagon(double x){
        super(x);
    }
}
```

*Summary of concepts just practiced: Abstract classes, concrete subclasses of abstract classes.*

<b>Polymorphism</b>	
78.	<p>As seen earlier, the class <code>Car</code> is a descendant class of the class <code>Vehicle</code>. If we have an object <code>car01</code> of class <code>Car</code>, explain why is it valid to write the following code:</p> <pre>Vehicle v01 = car01;</pre> <p>The code in question is valid because it is a feature of “polymorphism”, which is a key concept in object oriented programming. One of the characteristics of polymorphism is that an object of a descendant class can be assigned to a variable that is of the same type as <i>any</i> of its ancestor classes.</p>
79.	<p>Given an object <code>car01</code> of class <code>Car</code>, explain why the following code is valid:</p> <pre>Object obj01 = car01;</pre> <p>The code is valid for the same reason that the preceding solution is valid, because the class <code>Object</code> is a superclass of the class <code>Car</code> (recall that the class <code>Object</code> is the ultimate superclass of all classes in Java).</p>
<b>Polymorphism and method overriding for instance methods vs. method hiding for static methods</b>	
80.	<p>Determine the output of this program and explain why it is so.</p> <pre>public class A {     public void MyName() { System.out.println("My Name is class A."); }      public class B extends A {         public void MyName() { System.out.println("My Name is class B."); } //overriding          public static void main(String[] args){             A object01 = new B(); // NOTE: We are using a variable of type class A                                   // to access an object of type class B             object01.MyName();         }     } }</pre> <p><b>Output:</b> My Name is class B.</p> <p><b>Explanation:</b> It is very important to note that we did create an object of class <code>B</code>, but assigned it to a variable of an ancestor class, class <code>A</code> (this assignment is possible by virtue of polymorphism). Nevertheless, being that <code>MyName</code> is an instance method, it is the version of this method that belongs to the actual class of the object being accessed by a variable (in this case the actual type contained in the variable <code>object01</code> is of class <code>B</code>) that is invoked.</p> <p>Contrast this with the method version that is invoked under the same conditions when the method being invoked is a static method (see the next exercise).</p>
81.	<p><i>Method Hiding</i></p> <p>Determine the output of this program and explain why it is so. (The code is only different from the code of the preceding exercise in that the method <code>MyName</code> is <code>static</code> in this exercise).</p> <pre>public class A {     static public void MyName() { System.out.println("My Name is class A."); }      public class B extends A {         static public void MyName() { System.out.println("My Name is class B."); } //overriding          public static void main(String[] args){             A object01 = new B(); // NOTE: We are using a variable of type class A                                   // to access an object of type class B             object01.MyName();         }     } }</pre> <p><b>Output:</b> My Name is class A.</p> <p><b>Explanation:</b> In Java, when a static method is invoked, the variant of the method that is invoked is the variant that belongs to the class of the variable with which the method was invoked. Contrast this with the variant of the method which is invoked when the method in question is an instance method (see preceding exercise). The reader should repeat this exercise, this time instantiating <code>object01</code> as being of class <code>B</code>, in order to confirm this explanation.</p> <p>You can only “hide” static methods, you cannot override them.</p>

## Practice Your Java Level 1

The next set of exercises show a particular advantage of polymorphism; that of being able to use a grouping that corresponds to a type of a common ancestor and also, while referring to an object using a variable that is a type of a superclass be able to access the methods that pertain to the actual class of the object being referred to.

82. Instantiate each of the following objects and put them into an array of type `Shape`: a `Circle` object of radius 2.73, a `Square` object of side length 15, a `Pentagon` object of side 5.0 and a `Hexagon` object of side 3.7.

*Only main is shown here:*

```
public class PracticeYourJava {
    public static void main(String[] args) {
        Circle obj1 = new Circle(2.73f);
        Square obj2 = new Square(15);
        Pentagon obj3 = new Pentagon(5);
        Hexagon obj4 = new Hexagon(3.7);

        //Next we define the array of type Shape and put objects of its subclasses into it.
        Shape[] shapeArray = new Shape[4];
        shapeArray[0]=obj1;
        shapeArray[1]=obj2;
        shapeArray[2]=obj3;
        shapeArray[2]=obj4;
    }
}
```

**Note:** Observe that even though `Shape` is an abstract class, we are still able to create object references of it.

83. Applying a `for` loop to the array in the preceding exercise, print out the area and perimeter of each of the objects in the array.

```
public class PracticeYourJava {
    public static void main(String[] args) {
        Circle obj1 = new Circle(2.73f);
        Square obj2 = new Square(15);
        Pentagon obj3 = new Pentagon(5);
        Hexagon obj4 = new Hexagon(3.7);

        //Next we define the array of type Shape and put the objects into it.

        Shape[] shapeArray = new Shape[4];
        shapeArray[0]=obj1; shapeArray[1]=obj2;
        shapeArray[2]=obj3; shapeArray[3]=obj4;
        for(int j=0; j < shapeArray.length; j++){
            System.out.println("Area      = " + shapeArray[i].area());
            System.out.println("Perimeter = " + shapeArray[i].perimeter());
            System.out.println();
        }
    }
}
```

**Comments:** It can be observed that even though a variable of an ancestor class was used to access each object (polymorphism), through the instrumentality of method overriding for the instance methods the appropriate method for each actual subclass was invoked.

84. Loop through the array in the preceding exercise and using the `instanceof` operator, print out what kind of object each element of the array is.

```
public class PracticeYourJava {
    public static void main(String[] args) {
        Circle obj1 = new Circle(2.73f); Square obj2 = new Square(15);
        Pentagon obj3 = new Pentagon(5); Hexagon obj4 = new Hexagon(3.7);

        //Next we define the array of type Shape and put the objects into it.

        Shape[] shapeArray = new Shape[4];
        shapeArray[0]=obj1; shapeArray[1]=obj2;
        shapeArray[2]=obj3; shapeArray[3]=obj4;
        for(int j=0; j < shapeArray.length; j++){
            if(shapeArray[i] instanceof Circle) System.out.printf("[%d] = Circle\n",i);
            else if(shapeArray[i] instanceof Square) System.out.printf("[%d] = Square\n",i);
            else if(shapeArray[i] instanceof Pentagon) System.out.printf("[%d] = Pentagon\n",i);
            else if(shapeArray[i] instanceof Hexagon) System.out.printf("[%d] = Hexagon\n",i);
        }
    }
}
```

	<pre>         }     } } </pre>
85.	<p>Loop through the array in the preceding exercise and using the method <code>getClass</code> inherited from the class <code>Object</code>, print out what kind of object each element of the array is.</p> <pre> public class PracticeYourJava {     public static void main(String[] args) {         Circle obj1 = new Circle(2.73f); Square obj2 = new Square(15);         Pentagon obj3 = new Pentagon(5); Hexagon obj4 = new Hexagon(3.7);         Shape[] shapeArray = new Shape[4];         shapeArray[0]=obj1; shapeArray[1]=obj2;         shapeArray[2]=obj3; shapeArray[3]=obj4;         for(int j=0; j &lt; shapeArray.length; j++)             System.out.println("[%d] = %s\n", j, shapeArray[j].getClass());     } } </pre> <p><b>Note:</b> Contrast this solution with the solution of the preceding exercise. This solution for the same problem achieves essentially the same result in a more concise manner.</p>

*Summary of concepts just practiced: Polymorphism, method overriding, method hiding.*

<b>final classes and methods</b>	
86.	<p>If I change the first line of the definition of the previously defined class <code>Hexagon</code> (see exercise 77) by adding the keyword <code>final</code> as shown below, what effect does it have on the class?</p> <pre> public class Hexagon extends Shape to public final class Hexagon extends Shape </pre> <p>The effect is that no subclasses can be derived from the class <code>Hexagon</code>. Think of <code>final</code> as meaning “this is the final/last class in this class hierarchy”.</p> <p>For practice, you can attempt to derive a class from this final class to observe the reaction of the compiler.</p>
87.	<p>What is the result of attempting to compile the following code and why?</p> <pre> public class A {     public final void hello(){         System.out.println("Hello I'm from class A !");     } }  public class B extends A {     public void hello(){         System.out.println("Hello I'm from class B !");     } </pre> <p>It will <i>not</i> compile because it is not possible to override a method that has been declared <code>final</code> in an ancestor class.</p>

<b>Initializers</b>	
88.	<p><i>Static Class initializers</i></p> <p>Predict the output of the following code and explain why it is so.</p> <pre> public class Experiment {     static int x;     final static int y;     final static LocalDateTime programStartTime;     static{         x=12;         y=500;         programStartTime = LocalDateTime.now();     } } </pre>

## Practice Your Java Level 1

	<pre>public static void main(String args[]){     System.out.println("y=" + y);     System.out.println("x=" + x); }</pre> <p><b>Output:</b> y=500 x=12</p> <p>The <code>final static int</code> variable <code>y</code> (which is a <code>final</code> class variable) was set to the value of <code>500</code> inside a block of code that was directly placed within the body of its class declaration. This block of code, marked with the keyword <code>static</code>, is called a <i>static class initializer</i>. It is placed in classes to initialize class variables at class initialization time.</p> <p>You can think of a static class initializer as a constructor for class variables.</p>
89.	<p>What is the function of a class initializer?</p> <p>A class initializer is used to initialize class variables at class initialization time (this happens at some point after the class has been loaded into the JVM, but before the class is used in the program). There are times when the values to be assigned to class variables are not known at compile time, but rather determined at run-time (for example a value passed on the command line). Static class initializers afford us the ability to write the necessary blocks of code to achieve this objective of run-time initialization of static class variables with data unknown at compile time and to perform any specific desired action at class startup.</p>
90.	<p>Predict the output of the following code and explain how it is so.</p> <pre>public class Experiment {     static int x;     final static int y;     final static LocalDateTime programStartTime;      static{         x=12;     }      static{         y=500;         programStartTime = LocalDateTime.now();     }      static int z=y;      public static void main(String args[]){         System.out.println("y=" + y);         System.out.println("x=" + x);     } }</pre> <p><b>Output:</b> The output is the same as in exercise 88. It is valid for multiple static initializer blocks to be present in the same class definition. The static initializer blocks are executed in the order in which they appear.</p>
91.	<p><i>Instance initializers</i></p> <p>Predict the output of the following code and explain why it is so.</p> <pre>public class Experiment {     final LocalDateTime programStartTime;      {         programStartTime = LocalDateTime.now();         //instance initializer block     }      public static void main(String args[]){         Experiment exp01 = new Experiment();         System.out.println("programStartTime = " + exp01.programStartTime);     } }</pre> <p>Here we see an example of an “instance initializer” block. It looks like a class initializer block, only that the keyword <code>static</code> is not present.</p> <p>Java actually copies the instance initializer blocks into the beginning of every constructor of the class where they</p>

	appear (i.e. they end up running before the body of the constructor itself). <b>Note:</b> the keyword <code>super</code> can be used in an instance initializer block.
92.	<p>Why do instance initializers exist?</p> <p>One reason for the existence of instance initializers is the fact that <code>final</code> instance variables cannot be initialized in a constructor, however they can be in initializer blocks.</p> <p>Another reason for the existence of instance initializers is for use in anonymous classes. Anonymous classes <u>cannot</u> define their own constructors (they are limited only to the ones that they inherited), therefore initialization blocks can provide constructor functionality for them. We look at anonymous classes in Chapter 24.</p>

Summary of concepts just practiced: final classes and methods, class (static) initializers, instance initializers.

Package Access	
93.	<p>We have the following two classes defined in different packages:</p> <pre>package package_A; class X {</pre> <p><i>and</i></p> <pre>package package_B; import package_A.*;</pre> <pre>public class Y extends X {     public static void main(String[] args){ }</pre> <p>Explain why an attempt to compile class <code>Y</code> which extends class <code>X</code> does not succeed.  <i>Hint: What access level is class X?</i></p> <p>Class <code>X</code>, which is in <code>package_A</code> has an implicit visibility level of “package-private” (see exercise 8). Class <code>Y</code> which is in a different package is attempting to extend a class which it cannot “see”. That is why this compilation attempt fails.</p>
94.	<p>We have the following two classes:</p> <pre>package package_A; public class X {     int var01; }</pre> <p><i>and</i></p> <pre>package package_B; import package_A.*;</pre> <pre>public class Y extends X {     public static void main(String[] args){         Y obj01 = new Y();         obj01.var01 = 5;     } }</pre> <p>Explain why an attempt to compile class <code>Y</code> which extends class <code>X</code> does not succeed.</p> <p>The precise reason for why the compilation attempt fails is because <code>obj01</code> which is of class <code>Y</code> is attempting to access a package-private field (<code>var01</code>) in a class that is in another package.</p>
95.	<p>We modified class <code>Y</code> from the preceding exercise by removing its attempt to access the package-private variable <code>var01</code>. Will it compile?</p> <pre>package package_B; import package_A.*;</pre> <pre>public class Y extends X {     public static void main(String[] args){         Y obj01 = new Y();     } }</pre>

## Practice Your Java Level 1

The compilation succeeds because `obj01` does not attempt to access the field `var01` (as it had attempted to in the prior exercise). Observe that it does compile despite not having direct visibility to a member of its superclass. As far as class `Y` is concerned, `var01` in its superclass is like a variable with access level `private`; as long as you don't attempt to access it, you can be compiled.

*Summary of concepts just practiced: cross-package class and class member access.*

96.	<p><i>Accessors and Mutators (getter/ setter methods): further notes on data encapsulation</i></p> <p>Explain the concept of accessor and mutator methods and their function in Java.</p> <p>The concept of accessors and mutators (or getter and setter methods) is a very simple one with a fancy name. Accessor and mutator methods are a convention by which to implement data encapsulation. You have actually implemented examples of such before in exercises 29, 30, 52 and 53; please have a quick look at the solutions to these exercises now.</p> <p>An <u>accessor method</u> is an “other than <code>private</code>” (most likely <code>public</code>) method which is used to access a private field in an object; this method has the following requirements:</p> <ol style="list-style-type: none"><li>1. its naming convention must be <code>get&lt;VariableName (with the 1<sup>st</sup> letter in uppercase)&gt;</code></li><li>2. it returns the value of the private field to which it refers</li></ol> <p>As can be seen in exercise 52, you actually implemented a set of accessor methods.</p> <p>A <u>mutator method</u> is an “other than <code>private</code>” (most likely <code>public</code>) method which is used to set the value of a private field in an object; this method has the following requirements:</p> <ol style="list-style-type: none"><li>1. its naming convention must be <code>set&lt;VariableName (with the 1<sup>st</sup> letter in uppercase)&gt;</code></li><li>2. it is passed the value that we want used to set the field</li><li>3. it sets the value of the private field to the value received (Actually, the mutator method can apply whatever transformations are required to the input value; it does not have to set the value of the field directly to what is passed to it).</li></ol> <p>As can be seen in exercise 53, you actually implemented a set of accessor methods.</p> <p>Now, why do accessor and mutator methods exist? The reason is simple: encapsulation of data. If for example we wanted to access a given field in an object from another object, we have the programming option of making the field <code>public</code>. This is not optimal, because it violates a fundamental tenent of object-oriented programming, which is this key concept of encapsulation of data. Making the data field <code>public</code> implies free/unguarded access to the field by any object in the program; this can lead to the corruption of the internal state of the object which has presented a field as public which really should be private. If such corruption occurs, a search has to be made throughout the source code to see which part of the code corrupted the field in question. The accessor and mutator methods serve as single points of entry to the field in question (for reading and writing respectively). Any attempt to read or write the data can then be trapped by a single breakpoint in either case, making it a lot easier to identify the source of errors.</p> <p>The reader should also note that the notion of the naming convention for accessors and mutators is built into JavaBeans which expects to see this convention in place (JavaBeans are beyond the scope of this volume).</p>
-----	---

*Summary of concepts just practiced: accessor and mutator methods.*

# Chapter 23. Object References & Primitive Type Variables: A Deeper Understanding

This very important chapter addresses the need to have a proper understanding of the concepts of how object references and primitives are passed from method to method in Java. While indeed exercises on these topics have been presented implicitly to a degree in previous chapters, there is a need for focused exercises on the topic. This chapter presents exercises with which to hone your skills in these concepts.

As you go through the exercises and solutions in this chapter, you may observe a degree of restatement. This is deemed necessary, given the importance of the topics in this chapter.

1.	What is a <i>type</i> ?	A <i>type</i> is a specific “template” for whatever data it is appropriate for. In Java there are two general categories of types, primitives types and reference types. Whatever is not a primitive type (we have dealt with all of them earlier) is a reference type.  Examples of types include the type <code>double</code> , which is a <u>primitive type</u> that is appropriate for floating point values, the type <code>Double</code> is an appropriate <u>reference type</u> for floating point objects as we have seen earlier.  In Java the programmer is free to create more reference types, but not primitive types. These “templates” are necessary for the compiler to determine many things regarding the data at hand, such as how much memory to use to store the data, where in memory to store the data and what data manipulations/handling are valid for the data in question.
2.	Define what is meant in Java by the term <i>primitive type</i> .	A primitive type is one whose variable directly contains its data. Elucidating further by example, if I have an <code>int</code> variable named <code>x01</code> , “ <code>x01</code> ” is the name assigned to a location in the computer’s memory and the contents of that memory location is whatever value I assign to <code>x01</code> . <i>(Contrast the primitive type with the reference type)</i>
3.	List the different primitive types available in Java.	Primitive types include the numerical types <code>byte</code> , <code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> and the types <code>boolean</code> and <code>char</code> .
4.	What is a <i>primitive variable</i> ?	For a primitive type, a variable is simply a named space in computer memory for a data item. Of course the variable also corresponds to the name we are giving the data item. So when for example we say <code>x=5</code> , we are giving a name to a value, and also naming its location in memory (so that we can find it). The space allocated for it in memory corresponds to the space that is appropriate for the type that of the primitive that it is declared to be.
5.	Explain what a <i>reference type variable</i> is.	A “reference type” variable is one that “contains” a reference to an object. We have already used these to a large degree in this book, however we explain their details more formally here.  A reference type variable actual contains a reference to the place in memory where the object to which it refers is stored; for reference types this reference is always a reference to an <i>object</i> .  To restate the above, we would say that a variable that is a reference type variable is not the actual container of the data/object in question, but rather, contains a reference to the location where the data/object that it represents is stored.  Stated simply we can say that “a variable that is a reference type contains a reference to the address of the object it refers to”.  Here is an analogy to help explain the concept further. Say I have 10 dollars. If I was a primitive type variable I

## Practice Your Java Level 1

	<p>would have the 10 dollars directly in my pocket. If I was a reference type variable I wouldn't have the 10 dollars in my pocket, but rather a piece of paper in my pocket that has the address of where the 10 dollars is; physically you would then have to go and get the money from the address on the paper in my pocket.</p> <p>Note though, when I use a variable, whether primitive or reference, the Java runtime does the work of fetching the 10 dollars for me, so that in the case of reference types I don't have extra work to do; Java does the work for me. Think of it as Java internally doing two steps to fetch the actual value of a reference type for me; of taking the variable which contains the address of the value we want and then going to get the value from the address "written on that piece of paper", whereas for primitive types it only has to perform one step.</p> <p>The nature of the type that we are dealing with (reference or primitive) is of importance when it comes to dealing with questions of equality. For example, for variables <code>a=2</code> and <code>b=4</code> of any given <i>primitive type</i>, for the question <code>a==b</code> (i.e. is the value of <code>a</code> equal to the value of <code>b</code>), the <code>==</code> operator does a direct comparison of the values <code>2 &amp; 4</code> and gives the appropriate result (which in this case is <code>false</code>).</p> <p>In the case of reference types, if we have for example reference type variables <code>x</code> and <code>y</code> and these for example are assigned as follows: <code>x=object1</code> and <code>y=object1</code>, then the question <code>x==y</code> will yield a value of <code>false</code>! This will happen because, to continue the analogy of a reference type variable being a piece of paper on which the address of what it refers to is written, clearly <code>x</code> and <code>y</code> are different pieces of paper (even though the same thing is written on them). Now, while indeed sometimes you want to "check whether you are talking about the same sheet of paper", more often than not, you really want to compare what is written on the papers, i.e. compare the contents of what <code>x</code> and <code>y</code> are referring to. To do so you have to use the specific comparer method for the reference type in question to compare the values referred to by reference types.</p>
6.	<p>List the different types of reference types.</p> <p>Every non-primitive type in Java is a reference type. That includes classes, interfaces and also arrays (<u>even if the arrays are arrays of primitive types</u>).</p>
7.	<p>Describe the difference in passing from a <code>method<sub>A</sub></code> to a <code>method<sub>B</sub></code> (including what effect an attempt by <code>method<sub>B</sub></code> to change the value of variable passed to it) a variable of each of the following types:</p> <ul style="list-style-type: none"><li>• a primitive type</li><li>• a reference type</li></ul> <p>When you "pass" a <u>primitive type</u> variable from a <code>method<sub>A</sub></code> to a <code>method<sub>B</sub></code>, you are merely passing <i>a copy</i> of the data in the primitive type variable from <code>method<sub>A</sub></code> to the <code>method<sub>B</sub></code>. That copy does not have anything to do with the original variable!</p> <p>However, when you pass a <u>reference type</u> variable to a method, given that a reference type variable contains a reference to the address of the data that it refers to, you are actually passing <i>a copy of an address</i> (as opposed to a primitive type where you were passing a copy of data; here you are passing a copy of the address where the data is). Think of it this way; I have someone's address written on paper and I am writing that address on a different piece of paper for you.</p> <p>The result of an attempt by <code>method<sub>B</sub></code> to change the variable passed to it is:</p> <ol style="list-style-type: none"><li>1. <i>for a primitive type</i>: if <code>method<sub>B</sub></code> changes the value of a <u>parameter</u> passed to it, it doesn't have any effect on the value of the <u>variable</u> in <code>method<sub>A</sub></code> because you never really passed the variable in <code>method<sub>A</sub></code> to it, you only told <code>method<sub>B</sub></code> a value; <code>method<sub>B</sub></code> knows nothing about the original variable that contained the value.</li><li>2. <i>for a reference type</i>: if <code>method<sub>B</sub></code> changes the value in a parameter passed to it (meaning for reference types that it looks at the address passed and goes to change what is stored at that address), it definitely has an effect on the value of the variable in <code>method<sub>A</sub></code>, because, using the address analogy, <code>method<sub>B</sub></code> has been told where the data resides (an address was passed to <code>method<sub>B</sub></code>) and therefore <code>method<sub>B</sub></code> can modify the contents at that address.</li></ol> <p><b>NOTE:</b> The exception to this is cases which pertain to wrapper class objects and the <code>String</code> class. We will see this later in this chapter.</p>
8.	<p>What is meant by the statement "strings are immutable"?</p> <p>Please look again at the solution to exercise 9 of Chapter 5.</p>
9.	<p>In the program below, we pass a <code>String</code> variable from the method <code>main</code> to a method named <code>Modify</code> where the method <code>Modify</code> is supposed to modify the contents of the <code>String</code>. <i>Predict</i> and explain the result of the attempted modification of the <code>String</code>.</p>

```

public class PracticeYourJava {
    public static void Modify(String s){
        s = s + "Modified!!"; // Try to modify the String
        System.out.println(s);
    }
    public static void main(String[] args) {
        String str01 = "Hello. ";
        Modify(str01); // Call the Modify method
        System.out.println(str01);
    }
}

```

**Output:**

Hello. Modified ← this output from the method **Modify**  
Hello              ← this from the method **main**, after attempted modification by the method **Modify**

**Explanation:** As we've previously noted, **String** objects are not modifiable. Being that the type **String** is a reference type, when we passed the **String** to the method **Modify**, Java did copy/pass the address of **str01** in **main** to the method **Modify**. However, remember what happens when we try to modify a **String**; as explained earlier, being that objects of the type **String** is immutable, when method **Modify** “modified” the contents of the **String**, a new **String** object was created and its reference was assigned to the variable **s** in method **Modify** in the background; **str01** in **main** is still referring to the address of the original string content.

10. **VERY IMPORTANT:** In the program below, we attempt to modify certain variables in a called method named **Modify**. which we call from the method **main**. *Predict* the results in **main** of the attempted modification the contents of each of an **int**, **int[]**, **String**, **String[]** and a **StringBuilder**.

```

public class PracticeYourJava {
    public static void Modify(int z1, int[] a, String str01, String[] sa01, StringBuilder sb01){
        z1++; // Try to modify the int
        str01 = str01 + "Modified!!"; // Try to modify the String
        sa01[0] += "Modified!!"; // Try to modify the 1st String in the array
        sb01.append("Modified!!"); // Try to modify the StringBuilder
        if (a.length < 1) return; // Try to modify the ints in the array
        for (int count = 0; count < a.length; count++)
            a[count]++; // Try to increment each element in the array by 1
    }

    public static void main(String[] args) {
        int z1 = 500;
        int[] a = new int[] { 1, 2, 3, 4, 5 };
        String str01 = "Hello. ";
        String[] sArray01 = new String[] { "I am String array element 0: " };
        StringBuilder sb01 = new StringBuilder("Hello. ");

        Modify(z1, a, str01, sArray01, sb01); // Call the Modify method

        System.out.println("1. Integer : z1=" + z1);
        System.out.println("2. Int Array : The int array contents are: ");
        for (int count = 0; count < a.length; count++)
            System.out.println("\t" + a[count]);
        System.out.println("3. String : " + str01);
        System.out.println("4. String Array : sArray[0]=" + sArray01[0]);
        System.out.println("5. StringBuilder: " + sb01);
    }
}

```

**Results**

1. Integer : z1=500

2. Int Array : The int array contents are:  
2  
3

**Explanation (summary)**

- The **int** **z1** will not be modified, because an **int** is a primitive types and as previously discussed only a copy of its value is passed and not the variable itself.
- An array is a reference type! Therefore when we pass an array to a method, we are passing the address of

## Practice Your Java Level 1

	<pre>4 5 6  3. String      : Hello.  4. String Array : sArray[0]=I am String array element 0: Modified!! 5. StringBuilder: Hello. Modified!!</pre>	<p>the array and <u>thus implicitly passing the address of each of its contents</u>. Therefore, even though the array is an array of <code>int</code> which is a primitive type, the contents can be modified by a called method.</p> <ol style="list-style-type: none"><li>3. The <code>String</code> was not modified because it is an immutable reference type as noted in the preceding exercise.</li><li>4. An array is a reference type! Therefore the contents are changeable by a called method (see next exercise).</li><li>5. A <code>StringBuilder</code> is a reference type and therefore its contents can be modified (see exercise below).</li></ol>
11.	Explain why the <code>String str01</code> in exercise 9 was not modifiable, but the string in the array element <code>sArray01[0]</code> was modifiable when passed to another method.  We have discussed in the solution to exercise 9 why the <code>String</code> object was not modifiable. Now, why was the <code>String</code> that was in the array changeable? The reason is actually because an array, any array, can be seen as a structure which is a list of addresses of its contents. When we pass the structure to another method, the array being a reference type, has its address passed and thus implicitly the receiving method knows the “direct” contents of the structure, which as we have just said is a list of the addresses of its contents.  Therefore, when we modified a <code>String</code> in the array (in our exercise it was the <code>String</code> which was referred to by the address in position 0 of the <code>String[]</code> ), what really happened was that, because <code>String</code> objects are immutable, Java did create a new <code>String</code> object and assigned its address to the <code>String</code> array position 0.  Again, what we passed to <code>Modify</code> was the address of the <code>String[]</code> and thus implicitly the address of each of its entries; the receiving method, knowing the address of <code>String[0]</code> could change its contents which originally referred to the address of the original string, but <code>Modify</code> has now changed it to the address of the new <code>String</code> object.	
<h3>Special Issues</h3> <p>The following exercises present a number of very important concepts regarding the handling of objects of the primitive wrapper classes. The reader is urged to pay close attention to these.</p>		
12.	What does it mean when it is said that objects of a given class are immutable in Java?  What it means is that objects of that class cannot be changed! Therefore, if you attempt to change/modify the object, what Java will do is create another object that contains the modified value. For example;	<pre>String str01 = "Hello"; str01 = str01 + " how are you";</pre> A brand new <code>String</code> object that contains the value “Hello how are you” is created and <code>str01</code> is actually modified to contain a reference to the address of that new <code>String</code> object.
13.	List the immutable classes in Java.  The immutable classes in Java consist of <u>all of the wrapper classes</u> as well as the class <code>String</code> .	
14.	What is the result of attempting to modify the value of an object of an immutable class in Java?  See the solution to exercise 12.	
15.	Predict the output of the following code and explain why it is so.  <pre>public class PracticeYourJava{     public static void main(String[] args) {         Integer ref01 = 20;         ref01++;         System.out.println("value = " + ref01);     } }</pre>	<p><b>Output:</b> value = 21</p> <p><b>Explanation:</b> At the start of this code, <code>ref01</code> referred to the location of an <code>Integer</code> object that contained the value 20. We then decided to modify the value of the object that <code>ref01</code> points to. However when the value that <code>ref01</code> points to was modified, <u>given that objects of the primitive wrapper classes are immutable</u>, <code>ref01</code> was</p>

	<p>updated in the background to contain a reference to a new <code>Integer</code> object whose value was the result of the action prescribed on the value of the <code>Integer</code> object that <code>ref01</code> originally had a reference to. In short, <code>ref01</code> no longer points to the same object that it started out pointing to.</p> <p>This behavior is the same behavior as the <code>String</code> class.</p>
16.	<p>Keeping the explanation for the preceding exercise in mind, predict the output of the following code and explain why it is so.</p> <pre>public class PracticeYourJava{     public static void Modify(Integer refIn){         refIn++; // Try to modify the Integer         System.out.println("refIn = " + refIn);     }     public static void main(String[] args) {         Integer ref01 = 20;         Modify(ref01);         System.out.println("value = " + ref01);     } }</pre> <p><b>Output:</b></p> <pre>refIn = 21 value = 20</pre> <p>The value of the <code>Integer</code> referenced by <code>ref01</code> was not modified!</p> <p><b>Explanation:</b> We did indeed pass the contents of the object reference <code>ref01</code> to the method <code>Modify</code>. Therefore, the parameter <code>refIn</code> in the method <code>Modify</code> and the reference variable <code>ref01</code> have the same content; i.e. the address of an <code>Integer</code> object with the value <code>20</code>.</p> <p>Remember that objects of primitive wrapper classes are immutable, therefore when the method <code>Modify</code> attempted to modify the value referenced by <code>refIn</code>, the address in <code>refIn</code> was now updated to point to an <code>Integer</code> object which has the modified value (<code>21</code> in this example), however, <code>ref01</code> was not updated.</p> <p>The reason for why <code>ref01</code> is not be updated can be explained using the following analogy: <code>ref01</code> is a sheet of paper containing the address of an <code>Integer</code> object with the value <code>20</code>. That address was copied to the variable <code>refIn</code> of method <code>Modify</code>. Now we have two sheets of paper (<code>refIn</code> and <code>ref01</code>) with the same thing written on them. Now remember that objects of primitive wrapper classes are immutable and any changes to them result in a new object being created and the reference to the object in question being updated to refer to the new object. Therefore when an attempt was made to modify the <code>Integer</code> object that <code>refIn</code> and <code>ref01</code> point to, a new object was created and the variable <code>refIn</code> was updated with the address of the new object. In short, the old address on the paper named <code>refIn</code> was scratched out and a new address written on it. The paper named <code>ref01</code> would not have its contents updated, since the method <code>Modify</code> doesn't know anything about the paper named <code>ref01</code>, all it knows is that someone passed it the address of an <code>Integer</code> and it wrote it on a sheet of paper that it named <code>refIn</code>. Again, please note the following:</p> <p style="text-align: center;"><b>Primitive wrapper class objects are immutable. String class objects are immutable.</b></p>
17.	<p>Predict the output of the following code and explain why it is so.</p> <pre>public class PracticeYourJava{     public static void main(String[] args) {         Integer ref01 = 20;         Integer ref02 = ref01;         ref02++;         System.out.println("ref01 = " + ref01);         System.out.println("ref02 = " + ref02);     } }</pre> <p><b>Output:</b></p> <pre>ref01 = 20 ref02 = 21</pre>

## Practice Your Java Level 1

	<p><b>Explanation:</b> Think of <code>ref01</code> and <code>ref02</code> as pieces of paper. The variable <code>ref01</code> was set to contain a reference to the location of an <code>Integer</code> object with value <code>20</code>; in our paper analogy the address of the <code>Integer</code> with value <code>20</code> is now written on the piece of paper named <code>ref01</code>. The address written on <code>ref01</code> was copied to <code>ref02</code>, i.e. now both sheets of paper (<code>ref01</code> and <code>ref02</code>) have the same address written on them. We then decided to modify the object referred to by <code>ref02</code>; however given that the object being referred to is an immutable object (meaning the object itself cannot be changed), the modification caused the creation of a new object and <code>ref02</code> was updated to contain a reference to the address of the new object. This is why the values reported by <code>ref01</code> and <code>ref02</code> are different.</p> <p>Remember again: Primitive wrapper class objects are immutable. <code>String</code> class objects are immutable. Any attempt to modify an object of a primitive wrapper class or an object of the class <code>String</code> will result in the creation of a new object and the object reference with which the modification was attempted (<code>ref02</code> in this case) will be updated to contain a reference to the location of the new object.</p>
18.	<p>Predict the output of this code and explain why it is so.</p> <pre>public class PracticeYourJava {     public static void Modify(StringBuilder sb01){         sb01.append("How are you?"); // Try to modify the StringBuilder     }     public static void main(String[] args) {         StringBuilder sb01 = new StringBuilder("Hello. ");         Modify(sb01); // Call the Modify method         System.out.println(sb01);     } }</pre> <p><b>Output:</b> Hello. How are you?</p> <p><b>Explanation:</b> Objects of class <code>StringBuilder</code> are mutable, therefore any attempt to modify such objects will be successful.</p>
19.	<p>Predict the output of this code and explain why it is so.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         StringBuilder sb01 = new StringBuilder("Hello. ");         StringBuilder sb02 = sb01;         sb02.append("How are you?");         System.out.println(sb01);         System.out.println(sb02);     } }</pre> <p><b>Output:</b></p> <pre>Hello. How are you? Hello. How are you?</pre> <p><b>Explanation:</b> Imagine <code>sb01</code> and <code>sb02</code> as being two different pieces of paper. The paper named <code>sb01</code> was assigned the address of a <code>StringBuilder</code> object. We then copied the address written on <code>sb01</code> to <code>sb02</code>. A person then went to the address referred to on the paper <code>sb02</code> and modified the contents at that address (by using the method <code>append</code> in this case). Let me restate it; the person with the sheet <code>sb02</code> went to the address written thereon and modified what was stored at that address! Therefore, when a person picks up the piece of paper <code>sb01</code> and goes to the address written thereon, it is indeed whatever modifications made by anyone else who had the same address is what the person holding the piece of paper <code>sb01</code> will find at the address in question, since it is the same address.</p>
20.	<p><i>Passing objects</i></p> <p>What is the expected output of this code?</p> <pre>public class Person {     public String firstName;     public String lastName; }</pre>

```

public class PracticeYourJava{
    public static void Modify(Person y){
        y.firstName = "Me";
        y.lastName = "Myself";
    }
    public static void main(String[] args) {
        Person p = new Person(); //see class Person defined above
        Modify(p);
        System.out.printf("The person is: %s %s\n", p.firstName, p.lastName);
    }
}

```

**Output:** The person is: Me Myself

We see here that a called method can change the contents of a passed object, even if the contents are immutable reference types.

We know that when we “pass” objects, these being reference types, it is the reference to their address that is passed. With that though, implicitly the addresses of their members are passed too.

Think of an object as having an address and at that address is stored a list of addresses of all of its members.

Think about it using this analogy: I have someone’s address and at their address they have a list stuck on their fridge of the address of where they work, the address of their doctor, the address of their gym, the address of where they hid a treasure and suchlike. I can for example, change on that list the address given for the treasure (that for immutable reference types), or go to the address where they hid the treasure and change the treasure stored there (that for mutable reference types)!

21. Predict the output of this code, where we set `y = null` in the earlier shown method `Modify`.

```

public class PracticeYourJava{
    public static void Modify(Person y){
        y.firstName = "Me";
        y.lastName = "Myself";
        y=null; //In this exercise we set y to null after using it.
    }
    public static void main(String[] args) {
        Person p = new Person(); // class Person is defined in the preceding exercise
        Modify(p);
        System.out.printf("The person is: %s %s\n", p.firstName, p.lastName);
    }
}

```

**Output:** The person is Me Myself

Setting `y=null` in this code had no negative effects in `main`, because when we passed `p`, all we did was, to use the address on a piece of paper analogy, to copy the address of `p` to another sheet of paper and hand it to the method `Modify`. The method `Modify` eventually tore up its own copy of the sheet (by setting `y=null;`), but `main` still has the address on its own original sheet of paper `p`.



# Chapter 24. Classes II: Nested Classes, Local Classes, Anonymous Classes

The chapter *Classes I* was a foundational one for the concept of classes in Java. This chapter extends that one by presenting exercises on the interesting topic of non-top level classes, that is the different types of nested classes.

1.	<p>What is a <i>nested class</i>?</p> <p>A nested class is one which is declared within another class. There are <u>four</u> explicit types of nested classes, these being member classes (of which there are two types, namely the (1) <u>static nested class</u> and the non-static nested class known as an (2) <u>inner class</u>), (3) <u>local classes</u> and (4) <u>anonymous classes</u>. Member classes are directly declared within an outer class, local classes and anonymous classes are declared directly within methods.</p>
<b>Member Classes</b>	
2.	<p>Identify the static nested class and the inner class in the code below.</p> <pre>public class Shape{     public class Square{         public double area(double length){             return(length*length);         }     }      static public class Rectangle{         public double area(double length,double breadth){             return(length*breadth);         }     } }</pre> <p>The static nested class is the class <code>Rectangle</code>. This is discernible by the following two facts: (1) It is defined within another class (the class <code>Shape</code> in this case) and (2) it has the keyword <code>static</code> in front of the class definition.</p> <p>The inner class is the class <code>Square</code>. This is discernible by the following two facts (1) It is defined within another class and (2) it does <i>not</i> have the keyword <code>static</code> in its definition.</p>
<b>Member Classes — inner classes</b>	
3.	<p>Describe what an inner class is, when it makes sense to use an inner class and how objects thereof are instantiated.</p> <p>An inner class is a non-static class which is defined directly within the body of another class. It has the same style of declaration as a top-level class except that it is within another class. <u>Inner class objects can only be instantiated as sub-objects of an object of their containing class</u>. Inner class objects have access to the members of their containing object.</p> <p>Now, where would you use an inner class? An inner class is an appropriate choice when it contains information that is likely only pertinent as part of a component of its containing inner class.</p> <p>The following example shows the following:</p> <ol style="list-style-type: none"><li>1. declaration of an inner class</li><li>2. instantiation of an object of an inner class</li></ol> <pre>1. public class Shape{ 2.     public class Circle{ 3.     } 4.     public static void main(String args[]){ 5.         Shape shape01 = new Shape(); 6.         Shape.Circle circle01 = shape01.new Circle(); 7.     } }</pre>

## Practice Your Java Level 1

8. }

The inner class **Circle** is shown within the outer class **Shape**. Careful attention should be paid to the declaration on line 6 of the code which shows how to instantiate an object of an inner class. The elements of the line are described as follows:

1. The first item on the line is the fully resolved class name, in this case **Shape.Circle**.
2. The second item is the name of the inner class object that we want to create.
3. The third item is the equals sign which is used here as with any object instantiation.
4. The fourth item **shape01.new** shows that we are creating a sub-object of the outer class object. Recall that it was stated earlier that an inner class object is a sub-object of an object of its enclosing class. That is why the name of the outer class object appears, so that the compiler knows which outer class object to bind this new object to; the keyword **new** is attached as **.new**.
5. The fifth item in the line is the name of the class of the object being instantiated.

We show below how to access a variable in the outer class object that the inner object is a sub-object of.

```
1. public class Shape{  
2.     private int shapeInt;  
3.     public class Circle{  
4.         public void printOuterInfo(){  
5.             System.out.println(Shape.this.shapeInt);  
6.         }  
7.     }  
8.     public static void main(String args[]){  
9.         Shape shape01 = new Shape();  
10.        shape01.shapeInt = 50;  
11.        Shape.Circle circle01 = shape01.new Circle();  
12.    }  
13. }
```

We see on line 5 how an object of an inner class can access data of its containing object. Note the syntax well; it is **<outer class>.this.<outer class member>**. This access is possible even when the member of the containing object has an access level of **private**.

4. We want to create a class named **Alphabet** with an inner class named **Letter**. Each object of class **Letter** represents a letter in the alphabet. Each object of class **Letter** has the following two fields, **letterName** and **pronunciation**, which are both of type **String**.

The class **Alphabet** has the following features: a **private int numberofLetters** that an instance of class **Alphabet** is initialized with at instantiation, a **private String alphabetName** which contains the name of the **Alphabet** in question; the class also is initialized with a value for this variable.

Another member of the class **Alphabet** is an array of type **Letter**; this array is instantiated on object creation, with a size equal to **numberofLetters**. Write a method with which to print out the fields of the objects of class **Letter** that are stored in this array.

In the class **Letter**, include a method with which an object of class **Letter** can put itself into the array of type **Letter** that its outer object of class **Alphabet** has.

Test your code using a third class **AlphabetTest**. This class should do the following:

1. Create an object of class **Alphabet**, with parameters “Roman” and 26, to represent the Roman alphabet.
2. Create objects of class **Letter** that represents the characters “A” and “B” respectively.
3. Insert the **Letter** objects into the **Letter** array in their outer object.
4. Print out the contents of the **Letter** array in the previously created **Alphabet** object.

Note: the code lines are numbered for reference.

```
1. public class Alphabet{  
2.     private String alphabetName;  
3.     private int numberofLetters; // number of letters this alphabet has  
4.     private Letter[] letters; // We can do a forward reference to the inner class Letter  
// which we will declare later in this class  
// that will be declared later in the body of this  
5.     private int lettersIndex=0; // number of "letters" entered so far
```

```

6.     protected Alphabet(String alphabetName, int numberOfLetters){ //Constructor
7.         // We create an array of inner class Letter for as many characters as this
8.         // alphabet contains.
9.         this.alphabetName = alphabetName;
10.        this.numberOfLetters = numberOfLetters;
11.        letters = new Letter[numberOfLetters]; //instantiate the array letters
12.    }
13.    public void printAlphabet(){
14.        if(lettersIndex == 0) return;
15.        System.out.println("Alphabet Name=" + alphabetName);
16.        System.out.println("Expected # of characters in this alphabet = " + numberOfLetters);
17.        System.out.println("Currently entered # of characters = " + lettersIndex +"\n");
18.        for(int j=0; j < lettersIndex; j++)
19.            System.out.printf("Letter=%s Pronunciation=%s\n", letters[j].letterName,
                           letters[j].pronunciationDescription);
20.    }
21.    public class Letter{ //the inner class
22.        String letterName;
23.        String pronunciationDescription;
24.        Letter(String name, String description){ //Constructor of the inner class
25.            letterName = name;
26.            pronunciationDescription = description;
27.        }
28.        void addToOuterObjectList(){
29.            Alphabet.this.letters[lettersIndex] = this;
30.            lettersIndex++;
31.        }
32.    } //End inner class Letter
33. } //End outer class
34. public class testAlphabet {
35.     public static void main(String args[]){
36.         Alphabet roman = new Alphabet("Roman",26);
37.         Alphabet.Letter letter_A = roman.new Letter("A","Ayyyy");
38.         Alphabet.Letter letter_B = roman.new Letter("B","Beeee");
39.         letter_A.AddToOuterObjectList();
40.         letter_B.AddToOuterObjectList();
41.         roman.printAlphabet();
42.     }

```

Note line 29 in particular to see how from an inner class object we are able to access a field in the outer object. Lines 36 and 37 show the creation of sub-objects of the `Alphabet` object.

5.	<p>In our employee database, we store the following information about our employees:</p> <ul style="list-style-type: none"> <li>• first name</li> <li>• last name</li> <li>• We also store the salary history of the employee, this as tuples of data of the following nature: <i>(int monthsAtSalary, BigDecimal amount)</i></li> </ul> <p>Create a class <code>Employee</code> which has <code>String</code> fields named <code>firstName</code> and <code>lastName</code> for the employees' first and last name respectively. Also create in the class <code>Employee</code> an inner class named <code>SalaryHistory</code> with which to store the tuples of data indicated above. Create a method in class <code>Employee</code> to calculate the average salary of the employee over time.</p> <p>Test your code for a given employee of choice.</p> <pre> import java.math.BigDecimal; import java.math.RoundingMode;  public class Employee {     private String firstName;     private String lastName; </pre>
----	---

## Practice Your Java Level 1

```
private SalaryHistory[] salaryHistory = new SalaryHistory[100];
// We use 100 just set to a reasonable number
private int insertionCounter=0; //counter for the array above

public Employee(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

public void addToSalaryHistory(int numberOfMonths, String amount){
    SalaryHistory sh = new SalaryHistory();
    sh.numberOfMonths = numberOfMonths;
    sh.amount = new BigDecimal(amount);
    salaryHistory[insertionCounter] = sh;
    insertionCounter+=1;
}

public BigDecimal averageSalary(){
    if(insertionCounter==0) return(new BigDecimal("0"));
    int totalNumberOfMonths=0;
    BigDecimal runningTotal = new BigDecimal("0");
    for(int count=0; count < insertionCounter; count++)
    {
        BigDecimal aggregateForPeriod = (salaryHistory[count].amount).multiply(new
                                BigDecimal(salaryHistory[count].numberOfMonths));
        runningTotal = runningTotal.add(aggregateForPeriod);
        totalNumberOfMonths+= salaryHistory[count].numberOfMonths;
    }
    BigDecimal averageSalary = runningTotal.divide(new BigDecimal(totalNumberOfMonths),
                                                    RoundingMode.HALF_UP);
    return(averageSalary);
}

class SalaryHistory{
    int numberOfMonths;
    BigDecimal amount;
}

public static void main(String[] args) {
    Employee emp01 = new Employee("Tabitha","Gump");
    emp01.addToSalaryHistory(3, "3000");
    emp01.addToSalaryHistory(9, "3500");
    emp01.addToSalaryHistory(12, "4000");
    emp01.addToSalaryHistory(5, "4500");
    System.out.println("The average monthly salary of " + emp01.firstName + " " + emp01.lastName
+ " = " + emp01.averageSalary());
}
```

### Member Classes – Static nested classes

6. Describe what a static nested class is, when it makes sense to use a static nested class and how objects thereof are instantiated.

A static nested class is:

1. A class defined within a class
2. It has the class modifier **static**

**IMPORTANT:** The definition of the word **static** for a static nested class is NOT the same as the definition of **static** as used in previous discussions of classes and class members, as static inner classes can be instantiated.

The definition of **static** for a static nested class simply means that the class in question is a nested class but does NOT need for an object of the outer class to be instantiated before an object of it the inner class can be.

A static nested class makes sense when we simply want to group classes under an umbrella class. Static nested classes may use static data from their outer class if so desired, but otherwise there is no mandatory requirement for there to be any relationship between the static inner class and its outerclass; again the outer class is simply an

umbrella class.

See an example of a static nested class below:

```
public class Shape{
    static class Square{ // static nested class defined here
        protected double area(double length){
            return(length*length);
        }
    }
    public static void main(String[] args){
        Shape.Square square01 = new Shape.Square(); // object instantiation
        //as can be seen, the fully resolved class name has to be used
        //(Shape.Square in this case)
        //usage of objects of static nested classes is as normal, see below
        double area = square01.area(40f);
        System.out.println("Area of the square = " + area);
    }
}
```

The following can be seen from the code:

1. There is no need to create an object of the outer class before creating one of the static nested class (contrast this with the creation of objects of inner classes).
2. The fully resolved name of the static inner class is used when creating an object of a static nested class.

7. Enhance the code in the previous solution with the definition of static classes **Circle**, **Pentagon** and **Hexagon**. Create objects which print out the area of a given object **Circle** of radius 25 and a **Pentagon** and a **Hexagon** of side length 15 respectively.

```
public class Shape{
    static class Square{ // static nested class defined her
        protected double area(double length){
            return(length*length);
        }
    }
    static class Circle{
        protected double area(double radius){
            return(Math.PI*radius*radius);
        }
    }
    static class Pentagon{
        protected double area(double length){
            return(1.720477401 * Math.pow(length,2));
        }
    }
    static class Hexagon{
        protected double area(double length){
            return(2.598076211 * Math.pow(length, 2));
        }
    }
    public static void main(String[] args){
        Shape.Circle circle01      = new Shape.Circle();
        Shape.Square square01      = new Shape.Square();
        Shape.Pentagon pentagon01 = new Shape.Pentagon();
        Shape.Hexagon hexagon01   = new Shape.Hexagon();

        System.out.println("Area of the circle = " + circle01.area(25f));
        System.out.println("Area of the square = " + square01.area(40f));
        System.out.println("Area of the pentagon = " + pentagon01.area(15f));
        System.out.println("Area of the hexagon = " + hexagon01.area(15f));
    }
}
```

## Practice Your Java Level 1

8.	If I don't want my static inner class to be instantiable outside of itself and its containing classes, but only want to use their methods as static methods, what should I do to prevent instantiation of the static inner classes? The solution is to give the static inner class a constructor with an access level of <b>private</b> .
9.	Modify exercise 67 of Chapter 22 to have the class <b>Circle</b> as a static inner class of class <b>Shape</b> . The class should be uninstantiable and the methods should be static methods. Test your code with a <b>Circle</b> of radius 15. Output the results to 2 decimal places. <pre>public class Shape{     public static class Circle{         public static double area(double radius) { return (Math.PI * radius * radius); }         public static double diameter(double radius) { return (2 * radius); }         public static double circumference(double radius) { return (2 * Math.PI * radius); }          private Circle() {         }     }      public static void main(String[] args){         //Usage         System.out.printf("Area      = %.2f\n", Shape.Circle.area(15));         System.out.printf("Diameter = %.2f\n", Shape.Circle.diameter(15));         System.out.printf("Circumference = %.2f\n", Shape.Circle.circumference(15));     } }</pre> <p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>Observe how reference is made to static methods in the static inner class; we have to use the fully resolved class name.</li> <li>The static inner class <b>Circle</b> was made uninstantiable outside of its own class and outer class by giving it a private constructor.</li> </ol>

## Anonymous Classes

10.	Describe what an anonymous class is, when it makes sense to use one and how objects thereof are instantiated.  An anonymous class is a class that (1) has no name and (2) is defined at the point of instantiation of the object which is an instance of the anonymous class. An anonymous class is usually created <u>when a single instance of an already existing class needs the behavior of the class to be modified somewhat</u> ; this being the case, it is easy enough to define the class modification at the same point in code where the object is in question instantiated. An anonymous class definition is seen by the compiler as an expression and thus it is terminated with a semicolon. The following example shows the implementation of an anonymous class (in this case the anonymous class is a subclass of the class <b>Hello</b> ):  <pre>class Hello{     public void sayHello(){         System.out.println("Hello how are you?");     } }  public static void main(String[] args){     Hello obj01 = new Hello(){ // the left brace after constructor parameters indicates                             // the start of the anonymous class         public void sayHello(){ //overriding an inherited method             System.out.println("Hi !!!");         }     }; //end of the class. Indicated with right brace and ;     obj01.sayHello(); // Here we are using the object of the anonymous class. Run                       // the code to see indeed that it prints out "Hi !!!". }</pre> <p>As stated earlier, the anonymous class is defined at the point of instantiation of an object of the intended anonymous class. Why is it called anonymous? It is so called because normally a class is given an explicit name, whereas this one has no explicit name. While it is true that we used the class name from which we are inheriting</p>
-----	---

	<p>(Hello) to kick the instantiation off, this isn't the real class that pertains to the object. An anonymous class is identified by the existence of a left brace after the brackets that house the constructor parameters. The anonymous class <u>ends with a right brace and a semi-colon</u>; the semi-colon being the character used to end lines of code; this contributes to why an anonymous class is regarded as an expression.</p> <p>We see in this example the definition of an anonymous class which is derived from the class Hello.</p> <p>Again, anonymous classes are very useful when you have classes that really are one-use only in your code. These one-use scenarios occur a lot in Java GUI programming. For example, each button in your GUI might have a unique task; instead of having to create an explicitly named class for the actions that each button will perform, anonymous classes linked to each button instance can be created to handle such functionality.</p> <p>Also, anonymous classes can implement interfaces directly! We will do this in Chapter 26 entitled <i>Interfaces</i>.</p>
11.	<p>How do you define a new constructor in an anonymous class since it does not have a class name?</p> <p>You cannot define a new constructor for an anonymous class! It can only use an inherited constructor. If it wants to provide its own constructor-like functionality it can use an instance initialization block.</p>
12.	<p><i>Inherited constructors in an anonymous class</i></p> <p>We modify the class Hello from exercise 10 as follows (it now has a constructor) :</p> <pre>class Hello{     String name;     Hello(String name){         this.name = name;     }     public void sayHello(){         System.out.println("Hello " + name + " how are you?");     } }</pre> <p>Modify the anonymous method in main from the preceding exercise the anonymous class in the preceding question to use the name field, printing out the following in its method sayHello: "Hi &lt;name&gt; !!!"</p> <p>We show only main here.</p> <pre>public static void main(String[] args){     String nameToPass = "Jack";     Hello obj01 = new Hello(nameToPass){         public void sayHello(){             System.out.println("Hi " + name + " !!!");         }     };     obj01.sayHello(); }</pre> <p>This code shows you that anonymous classes inherit the constructors of their superclass and that the appropriate constructor is invoked on instantiation of the anonymous class as can be seen in this exercise where clearly the value of the parameter passed to the constructor of the anonymous class has been assigned to the instance variable name (we used this variable in the method sayHello).</p>
13.	<p>Modify the anonymous class above so that it uses an <u>instance initializer block</u> to store the value in its field name in uppercase. Confirm the validity of your code by modifying the method sayHello to print out just the field name.</p> <p>(only main is shown below)</p> <pre>public static void main(String[] args){     String x= "Jack";     Hello obj01 = new Hello(x){         {             name = name.toUpperCase();         } //this is the instance initializer block         public void sayHello(){             System.out.println(name);         }     }; }</pre>

## Practice Your Java Level 1

	<pre>         obj01.sayHello();     } } </pre>
14.	<p>I have the following class which represents swans.</p> <pre> class swan{     public void sayType(){         System.out.println("I am a white swan!");     } } </pre> <p>I have an array of class <code>swan</code> which contains objects that represent each of my 100 swans. I have 99 white swans and only one black swan. Create an anonymous class which will take care of the black swan; when the method <code>sayType</code> is called for the black swan it should say "I am a black swan!".</p> <p>Put all of the swans into the array, starting with the white swans and last of all put the black swan into the array. Loop through the list of swans and invoke their <code>sayType</code> method.</p> <pre> class swan{     public void sayType(){         System.out.println("I am a white swan!");     } }  public static void main(String[] args){     swan[] swanArray = new swan[100];     for(int j=0; j &lt;=98; j++)         swan[j] = new swan();      //Now the anonymous class for the black swan     swan blackSwan = new swan(){         public void sayType(){             System.out.println("I am a black swan!");         }     };      swanArray[99] = blackSwan;     for(int j=0; j &lt; 100; j++)         swanArray[j].sayType(); } } </pre>
<b>Local Classes</b>	
15.	<p>What is a local class?</p> <p>A local class is one which is defined directly <u>within a code block</u>. This code block can be a method, a loop structure, or a conditional statement block. Instances of local classes can be instantiated right within the block in which they appear.</p> <p>Local classes can be useful when the manipulations that a code block has to perform are better grouped into classes and methods themselves or for keeping related data items together inside a code block. The benefit of these being grouped as classes is readability of code.</p>
16.	<p>Write a program which determines and prints out which data points in a variable array of integers is more than 2 standard deviations away from the mean. Use a local class to store each value passed along with the square of its deviation from the mean and also to store the distance of each point from the mean.</p> <pre> public class computation01 {     public void printValues2sdAway(int[] values)     {         class dataPoint{ // The local class, inside this method             int value;             float squareDeviation;             float distanceFromMean;              dataPoint(int value){                 this.value = value;             }         }     } } </pre>

```

int numberOfDataPoints = values.length;
int sum=0;  float mean=0;  float variance=0; float standardDeviation;
dataPoint[] myDataPoints = new dataPoint[numberOfDataPoints]; //an array of the Local class
for(int j=0; j < numberOfDataPoints; j++){
    myDataPoints[j] = new dataPoint(values[j]); //instantiating the objects of the Local class
    //calculate the sum at the same time for use in determining the mean
    sum+=values[j];
}
mean = (float)sum/numberOfDataPoints;
System.out.println("The mean = " + mean);
//Now calculate the variance and standard deviation
for(int j=0; j < numberOfDataPoints; j++){
    myDataPoints[j].squareDeviation = (float)Math.pow((float)myDataPoints[j].value - mean, 2);
    variance+=myDataPoints[j].squareDeviation;
}
variance/=numberOfDataPoints; //This is the variance
standardDeviation=(float)Math.sqrt(variance);
System.out.println("The standard deviation = " + standardDeviation);
// Now we have the standard deviation, let's check which of the values is >=2 standard
// deviations from the mean.
for(int j=0; j < numberOfDataPoints; j++){
    myDataPoints[j].distanceFromMean = Math.abs((float)myDataPoints[j].value - mean);
    if(myDataPoints[j].distanceFromMean > (2*standardDeviation))
        System.out.println("The data point " + myDataPoints[j].value + " is + more than
                           2 standard deviations from the mean");
}
}

public static void main(String[] args) {
    int[] values = {3, 4, 4, 4, 5, 7, 7, 8, 9, 15};
    localClassTest x = new localClassTest();
    x.printValues2sdAway(values);
}
}

```



# Chapter 25. Collections I: Collections with Generics

Collections are built-in classes which represent well-known computer science data structures such as lists, stacks, queues and dictionaries among others.

The exercises in this chapter largely pertain to the collections classes `ArrayList`, `Stack`, `TreeSet`, `HashSet` and `HashMap`. We also make use of a number of the methods of the classes `Arrays` and `Collection`.

What are “Collections” in Java?		
“Collections” are specifically designed data structures that support the storage of a set of data items and provide individual access to the contents. Many of these collections are well-known data structures in computer science, each having a specific design with its own capabilities and performance characteristics. In general, the features that collections support include data insertion, removal, sorting and searching among other operations. Traditional collections you may recognize include stacks, queues, hash tables, trees, lists, sets, heaps, maps and variants thereof. Java presents a significant number of collections, groupable in the categories Set, List, Queue, Deque and Map.		
1.	Which would be the best out of (1) a <code>Stack</code> , (2) a <code>Queue</code> or (3) a <code>List</code> to represent each of the following scenarios: a) You put 5 books on top of each other into a box b) People lining up to pay for groceries at a single checkout counter c) Books on a bookshelf	a) A <code>Stack</code> . The reason is because a stack is a last-in first-out structure and clearly, the last book you put into the box is the first that you will bring out. b) A <code>Queue</code> . A queue is generally a first-in first-out structure and the first person on the queue is the one that is generally next to be served. c) A <code>List</code> . A bookshelf has no specific order in which you must take books out or put them in.
2.	What is the concept of “Generics” in Java?	The concept of “Generics” in Java is the means within Java to create classes or interfaces that can support any class. For example, because the <code>ArrayList</code> class in Java is written to support generics, it can support objects of <i>any class</i> as its content. Continuing with the <code>ArrayList</code> class as an example, because it can handle any data type, it is spoken/written of as <code>ArrayList&lt;E&gt;</code> , the placeholder <code>E</code> being replaced by whichever specific type the <code>ArrayList</code> at hand is referencing. The placeholder <code>E</code> is termed the generic type parameter.
<b>ArrayList</b>		
4.	Explain what is meant by the following declaration: <code>ArrayList&lt;Integer&gt; arrayList01 = new ArrayList&lt;Integer&gt;();</code>	We are telling the compiler that we want to instantiate an <code>ArrayList</code> object that will contain objects of type <code>Integer</code> . This is the first appearance of the “diamond operator” in this book. The diamond operator is used to specify parameters that can be generic; for example, for an <code>ArrayList</code> we can also specify <code>&lt;String&gt;</code> , <code>&lt;Boolean&gt;</code> , <code>&lt;any other class&gt;</code> within the diamond operator. As noted in the answer to the preceding exercise, the <code>ArrayList</code> class is a class that supports the use of a generic type parameter that lets the user specify what kind of type they want a given instance of an <code>ArrayList</code> to contain.
5.	Instantiate an <code>ArrayList&lt;String&gt;</code> variable named <code>StringList01</code> .	<code>ArrayList&lt;String&gt; StringList01 = new ArrayList&lt;String&gt;();</code> --OR-- <code>ArrayList&lt;String&gt; StringList01 = new ArrayList&lt;&gt;();</code>

## Practice Your Java Level 1

		As of Java 7, it is not required to restate within the diamond operator on the right hand side the already stated type ( <code>String</code> in this case) which was stated on the left hand side. In this chapter we will use both of these styles.
6.	Instantiate an <code>ArrayList&lt;Character&gt;</code> variable named <code>characterList01</code> .	<code>ArrayList&lt;Character&gt; characterList01 = new ArrayList&lt;&gt;();</code>
7.	Instantiate an <code>ArrayList&lt;Object&gt;</code> variable named <code>objList01</code> .	<code>ArrayList&lt;Object&gt; objList01 = new ArrayList&lt;Object&gt;();</code>
8.	Explain why the following code is invalid:  <code>ArrayList&lt;int&gt; arrayList01 = new ArrayList&lt;int&gt;();</code>	The code is invalid because the Java collections only support objects, not primitives ( <code>int</code> , <code>char</code> , <code>float</code> , etc).
9.	Explain why the following code is invalid:  <code>ArrayList&lt;boolean&gt; arrayList01 = new ArrayList&lt;boolean&gt;();</code>	For the same reason as in the preceding exercise, the code is invalid because the Java collections only work on objects, not on primitives; and <code>boolean</code> is a primitive.
10.	What is the first index of the contents of the <code>ArrayList</code> data structure in Java?	0.
11.	Write a program to instantiate a <code>ArrayList&lt;Integer&gt;</code> object named <code>arrList01</code> . Put the values 1, 60, 70, 80 and 90 into it, using the <code>ArrayList.add</code> method.	<pre>import java.util.*;  public class ListPractice {      public static void main(String[] args){         ArrayList&lt;Integer&gt; intList01 = new ArrayList&lt;Integer&gt;();          intList01.add(1);         intList01.add(60);         intList01.add(70);         intList01.add(80);         intList01.add(90);     } }</pre>
12.	Rewrite the solution to the preceding exercise, this time however, initialize the <code>ArrayList&lt;Integer&gt;</code> at the same time that it is instantiated using the <code>Arrays.asList</code> method.	<pre>import java.util.*;  public class ListPractice {      public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;Integer&gt;(Arrays.asList(1,60,70,80,90));     } }</pre> <p><b>IMPORTANT:</b> In our Collections exercises with numerical values, understand that these numerical values are not put into the Collection as primitives, but rather are auto-boxed into their wrapper class object equivalents.</p>
13.	Replace the value in the first position of the <code>ArrayList</code> object of the preceding exercise with the value 10. <i>Hint: <code>ArrayList.set</code></i>	<pre>import java.util.*;  public class ListPractice {      public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,60,70,80,90));         arrList01.set(0, 10); //remember 0 is the 1st position, 1 is the 2<sup>nd</sup>, etc.     } }</pre>
14.	Do the following: 1. Insert the value 20 into the second position of the resulting <code>ArrayList</code> object <code>arrList01</code> of the solution to the preceding exercise.	

	<p>2. Print out the contents of <code>arrList01</code>.</p> <pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,60,70,80,90));         arrList01.add(1, 20); //remember 0 is the 1st position, 1 is the 2nd         for(Integer j:arrList01){             System.out.println(j);         }     } }</pre>
15.	<p>Enhance the solution to the preceding exercise by concatenating to it the contents of the following <code>ArrayList</code>:</p> <pre>ArrayList&lt;Integer&gt; arrList02 =     new ArrayList&lt;&gt;(Arrays.asList(100,100,110,110,120,120,130,130,140,140,150,150));</pre> <p>Print out the contents of the enhanced <code>arrList01</code>.</p> <p><i>Hint: ArrayList.addAll</i></p> <pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,60,70,80,90));         arrList01.add(1,20);         ArrayList&lt;Integer&gt; arrList02 =             new ArrayList&lt;&gt;(Arrays.asList(100,100,110,110,120,120,130,130,140,140,150,150));         arrList01.addAll(arrList02);          for(Integer j:arrList01){             System.out.println(j);         }     } }</pre>
16.	<p>With reference to the resulting <code>ArrayList arrList01</code> in the solution to the preceding exercise, enhance that solution by inserting, starting at the 3<sup>rd</sup> position of <code>arrList01</code>, the contents of the following <code>ArrayList</code>:</p> <pre>ArrayList&lt;Integer&gt; arrList03 = new ArrayList&lt;&gt;(Arrays.asList(30,40,50));</pre> <p>Print out the contents of the modified <code>arrList01</code>.</p> <p><i>Hint: ArrayList.addAll(position, ArrayList)</i></p> <pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,60,70,80,90,100,100,   110,110,120,120,130,130,140,140,150,150));         ArrayList&lt;Integer&gt; arrList03 = new ArrayList&lt;&gt;(Arrays.asList(30,40,50));         arrList01.addAll(2,arrList03);         for(Integer j:arrList01){             System.out.println(j);         }     } }</pre>
17.	<p>With respect to the resulting <code>arrList01</code> of the solution to the preceding exercise print out the following:</p> <ol style="list-style-type: none"> <li>1. The position in <code>arrList01</code> at which the value 100 first appears.</li> <li>2. The last position in <code>arrList01</code> in which the value 100 appears.</li> </ol> <p><i>Hint: ArrayList.indexOf, ArrayList.lastIndexOf</i></p> <pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,   90,100,100,110,110,120,120,130,130,140,140,150,150));     } }</pre>

## Practice Your Java Level 1

	<pre>         System.out.println("First position of 100 = " + arrList01.indexOf(100));         System.out.println("Last position of 100 = " + arrList01.lastIndexOf(100));     } } </pre>
18.	<p>Print out the 2<sup>nd</sup> to the 5<sup>th</sup> elements of <code>arrList01</code>.</p> <p><i>Hint: ArrayList.get</i></p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,             90,100,100,110,110,120,120,130,130,140,140,150,150));         for(int j=1; j &lt; 5; j++){             System.out.printf("item[%d] = %s\n", j,arrList01.get(j));         }     } } </pre>
19.	<p>Copy the contents in the <code>ArrayList</code> from the preceding exercise to another <code>ArrayList</code> object. Print out the new <code>ArrayList</code>.</p> <p><i>Hint: ArrayList.addAll</i></p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,             90,100,100,110,110,120,120,130,130,140,140,150,150));          ArrayList&lt;Integer&gt; arrList02 = new ArrayList&lt;&gt;();         arrList02.addAll(arrList01);         System.out.println(arrList02);     } } </pre>
20.	<p>Copy all of the elements in <code>arrList01</code> of the solution to the preceding exercise into an <code>int[]</code>. Print the resulting <code>int[]</code> out.</p> <p><i>Hint: first copy this to an Integer[] and from there to an int[]. Also see method ArrayList.toArray.</i></p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,             90,100,100,110,110,120,120,130,130,140,140,150,150));          //First create an Integer[] of the right size         Integer[] array01 = new Integer[arrList01.size()];         //Copy the ArrayList to the Integer[]         arrList01.toArray(array01);         //Copy the Integer[] elements to the int[] 1 by 1         int[] array02 = new int[array01.length];         for(int j=0;j&lt;array01.length;j++){             array02[j] = array01[j];         }     } } </pre>
21.	<p>Remove the 1<sup>st</sup> and 3<sup>rd</sup> elements in <code>arrList01</code> of the preceding exercise.</p> <p><i>Hint: ArrayList.remove</i></p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,             90,100,100,110,110,120,120,130,130,140,140,150,150));     } } </pre>

	<pre>         arrList01.remove(2); //remove the later one(s) first         arrList01.remove(0);     } } </pre>
22.	<p>Remove every instance of the value 130 from <code>arrList01</code> of the previous exercise.</p> <p><i>Hint: ArrayList.removeAll</i></p> <pre> import java.util.*;  public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,                 90,100,100,110,110,120,120,130,130,140,140,150,150));         ArrayList&lt;Integer&gt; arrListToRemove = new ArrayList&lt;&gt;(Arrays.asList(130));         arrList01.removeAll(arrListToRemove);     } } </pre>
23.	<p>Discard every element from the resulting <code>arrList01</code> of the previous exercise except the values 100 and 110. Print out the resulting <code>ArrayList</code>.</p> <p><i>Hint: ArrayList retainAll</i></p> <pre> import java.util.*;  public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(10,20,30,40,50,60,70,80,                 90,100,100,110,110,120,120,140,140,150,150));         ArrayList&lt;Integer&gt; arrListItemsToKeep = new ArrayList&lt;&gt;(Arrays.asList(100,110));         arrList01.removeAll(arrListItemsToKeep);         for(Integer j:arrList01){             System.out.println(j);         }     } } </pre>
24.	<p>Using the method <code>removeAll</code> instead of the method <code>clear</code>, empty the <code>ArrayList arrList01</code> of the preceding solution. Before deleting all of the elements, first print out the size of the <code>ArrayList</code>. After emptying the list, confirm that it is empty by using the <code>ArrayList.isEmpty</code> method.</p> <pre> import java.util.*;  public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(100,100,110,110));         System.out.println("Current size = " + arrList01.size());         arrList01.removeAll(arrList01); // Observe the use of removeAll here         if(arrList01.isEmpty())             System.out.println("It's now empty!");         else             System.out.println("It is not empty!");     } } </pre>
25.	<p>Given two <code>ArrayList&lt;String&gt;</code> objects <code>arrList01</code> and <code>arrList02</code>, write a code fragment which shows whether the contents of <code>arrList02</code> are a subset of the contents of <code>arrList01</code>.</p> <p><i>Hint: ArrayList.containsAll</i></p> <pre> //assume we have arrList01 and arrList02 declared already . . .  boolean result = arrList01.containsAll(arrList02); if(result==true)     System.out.println("The contents of arrList02 are fully contained in arrList01"); </pre>

## Practice Your Java Level 1

	<pre> else     System.out.println("The contents of arrList02 are NOT fully contained in arrList01"); . . .</pre>
26.	<p>Repeat exercise 85 of the Chapter <i>Classes I</i>, this time using an <code>ArrayList&lt;Shape&gt;</code> instead of a <code>Shape[]</code>.</p> <p>(Only main is shown here)</p> <pre> public static void main(String[] args) {     Circle obj1 = new Circle(2.73f);     Square obj2 = new Square(15);     Pentagon obj3 = new Pentagon(5);     Hexagon obj4 = new Hexagon(3.7);      ArrayList&lt;Shape&gt; shapeList = new ArrayList&lt;&gt;();     shapeList.add(obj1); shapeList.add(obj2);     shapeList.add(obj3); shapeList.add(obj4);     for(Shape item:shapeList){         System.out.println("Object class = " + item.getClass());         System.out.println("Area      = " + item.area());         System.out.println("Perimeter = " + item.perimeter());         System.out.println();     } }</pre>
27.	<p>Applying the methods of the class <code>Collections</code> to <code>ArrayList</code> type objects</p> <p>Given the following <code>ArrayList</code> definition:</p> <pre>ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,0,0));</pre> <p>Write a program which will rotate the elements of the list to the right by 2 elements. Print the elements of the rotated <code>ArrayList</code> out.</p> <p>Hint: <code>Collections.rotate</code></p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,0,0));         Collections.rotate(arrList01, 2);         System.out.println(arrList01);     } }</pre>
28.	<p>Using the same original <code>ArrayList</code> as in the preceding exercise, note the elements of the list to the left by 2 elements. Print the elements of the rotated <code>ArrayList</code> out.</p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,0,0));         Collections.rotate(arrList01, -2);         System.out.println(arrList01);     } }</pre>
29.	<p>Print out the frequency of the number value 0 in the <code>ArrayList</code> in the preceding exercise.</p> <p>Hint: <code>Collections.frequency</code></p> <pre> import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,0,0));         int frequency = Collections.frequency(arrList01, 0);         System.out.println("The frequency is " + frequency);     } }</pre>

	<p>Replace all the instances of the value 0 in the <code>ArrayList</code> in the preceding exercise with the value 5. Print the elements of the rotated <code>ArrayList</code> out.</p> <p><i>Hint:</i> <code>Collections.replaceAll</code></p>
30.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,0,0));         Collections.replaceAll(arrList01, 0, 5);         System.out.println(arrList01);     } }</pre> <p><b>Reminder:</b> Recall that the values 0 and 5 in the <code>Collections.replaceAll</code> method call are actually autoboxed into their <code>Integer</code> equivalents.</p>
	<p>Using the methods <code>Collections.lastIndexOfSubList</code> and <code>ArrayList.set</code>, replace the last appearance of the value 5 in the resulting <code>ArrayList</code> from the solution to the preceding exercise with the value 6.</p>
31.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,5,5));         ArrayList&lt;Integer&gt; arrList02 = new ArrayList&lt;&gt;(Arrays.asList(5));         int positionofLast5 = Collections.lastIndexOfSubList(arrList01, arrList02);         if(positionofLast5 == -1)             return;         arrList01.set(positionofLast5, 6);         System.out.println(arrList01);     } }</pre>
	<p>Reverse the order of the elements in the resulting <code>ArrayList</code> of the above solution. Print out the resulting <code>ArrayList</code>.</p> <p><i>Hint:</i> <code>Collections.reverse</code></p>
32.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,5,6));         Collections.reverse(arrList01);         System.out.println(arrList01);     } }</pre>
	<p>Randomly shuffle the resulting <code>ArrayList</code> presented in the preceding exercise. Print out the resulting <code>ArrayList</code>.</p> <p><i>Hint:</i> <code>Collections.shuffle</code></p>
33.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,5,6));         Collections.shuffle(arrList01);         System.out.println(arrList01);     } }</pre>
	<p>Print out the <code>maximum</code> and <code>minimum</code> elements in the <code>ArrayList</code> from the preceding exercise.</p> <p><i>Hint:</i> <code>Collections.min</code></p>
34.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,5,6));         System.out.println("min=" + Collections.min(arrList01));         System.out.println("max=" + Collections.max(arrList01));     } }</pre>

## Practice Your Java Level 1

	<pre>}</pre>
	<p>Swap the first and last elements in the <code>ArrayList</code> from the preceding exercise. Print the resulting <code>ArrayList</code> out.</p> <p><i>Hint: Collections.swap</i></p>
35.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Integer&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList(1,2,3,4,5,6));         Collections.swap(arrList01,0,(arrList01.size() - 1));         System.out.println(arrList01);     } }</pre>
	<p>Perform the following operations on the <code>ArrayList</code> shown below:</p> <pre>ArrayList&lt;Character&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList ('Q','C','A','S','W')); 1. show the “smallest” character in the ArrayList 2. show the “largest” character in the ArrayList 3. sort and print out the sorted ArrayList</pre>
36.	<pre>import java.util.*; public class ListPractice {     public static void main(String[] args){         ArrayList&lt;Character&gt; arrList01 = new ArrayList&lt;&gt;(Arrays.asList('Q','C','A','S','W'));         System.out.println("smallest=" + Collections.min(arrList01));         System.out.println("largest=" + Collections.max(arrList01));         Collections.sort(arrList01);         System.out.println("Sorted array = " + arrList01);     } }</pre>
37.	<p>We've noted earlier that an <code>ArrayList&lt;T&gt;</code> can be used to support any kind of class. How then does the <code>Collections.sort</code> method know how to sort the different types of data types that it might be assigned to sort, as ordering is a function of the type of item (<code>Integer</code>, <code>String</code>, user defined type, etc.) passed to it?</p> <p>The <code>Collections.sort</code> method is able to sort (which involves knowing the relative order of the items it contains) the different types it supports because the type inserted into it has “implemented” the <b>interface Comparable</b>. We look at exercises on interfaces later in this book.</p>
<b>Stack</b>	
38.	<p>I have the following class which represents individual books:</p> <pre>public class Book{     String title;     public Book(String title){         this.title = title;     } }</pre> <p>I have the following copies of books: (1) Pride and Prejudice, (2) Illiad (3) Illiad (<i>2nd copy</i>), (4) As You Like It</p> <p>Do the following: Instantiate a <code>Stack</code> object, create an object of class <code>Book</code> for each of the listed books and place the <code>Book</code> objects into the <code>Stack</code> listed order.</p> <p><i>Hint: Stack.push</i></p> <pre>import java.util.*; public class Book {     String title;     public Book(String title) {         this.title = title;     }     public static void main(String[] args){         Book book01 = new Book("Pride &amp; Prejudice");     } }</pre>

	<pre> Book book02 = new Book("Illiad"); Book book03 = new Book("Illiad"); Book book04 = new Book("As You Like It");  Stack&lt;Book&gt; stack01 = new Stack&lt;&gt;(); stack01.push(book01); stack01.push(book02); stack01.push(book03); stack01.push(book04); } } </pre>
39.	<p>Determine and print out the title of the book at the top of the stack.</p> <p><i>Hint: Stack.peek</i></p> <p><i>(Add the following code to main)</i></p> <p>Add either of the following code blocks to the end of <code>main</code>:</p> <pre> Book bookOnTop = stack01.peek(); System.out.println("The top item is: " + bookOnTop.title); </pre> <p>Or, just print the <code>title</code> field out directly as follows:</p> <pre> System.out.println("The top item is: " + stack01.peek().title); </pre>
40.	<p>Print out the position of the 1<sup>st</sup> copy of <i>Illiad</i> placed into the box and the position of <i>Pride &amp; Prejudice</i>.</p> <p><i>(Add the following code to main)</i></p> <pre> System.out.println("The position of the 1st copy of the Illiad = " + stack01.search(book02)); System.out.println("The position of Pride &amp; Prejudice = " + stack01.search(book01)); </pre> <p><b>Note:</b> You will observe that the copies of <i>Illiad</i> are in position 3 and 2 respectively (keep in mind the order in which you put them into the stack). This tells us the following:</p> <ol style="list-style-type: none"> <li><b>IMPORTANT:</b> The item at <code>top</code> of a <code>Stack</code> object is in position 1.</li> <li><u>Positioning in a <code>Stack</code> object is indexed from the value 1</u>, not 0.</li> </ol>
41.	<p>Remove the book at the top of the stack and then print out the title of the book now at the top of the stack.</p> <p>Ensure that the <code>stack</code> object is not empty before attempting to remove an object from it.</p> <p><i>(Add the following code at the end of main)</i></p> <pre> if(stack01.empty()==false)     stack01.pop(); if(stack01.empty()==false)     System.out.println("The top item is now: " + stack01.peek().title); </pre>
42.	<p>The inherited <code>Stack</code> method <code>removeAt</code> treats a <code>Stack</code> object as a 0-based array, starting at the 1<sup>st</sup> element inserted. Use the <code>removeAt</code> method to remove the 1<sup>st</sup> object placed into the <code>Stack</code> object <code>stack01</code>. Print out the title of the removed <code>Book</code> object.</p> <p><i>(Add the following code at the end of main)</i></p> <pre> Book bookRemoved = stack01.remove(0); System.out.println("the removed book=" + bookRemoved.title); </pre> <p><b>Note:</b> The reader is encouraged to study the methods that the class <code>Stack</code> inherited from the class <code>Vector</code>.</p>
<b>TreeSet</b>	
43.	<p>Predict the order of data in the output of the code below and explain why it is so (note the order of input of data into the object):</p> <pre> import java.util.TreeSet; public class Set {     public static void main(String args[]){         TreeSet&lt;Integer&gt; ss01 = new TreeSet&lt;Integer&gt;();         ss01.add(1000);ss01.add(120);ss01.add(1001);ss01.add(2000);         System.out.println(ss01);     } } </pre> <p><b>Output:</b> [120, 1000, 1001, 2000]</p> <p><b>Explanation:</b> The reason for the order of the data in the output is because a <code>TreeSet</code> stores objects inserted into</p>

## Practice Your Java Level 1

	it in their “natural order” or the order determined by the <code>compareTo</code> method of the interface <code>Comparable</code> as implemented for the objects put into the <code>TreeSet</code> object.
44.	<p>Predict the output of the following code fragment:</p> <pre>TreeSet&lt;Character&gt; ss02 = new TreeSet&lt;&gt;(); ss02.add('C'); ss02.add('a'); ss02.add('X'); ss02.add('z'); ss02.add('Z'); ss02.add('A'); System.out.println(ss02);</pre> <p><b>Output:</b> [A, C, X, Z, a, z]</p> <p><b>Explanation:</b> Same as for the preceding exercise; a <code>TreeSet</code> object maintains data that is input into it in their natural order.</p>
<b>HashSet</b>	
45.	<p>Predict the output of this program and explain why it is so.</p> <pre>import java.util.*; public class Hash {     public static void main(String args[]){         HashSet&lt;Integer&gt; hashSetB = new HashSet&lt;&gt;(Arrays.asList(55,54,55,55,17,18,54));         System.out.println(hashSetB.size());         System.out.println(hashSetB);     } }</pre> <p><b>Output:</b> 4 [17, 18, 54, 55]</p> <p><b>Explanation:</b> <code>HashSet</code> is actually a type of set, which is a data type that stores unique object entries, rejecting attempts to enter into it references to the same object.</p>
46.	<p>Using a <code>HashSet</code> object, count and print out the number of unique words in the following nursery verse represented in a <code>String</code> object:</p> <pre>String rhyme = "Mary had a little lamb\nlittle lamb\nlittle lamb\nMary had a little lamb that was as white as snow";</pre> <p><i>Hint: break the String into tokens first, perhaps using <code>String.split</code></i></p> <pre>import java.util.*; public class Hash {     public static void main(String args[]){         String rhyme = "Mary had a little lamb\nlittle lamb\nlittle lamb\nMary had a little lamb that was as white as snow";         rhyme = rhyme.replace('\n', ' ');         String[] rhymeArray = rhyme.split(" ");         HashSet&lt;String&gt; hashSet01 = new HashSet&lt;String&gt;(Arrays.asList(rhymeArray));         System.out.println("Number of unique words in the verse = " + hashSet01.size());         System.out.println("The unique words are:");         for(String str01:hashSet01)             System.out.println(" " + str01);     } }</pre>
<b>HashMap</b>	
47.	<p>The list of continents, along with their sizes (in square kilometres) is as follows: (Europe, <math>1.018 \times 10^8</math>), (North America, <math>2.449 \times 10^8</math>), (Africa, <math>3.037 \times 10^8</math>), (Asia, <math>4.382 \times 10^8</math>), (Australia, <math>9.0085 \times 10^7</math>), (South America, <math>1.7840 \times 10^8</math>) and (Antarctica, <math>1.372 \times 10^8</math>). Write a program which will first create an object of class <code>HashMap</code> named <code>continents</code> and afterwards put each of these pairs of data, with the continent name as the key, into the object. Following this insertion, use an enhanced <code>for</code> loop to print out the name of each continent and its size as entered into the <code>HashMap</code> object.</p> <pre>import java.util.*; public class Hash {     public static void main(String args[]){ </pre>

	<pre> HashMap&lt;String, Double&gt; continents = new HashMap&lt;&gt;(); continents.put("Europe",1.018E8); continents.put("North America",2.449E8); continents.put("Africa",3.037E8); continents.put("Asia",4.382E8); continents.put("Australia",9.0085E7); continents.put("South America",1.7840E8); continents.put("Antarctica",1.372E8);  //Now get the keys from the HashMap; put them into an array &amp; loop through that array String[] keys = continents.keySet().toArray(new String[continents.size()]); if(keys.length &lt; 0) return; for(String str01:keys)     System.out.println("continent="+ str01 + "\t\tsize= " + continents.get(str01)); } } </pre> <p><b>Note:</b> Observe from the output that there is no predictability in the order in which the keys are returned to you.</p>
48.	<p>Modify the code in the preceding exercise to print the continents out in alphabetical order, along with their size.  <i>Hint: Sort the keys after putting them into the array keys.</i></p> <p>Simply add the following line to the code in the preceding solution after the variable <code>keys</code> is assigned its value:  <code>Arrays.sort(keys);</code></p>
49.	<p>Write a program which, using a <code>HashMap</code>, will print the continents out in ascending order of size.  <i>(Hint: use the size of the continents as the key.)</i></p> <pre> import java.util.*;  public class Hash {     public static void main(String args[]){         HashMap&lt;Double, String&gt; continents = new HashMap&lt;&gt;();         continents.put(1.018E8,"Europe");         continents.put(2.449E8,"North America");         continents.put(3.037E8,"Africa");         continents.put(4.382E8,"Asia");         continents.put(9.0085E7,"Australia");         continents.put(1.7840E8,"South America");         continents.put(1.372E8,"Antarctica");          Double[] keys = continents.keySet().toArray(new Double[continents.size()]);         Arrays.sort(keys);         if(keys.length &lt; 0)             return;         for(Double size:keys)             System.out.println("continent="+ size + "\t\tsize= " + continents.get(size));     } } </pre> <p><b>Note:</b> Here we took advantage of the fact that the size of each continent is unique and thus could serve as a unique key.</p>
50.	<p>Using a <code>HashMap</code>, determine the frequency of each unique word in the following rhyme verse:</p> <pre> String rhyme = "Mary had a little lamb\nlittle lamb\nlittle lamb\nMary had a little lamb that was as white as snow"; </pre> <p>Print each word and its frequency out.  <i>Hint: Instantiate a <code>HashMap&lt;String, Integer&gt;</code> with which to count the individual words and their frequencies. Split the String into its constituent words using <code>&lt;String&gt;.split</code>. Use a loop to go through the resulting <code>String[]</code>, inserting distinct words into the <code>HashMap</code> or incrementing the frequency of the words you've already observed and put into the <code>HashMap</code>. Also, see the methods <code>HashMap.containsKey</code> and <code>HashMap.replace</code>.</i></p> <pre> import java.util.*;  public class Hash {     public static void main(String args[]){         String rhyme = "Mary had a little lamb\nlittle lamb\nlittle lamb\nMary had a little lamb that </pre>

## Practice Your Java Level 1

	<pre>was as white as snow"; rhyme = rhyme.replace("\n", ' '); String[] rhymeArray = rhyme.split(" "); HashMap&lt;String, Integer&gt; hashMap01 = new HashMap&lt;String, Integer&gt;(); for(int count=0; count&lt;rhymeArray.length; count++){     if(hashMap01.containsKey(rhymeArray[count])){         hashMap01.replace(rhymeArray[count], hashMap01.get(rhymeArray[count])+1);     }     else         hashMap01.put(rhymeArray[count], 1); } System.out.println(hashMap01); }</pre>
<b>E1</b>	<p>Write a program which requests a string from the user. The program should then, using a <code>HashMap</code>, determine the frequency of each unique character in the string. The unique characters and their respective frequencies should be printed out in alphabetical order.</p> <p>Put this code into an infinite loop which requests successive strings from the user (which you then assess as described above) until the user types the CTRL-C sequence to end the program.</p>
<b>E2</b>	Modify the program in exercise 49 to print the continents out in descending order of size.

# Chapter 26. Interfaces

The concept of interfaces is an important one in Java, as it facilitates the enforcement of certain behaviors in the classes that *implement* the interfaces, without using an abstract class for the same purpose. The exercises presented in this chapter are introductory exercises on this important topic.

1.	<p>What is an <b>interface</b>?</p> <p>We present two complementary answers to this question.</p> <p><b>Solution #1</b></p> <p>An “interface” is defined as a “contract” (which in practice looks like a class) that specifies unimplemented methods that must be implemented by any class that states that it “implements” the interface. An interface actually looks like an abstract class.</p> <p>Interfaces are <i>implemented</i> by classes; in fact, multiple interfaces can be implemented by a single given class. An interface can <i>extend</i> other interfaces.</p> <p>The requirement for implementing an interface is that any class that <i>implements</i> the interface in question must implement all of the methods of the interface in question.</p> <p>We restate the foregoing in the following manner: Formally, <u>an interface is said to be a “contract”</u> that the implementing class enters into, mandating the developer of the implementing class to define in the implementing class all of the methods specified in the interface.</p> <p>The interface does not specify how to implement its methods; it only specifies the methods.</p> <p>Why interfaces? Interfaces are useful for ensuring that specific functionality is implemented by the classes that implement the interface <u>so that any user of those classes which implement the interface(s) is assured that the functionality specified by the interface is present in the class in question</u>. For example, if a class implements the interface <b>Comparable</b> (this is an interface that specifies a single method that is used to establish ordering among objects of the same kind), then any program that needs to sort a list of elements of the given implementing class is assured that the class itself has provided way of ordering objects of the class. This compulsion of implementation of methods is similar in concept to abstract classes, there is no limit however on the number of interfaces that a given class can implement, whereas a class can inherit directly from only one class whether that class be abstract or otherwise.</p> <p>Again, a class is said to <i>implement</i> an interface.</p> <p><b>Solution #2</b></p> <p>(We start our this solution using an analogy)</p> <p>In a real-world sense, you can think of the steering wheel, gearbox, accelerator and brake pedals of a car as an “interface” to its transmission and engine. You may not know the full nature of what goes on in the engine compartment of a vehicle or its transmission, but we have the above listed standard interfaces to it. In fact, every car is expected to implement these interfaces to its transmission and engine. These common interfaces to the engine and transmission are what allow us to drive different vehicles with minimal re-learning once we have learnt how these common interfaces work.</p> <p>Similarly in Java, an interface is a contract which stipulates specific methods that classes that choose to implement the interface in question must implement (in practice an interface looks like an abstract class). Think of it like this; every car manufacturer has essentially agreed to put a steering wheel, gearbox and foot pedals into their vehicles; in short they have agreed to “implement” these common interfaces in order to hide the actual implementation details of what the interfaces are connected to (they are connected to the engine and the transmission) from the user. Once we learn an interface we can in effect use whatever the interface is attached to. Well known built-in interfaces in Java include, <b>Set</b>, <b>List</b> and <b>CompareTo</b>.</p>
----	---

## Practice Your Java Level 1

### Usage (and Interfaces as a type)

If you were a bit eccentric, you could take your own steering wheel to your multi-car garage and attach it to your vehicle of choice for the day. It sounds strange, but it will actually work. Similarly in Java, you can declare what we can call a reference of type “the interface in question” and assign to it an object of whatever type implements the interface. That looks like this:

```
<interface type> <reference name> = object of class that implements the interface;
```

Now, understand that interfaces can't be instantiated, unlike car steering wheels which can. But we can use references of type `Interface_x` to point to an object that implements the said `Interface_x`.

For example, assume I have an interface for the steering wheel, (I'll call it `ITurning`), which supports the methods `turnLeft` and `turnRight`. We define our interface `ITurning` as follows:

```
Interface ITurning{  
    public void turnLeft();  
    public void turnRight();  
}
```

Say we have a class `Mercedes300` which “implements” `ITurning`:

```
public class Mercedes300 implements ITurning{  
    public void turnLeft(){  
        // engage/do the necessary things in the transmission to turn left in a Mercedes300  
    }  
    public void turnRight(){  
        // engage/do the necessary things in the transmission to turn right in a Mercedes300  
    }  
}
```

With the interface and the `Mercedes300` class we can do the following:

```
.  
.ITurning mySteering = new Mercedes300();  
// Note very well the use of an interface reference to an actual  
// object of a class that implements that interface in question  
mySteering.turnLeft(); //the Mercedes300 steers left  
mySteering.turnRight(); //the Mercedes300 steers right  
.  
.
```

### Multiple interfaces implemented by a single class

We also mentioned a gear for cars. So we can have an interface that we will call `IGear`. Its methods will be `shiftUp` and `shiftDown`. So the interface looks like the following:

```
Interface IGear{  
    public void shiftUp();  
    public void shiftDown();  
}
```

We'd like the class `Mercedes300` to support this interface `IGear` too. Therefore the class `Mercedes300` is extended to look as follows:

```
public class Mercedes300 implements ITurning, IGear{ // we see the class implementing  
                                                    // multiple interfaces simultaneously.  
    public void shiftUp(){  
        //Do the things in the Mercedes300 transmission to shift the gear higher  
    }  
    public void shiftDown(){  
        //Do the things in the Mercedes300 transmission to shift the gear lower  
    }  
    public void turnLeft(){  
        //engage the necessary things in the transmission to turn left in a Mercedes300  
    }  
    public void turnRight(){  
    }
```

```

    engage the necessary things in the transmission to turn right in a Mercedes300
}
}

```

We can now do the following:

```

.
.
IGear myGear = new Mercedes300(); // Since the class Mercedes300 has implemented IGear we can
// do this declaration. Do remember that the class Mercedes300 also implemented ITurning as
// as well, and it is still valid to refer to a Mercedes300 instance with a reference type
// of ITurning.
myGear.shiftUp();
myGear.shiftDown();
.
.
```

We've seen that a class can implement multiple interfaces. We've also seen that we can use references of the different interface types to access objects of the classes which implement the interface. We show below how to apply interfaces to the same object:

```

Mercedes300 mercedes01 = new Mercedes300();
ITurning mySteeringWheel = (ITurning)mercedes01; // this is valid to get an interface reference
// to an object that implements the interface

```

To turn the car right I can have either of the following lines:

```

mercedes01.turnRight();
or
mySteeringWheel.turnRight();

```

The following question arises; "why use an interface reference at all since the object in question has the methods of the interface?"

One reason to use an interface reference is to perform interface specified actions on unrelated objects. See the example below.

Say we have a class `Human` that implements `ITurning` as well. Clearly how a human being turns is different from how a car turns.

Observe now how by using an interface reference we can apply an interface defined method to items of unrelated classes which implement the interface:

```

Human human01 = new Human();
Human human02 = new Human();
mercedes01 = new Mercedes300();

ITurning[] itemList = new ITurning[3]; //We've made an array of interface references
itemList[0] = car01; //We can put any object that implements that interface
itemList[1] = person01; //into it
itemList[2] = person02;
for(int count = 0; count < 3; count++)
    itemList[count].turnRight();

```

Without interfaces you would only be able to group these unrelated items together to invoke a similarly named method if they have a common ancestor class.

#### *Benefits of interfaces (summary)*

To summarize, we can say that the benefits of interfaces are as follows:

1. Interfaces afford us a common mechanism for accessing different classes that perform the methods specified in the interface, thus obviating the need to learn the different method signatures that each different non-interface based implementation of the same functionality might present. This also allows the classes that implement the functionality to change the implementation without any fear, since all that they have to do is to leave the interface to the method constant (for example, a vehicle manufacturer can change the engine and drive-train of a car if they want, yet we will be able to drive the car with the new new components because the interfaces, the steering wheel and the gearbox are the same).
2. Interfaces allow us to group objects which are not of the same ancestry and perform actions common to

## Practice Your Java Level 1

	<p>the interface on the group of objects.</p> <p>3. Due to the fact that you can inherit directly from only one class, interfaces have a similar effect to multiple inheritance of abstract classes because a single class can implement as many interfaces as it needs.</p> <p>Whew! That was long, but necessary!</p> <p><b>Note:</b> (Java interface naming convention) There is no particular naming convention for interfaces in Java. The ones we define in this book will have the uppercase character “I” as the starting character for their names.</p>
2.	<p>Given the following interface and class definitions:</p> <pre>interface ITakeOff{...}; interface ILanding{...};  public class Helicopter implements ITakeoff,ILanding{...} public class Airplane implements ITakeoff,ILanding{...}</pre> <p>State which if any of the following lines of code is invalid:</p> <pre>public Helicopter h01 = new Helicopter(); public Helicopter h02 = new Helicopter(); public Airplane a01 = new Airplane(); public Airplane a02 = new Airplane();  ILanding landings[] = new ILanding[4]; landings[0] = (ILanding)h01; landings[1] = (ILanding)h02; landings[2] = (ILanding)h03; landings[3] = (ILanding)h04;  ArrayList&lt;ITakeOff&gt; takeOffs = new ITakeOff&lt;&gt;(); takeOffs.add((ITakeOff)h01); takeOffs.add((ITakeOff)h02); takeOffs.add((ITakeOff)a01);</pre>
	<p>All the lines of code are valid! This is because as stated in the solution to exercise 1, a reference variable of an interface type can always be used to reference objects of classes which implement it. The classes <code>Helicopter</code> and <code>Airplane</code> implement both of these interfaces <code>ITakeOff</code> and <code>ILanding</code>, therefore each of these references is valid. With regard to the declaration <code>ArrayList&lt;ITakeOff&gt;</code>, just as easily as a declaration of <code>ArrayList&lt;Integer&gt;</code> is valid, so also is a declaration <code>ArrayList&lt;interface name&gt;</code> valid.</p> <p>This exercise was given to ensure clarity and reinforcement of this sub-topic of the preceding exercise.</p>
3.	<p>Write the declaration template of the basic syntax of a class that extends another and also implements a number of interfaces.</p> <pre>&lt;accessibility&gt; &lt;class name&gt; extends &lt;superclass&gt; implements &lt;interface<sub>1</sub>&gt;, ..., &lt;interface<sub>n</sub>&gt;</pre> <p>The name of the superclass must come before the names of the interfaces. <i>See the solution to exercise 1.</i></p>
4.	<p>What is the access level of the members of an interface?</p> <p>The access level of the members of an interface is always the access level <code>public</code>.</p>
5.	<p>What kind of class is an interface most like?</p> <p>An interface is most like an abstract class because, similar to an abstract class, the methods in an interface are not required to be implemented (but should at least be specified) and also, whenever an interface is implemented in a concrete class <b>all</b> of the methods of the interface must be implemented just as with an abstract class.</p>
6.	<p>How many interfaces can a given class implement?</p> <p>There is no limit.</p>
7.	<p>What benefits do interfaces present over abstract classes?</p> <p>The main benefit is that a given class can implement <u>multiple</u> interfaces, however it can only inherit from one class at a time, whether that class be abstract or otherwise.</p> <p>Also, there are times when inheritance is not the correct relationship between given classes; for example, a car has as components a steering wheel and a gear box; even if multiple inheritance was possible in Java, a car most definitely is not a descendant class of a steering wheel or gear box, they are merely components of a car with specific utilization parameters as specified by their respective interfaces.</p>
<b>M1</b>	At this time, the reader is invited to review the official class definition of any number of the built-in classes (for

	example, <code>String</code> , <code>List</code> , <code>StringBuffer</code> , <code>ArrayList</code> , etc.), look at the interfaces that are associated therewith and do a web search for the definitions of the interfaces in question.
M2	The reader is also invited to review the official definition of any number of the built-in interfaces (for example <code>Collection</code> , <code>Iteration</code> , etc) and observe the list of the built-in classes that implement the interface(s) in question.
8.	We have defined an interface named <code>IGreeting</code> (see below) that has methods that represent the greeting that should be uttered at specific times of the day. We have also defined a class named <code>Person</code> (see below):
	<pre>public interface IGreeting{     public void morningGreeting();     public void afternoonGreeting();     public void eveningGreeting(); }</pre> <pre>public class Person{     private String firstName;     private String lastName;     public Person(firstName,lastName){         this.firstName = firstName;         this.lastName = lastName;     } }</pre> <p>Create a class named <code>EnglishGreeter</code> which extends the class <code>Person</code> and implements the interface <code>IGreeting</code>.</p> <pre>public class EnglishGreeter extends Person implements IGreeting{     public EnglishGreeter(String firstName, String lastName) {         super(firstName, lastName);     }     public void morningGreeting(){         System.out.println("Good morning!");     }     public void afternoonGreeting(){         System.out.println("Good afternoon!");     }     public void eveningGreeting(){         System.out.println("Good evening!");     } }</pre> <p><b>Note:</b> In this exercise we see an interface being implemented and a class being extended at the same time.</p>
9.	<p>Create the following classes which extend the class <code>Person</code> and implement the interface <code>IGreeting</code>:</p> <ol style="list-style-type: none"> <li>1. A class named <code>FrenchGreeter</code>. This class should print out the greetings in French.</li> <li>2. A class named <code>SpanishGreeter</code>. This class should print out the greetings in Spanish.</li> </ol> <p><i>Note: language translation software/links can be found on the web, for example, <a href="https://translate.google.com">https://translate.google.com</a></i></p> <pre>public class FrenchGreeter extends Person implements IGreeting{     public FrenchGreeter(String firstName, String lastName) {         super(firstName, lastName);     }     public void morningGreeting(){         System.out.println("Bonjour!");     }     public void afternoonGreeting(){         System.out.println("Bon après-midi!");     }     public void eveningGreeting(){         System.out.println("Bonne soirée!");     } }  public class SpanishGreeter extends Person implements IGreeting{     public SpanishGreeter(String firstName, String lastName) {         super(firstName, lastName);     }     public void morningGreeting(){         System.out.println("Buenos días!");     }     public void afternoonGreeting(){ }</pre>

## Practice Your Java Level 1

	<pre>         System.out.println("Buenas tardes!");     }     public void eveningGreeting(){         System.out.println("Buenas noches!");     } } </pre>
10.	<p>In a class named <code>greeting</code>, create an object for each of the greeting classes created previously. The names of the greeters are Matthew Charlis, Jacques Gormand and Hernandez Po; these being the English, French and Spanish greeters respectively. Put the objects that represent them into an <code>ArrayList</code> of type <code>IGreeting</code> and then, using a loop, print out the morning greeting in each language.</p> <pre> import java.util.ArrayList; public class greeting {     public static void main(String args[]){         IGreeting englishGreeter = new EnglishGreeter("Matthew","Charlis");         IGreeting frenchGreeter = new FrenchGreeter("Jacques","Gormand");         IGreeting spanishGreeter = new SpanishGreeter("Hernandez","Po");          ArrayList&lt;IGreeting&gt; greeters = new ArrayList&lt;&gt;();         greeters.add(englishGreeter); greeters.add(frenchGreeter); greeters.add(spanishGreeter);         for(IGreeting m:greeters){             m.morningGreeting();         }     } } </pre> <p><b>Notes:</b> Observe how in this solution we put each of the objects into a variable/reference that was of the type of the interface; this concept was discussed in the solution to the first exercise of this chapter. For this exercise this is just a design choice to show the usage of an interface reference to refer to an object which implements the interface in question; it would have been equally as valid to write the code below:</p> <pre> EnglishGreeter englishGreeter = new EnglishGreeter("Matthew","Charlis"); FrenchGreeter frenchGreeter = new FrenchGreeter("Jacques","Gormand"); SpanishGreeter spanishGreeter = new SpanishGreeter("Hernandez","Po"); </pre> <p>and then do the following:</p> <pre> ArrayList&lt;IGreeting&gt; greeters = new ArrayList&lt;&gt;(); greeters.add((IGreeting)englishGreeter); greeters.add((IGreeting)frenchGreeter); greeters.add((IGreeting)spanishGreeter); </pre> <p>Note the cast to the type <code>IGreeting</code>.</p> <p>From the solution it can also be seen how <u>we can put unrelated objects into a collection whose type is of an interface that all the objects inserted into it implement</u>, thus enabling us to apply a loop to perform common actions on objects which are not related by inheritance.</p>
11.	<p><i>Default interface methods</i></p> <p>If we wanted to implement english as our default greeting language in our interface <code>IGreeting</code>, how would we go about it?</p> <p>We would implement each of the greetings methods in the interface <code>IGreeting</code> as a “default method” using the language english. The ability to put default functionality into interface methods was introduced in Java 8.</p> <p>Note: Default methods in Java interfaces are also called “defender” methods.</p>
12.	<p>Reimplement the interface <code>IGreeting</code> using default methods, with english as the language of choice.</p> <p>Afterwards, reimplement the classes that implement <code>IGreeting</code> to account for the default methods in the interface.</p> <pre> public interface IGreeting {     default public void morningGreeting(){ //Note the keyword <u>default</u> in each of these   //methods         System.out.println("Good morning!");     }     default public void afternoonGreeting(){ } </pre>

	<pre>         System.out.println("Good afternoon!");     }      default public void eveningGreeting(){         System.out.println("Good evening!");     } } </pre> <p>And the modifications to class <code>EnglishGreeter</code>:</p> <pre> public class EnglishGreeter extends Person implements IGreeting {     public EnglishGreeter(String firstName, String lastName) {         super(firstName, lastName);     } } </pre> <p>The classes <code>FrenchGreeter</code> and <code>SpanishGreeter</code> stay the same. They in effect override the default methods in the interface <code>IGreeting</code> that they implement.</p> <p>Observe that since the interface <code>IGreeting</code> already implements by default the functionality that the class <code>EnglishGreeter</code> had implemented, there is no need for the class <code>EnglishGreeter</code> to reimplement the same functionality as the interface <code>IGreeting</code>; the default methods in the interface are sufficient in this case.</p>
13.	<p>Looking at the definition of the interface <code>Comparable</code>, explain the following:</p> <ol style="list-style-type: none"> <li>1. The purpose of the interface</li> <li>2. What is meant by the statement <code>&lt;T&gt;</code> in the definition of the interface</li> </ol> <p>1. The interface <code>Comparable</code> is an interface with a single method named <code>compareTo</code>, whose task it is to compare an instance of an object of a class that implements the method with another object of the same class and return an <code>int</code> indicating which is greater, or smaller, or whether the items in question are equal. As far as Java is concerned, when a class implements <code>Comparable</code>, the ordering that is implemented by the method <code>compareTo</code> of the interface <code>Comparable</code> is deemed to be the “natural ordering” of the objects of that class.</p> <p>2. The <code>&lt;T&gt;</code> in the statement is a placeholder to specify that the interface in question has been designed to support generics (i.e. anything). Therefore, we can use the interface for any reference type that we want; we just specify the actual type in question in place of the placeholder <code>T</code>; for example if we want to implement the interface <code>Comparable</code> for our previously defined class <code>Person</code>, we would declare the class <code>Person</code> as follows:</p> <pre> class Person implements Comparable&lt;Person&gt; </pre> <p>It can be seen that the class name <code>Person</code> takes the place of the placeholder <code>T</code> in the diamond operator in our implementation of <code>Comparable</code> for the class <code>Person</code>.</p>
14.	<p>The method signature for the method <code>compareTo</code> of the interface <code>Comparable</code> is as follows:</p> <pre> int compareTo(T o) </pre> <p>Explain what the components of the parameter list of this method means.</p> <p>The parameter list for this method indicates just one input object type, which is indicated by the letter <code>T</code>; this <code>T</code> is the same <code>T</code> that appears in the definition of the interface (see preceding exercise). Therefore, whatever type we replace <code>T</code> with in our implementation statement in the class definition is the same type that we will replace it with in the methods of the interface that have the same placeholder for a type. Continuing with the example of class <code>Person</code> implementing the interface <code>Comparable</code>, when we implement the interface method <code>compareTo</code> in the class <code>Person</code> we will write the following for the method signature:</p> <pre> int compareTo(Person o) </pre> <p>where clearly the type <code>Person</code> has taken the position of the placeholder <code>T</code>. The letter <code>o</code> is just the variable name of the instance of whatever the type <code>T</code> represents.</p>
15.	<p>The code below does not work. Explain why.</p> <pre> import java.util.ArrayList; import java.util.Collections;  public class Person {     private String firstName;     private String lastName; } </pre>

## Practice Your Java Level 1

```

public Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

public static void main(String[] args){
    Person p1 = new Person("Marie", "Tabith");
    Person p2 = new Person("Ruddy", "Gump");
    Person p3 = new Person("George", "Hudderstein");
    Person p4 = new Person("Peter", "Hudderstein");

    ArrayList<Person> personList = new ArrayList<>();
    personList.add(p1); personList.add(p2); personList.add(p3); personList.add(p4);
    Collections.sort(personList);
    for(Person p:personList){
        System.out.println(p.lastName+ ", " + p.firstName);
    }
}
}

```

The method `Collections.sort` requires that any class whose objects it is called upon to sort must implement the interface `Comparable` in order for it to use the implemented interface method `compareTo`. We have not yet implemented the interface for the class `Person`, that is why the code does not work.

16. Implement the interface `Comparable` for the class `Person`. The order of the objects is to be determined based on the alphabetical order of the fields `lastName` and `firstName`, in the order `lastName` then `firstName`. To test your implementation, put the names listed below into an `ArrayList<Person>` and then apply the method `Collections.sort` to it and print out the contents of the sorted `ArrayList`. The results should be:

*Gump, Ruddy  
Hudderstein, George  
Hudderstein, Peter  
Tabith, Marie*

```

public class Person implements Comparable<Person> {
    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public int compareTo(Person p) {
        if (lastName.compareTo(p.lastName) < 0)
            return(-1);
        if (lastName.compareTo(p.lastName) > 0)
            return (1);
        //otherwise they are equal so compare the first names
        if(firstName.compareTo(p.firstName)< 0)
            return(-1);
        if(firstName.compareTo(p.firstName)> 0)
            return(1);
        else
            return(0);
    }

    public static void main(String[] args){
        //main is as in the preceding exercise
    }
}

```

17. Explain the concepts of a *super-interface* and a *sub-interface*.

In Java it is possible for an interface to “extend” another interface. The interface which is being extended is referred to as the *super-interface*, the interface doing the extending is called a *sub-interface*.

The syntax for extending interfaces is as follows:

*sub-interface extends super-interface<sub>1</sub>, super-interface<sub>2</sub>, ...super-interface<sub>n</sub>*

18.	<p>List the <i>super-interfaces</i> of each of the following interfaces: (1) Queue, (2) ConcurrentLinkedQueue&lt;E&gt;, (3) BlockingQueue, (4) Collection</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 15%;">Queue&lt;E&gt;</td><td>- Collection&lt;E&gt;, Iterable&lt;E&gt;</td></tr> <tr> <td>ConcurrentLinkedQueue&lt;E&gt;</td><td>- Serializable, Iterable&lt;E&gt;, Collection&lt;E&gt;, Queue&lt;E&gt;</td></tr> <tr> <td>BlockingQueue&lt;E&gt;</td><td>- Collection&lt;E&gt;, Iterable&lt;E&gt;, Queue&lt;E&gt;</td></tr> <tr> <td>Collection&lt;E&gt;</td><td>- Iterable&lt;E&gt;</td></tr> </table>	Queue<E>	- Collection<E>, Iterable<E>	ConcurrentLinkedQueue<E>	- Serializable, Iterable<E>, Collection<E>, Queue<E>	BlockingQueue<E>	- Collection<E>, Iterable<E>, Queue<E>	Collection<E>	- Iterable<E>
Queue<E>	- Collection<E>, Iterable<E>								
ConcurrentLinkedQueue<E>	- Serializable, Iterable<E>, Collection<E>, Queue<E>								
BlockingQueue<E>	- Collection<E>, Iterable<E>, Queue<E>								
Collection<E>	- Iterable<E>								
19.	<p>List the <i>sub-interfaces</i> of each of the following interfaces: (1) Queue, (2) ConcurrentLinkedQueue&lt;E&gt;, (3) BlockingQueue</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 15%;">Queue&lt;E&gt;</td><td>- BlockingDeque&lt;E&gt;, BlockingQueue&lt;E&gt;, Deque&lt;E&gt;, TransferQueue&lt;E&gt;</td></tr> <tr> <td>ConcurrentLinkedQueue&lt;E&gt;</td><td>- none</td></tr> <tr> <td>BlockingQueue&lt;E&gt;</td><td>- BlockingDeque&lt;E&gt;, TransferQueue&lt;E&gt;</td></tr> </table>	Queue<E>	- BlockingDeque<E>, BlockingQueue<E>, Deque<E>, TransferQueue<E>	ConcurrentLinkedQueue<E>	- none	BlockingQueue<E>	- BlockingDeque<E>, TransferQueue<E>		
Queue<E>	- BlockingDeque<E>, BlockingQueue<E>, Deque<E>, TransferQueue<E>								
ConcurrentLinkedQueue<E>	- none								
BlockingQueue<E>	- BlockingDeque<E>, TransferQueue<E>								
20.	<p>What does it actually mean when it is said that an interface extends another interface? What are the implications for the implementation of the methods of the interface being extended?</p>								
	<p>What it actually means when an interface extends another is simply that the sub-interface is now seen as the conglomeration of the methods of its super interfaces as well as its own methods. The sub-interface does not implement the methods of its super interfaces. We illustrate further by example:</p> <pre> public interface A{     public void methodA1(...);     public void methodA2(...); }  public interface B extends A{     public void methodB1(...);     public void methodB1(...); } </pre> <p>Now if we have a class X which implements the interface B, the class will look as follows:</p> <pre> class X implements B{     // class X now has to implement all the methods of interface A and interface B     public void methodA1(...){}     public void methodA2(...){}     public void methodB1(...){}     public void methodB1(...){} } </pre>								
21.	<p><i>Interfaces and anonymous classes combined</i></p>								
	<p>MouseListener is an interface. Explain what is going on in the code below where it appears that we are trying to instantiate MouseListener.</p> <pre> MouseListener ml = new MouseListener(){     public void mouseClicked(MouseEvent arg0) {         button01.setText("You clicked me!");     }     public void mouseEntered(MouseEvent arg0) {         button01.setText("You are hovering over me!");     }     public void mouseExited(MouseEvent arg0) {         button01.setText("Phew! That mouse left!");     }     public void mousePressed(MouseEvent arg0) {         button01.setText("Why are you pressing the mouse?");     }     public void mouseReleased(MouseEvent arg0) {         button01.setText("Now you've released it!");     } }; </pre> <p>Recall that we cannot instantiate interfaces, however we can use an interface reference to point to an object that implements that interface. As stated, MouseListener is an interface, therefore when we write new MouseListener() it is not a MouseListener object that we are creating (for there can be none as it is an</p>								

## *Practice Your Java Level 1*

interface), but rather an object of an anonymous class which implements the `MouseListener` interface. Once we write `new <interface name>()`, we are saying that we want an object of an anonymous class that implements the specified interface; we need to implement the interface right there on the spot and that is what you see happening in this code.

# Chapter 27. Collections II: Collections using Interfaces, along with Iterators

In earlier chapters, we looked at collections and interfaces. In this chapter we look at the relationship between both, as Java presents built-in interfaces for each different group of collections that it provides in order to ease their usage. We also look at the usage of the `Iterator` and `ListIterator` interfaces in this chapter.

1.	<p>List the different key Collections interfaces provided in Java.</p> <p>The key Collection interfaces that Java presents are: <code>Set</code>, <code>List</code>, <code>Queue</code>, <code>Deque</code> and <code>Map</code>.</p> <p>It should be noted that the <code>Set</code>, <code>List</code>, <code>Queue</code> and <code>Deque</code> interfaces have as super-interface the interface <code>Collection</code>.</p>
2.	<p>Given an <code>ArrayList&lt;Integer&gt;</code> defined as follows;</p> <pre>ArrayList&lt;Integer&gt; arrlist01 = new ArrayList&lt;&gt;();</pre> <p>explain why all of the following interface assignments are valid:</p> <pre>Serializable ser01          = arrlist01; Cloneable cloneable01       = arrlist01; Iterable&lt;Integer&gt; iter01     = arrlist01; Collection&lt;Integer&gt; collection01 = arrlist01; List&lt;Integer&gt; list01        = arrlist01; RandomAccess ra01           = arrlist01;</pre> <p><i>Hint: Look at the definition of the class <code>ArrayList</code> and note all of its implemented interfaces.</i></p> <p>These interface assignments are all valid because the class <code>ArrayList&lt;T&gt;</code> implements all of these interfaces. As we have seen in the chapter on interfaces, an interface reference variable can be used to refer to any object of any class which implements the interface in question and since <code>ArrayList&lt;T&gt;</code> implements each of these interfaces, the assignments presented above are valid.</p>
3.	<p>Explain why all of the assignments below are valid:</p> <pre>Serializable ser01 = new ArrayList&lt;Integer&gt;(); Serializable ser02 = new TreeMap&lt;String, Integer&gt;(); Serializable ser03 = new LinkedList&lt;Float&gt;();</pre> <p>All of these assignments to interface reference variables of type <code>Serializable</code> in this case are valid because each of the object types to which the respective <code>Serializable</code> reference variables in the question refer implement the interface <code>Serializable</code>.</p>
4.	<p>Given the following declaration:</p> <pre>Serializable ser01 = new ArrayList&lt;Integer&gt;();</pre> <p>Write a program which shows the insertion of the value 27 into the <code>ArrayList&lt;Integer&gt;</code> instance referenced by the interface reference <code>ser01</code>. Print the contents of the <code>ArrayList&lt;Integer&gt;</code> in question out.</p> <pre>import java.io.Serializable; import java.util.ArrayList;  public class InterfacesAndCollections {     public static void main(String[] args){         Serializable ser01 = new ArrayList&lt;Integer&gt;();         ((ArrayList&lt;Integer&gt;)ser01).add(27);         // Observe how we applied the cast ArrayList&lt;Integer&gt; to <u>ser01</u> in order to "see"         // the actual object ser01 referenced as it is indeed, an ArrayList&lt;Integer&gt;.         // When we apply the cast we can then apply the methods appropriate for         // the "casted" type to it, such as the method ArrayList&lt;T&gt;.add that we         // applied here.          for( Integer j: (ArrayList&lt;Integer&gt;)ser01 ) {</pre>

## Practice Your Java Level 1

```
//Again, we cast ser01 in the same way as previously done in order  
//to “see” the object it references as an ArrayList<Integer>.  
    System.out.println(j); // We print the variable out here  
}  
}  
}
```

An alternate solution to applying a cast to `ser01` everywhere we need it would be to actually declare an `ArrayList<Integer>` variable as follows:

```
ArrayList<Integer> arrayList01 = (ArrayList<Integer>)ser01;
```

And then we would use the variable `arrayList01` in the subsequent places where we had applied a cast to `ser01`.

5. In our earlier chapter on Collections, we would access an `ArrayList` instance directly using a reference variable of its very type as follows:

```
ArrayList<String> StringList01 = new ArrayList<String>();
```

Explain the benefits of referring to an `ArrayList` object using a reference variable of its primary interface, `List` as in the following example:

```
List<String> StringList01 = new ArrayList<String>();
```

On observation, it is noted that many of the key methods of class `ArrayList` are actually the implemented methods of the interface `List`, therefore there is no loss of generality in using an object reference of type interface `List`. Beyond this, one benefit of using a `List<String>` variable is that if we decide to change the particular collection class type that it references to an object of another collection type that implements the interface `List`, we would only have to make the change at the point of initial instantiation of the object, rather than having to make modifications throughout our code. Now why would we potentially want to change the object to a different type (for example from an `ArrayList` to a `LinkedList` which also implements the interface `List`)? We might want to make such a change for performance reasons. The grouping in Java of different collection types by their primary interface facilitates this easy modification. It should also be noted that one key benefit of interfaces is the separation as far as possible of the concept of how the data is accessed from the implementation of the storage of the data.

### Iterators

6. What is an iterator?

An iterator is an object that facilitates the accessing in a sequential manner (sequential manner meaning whatever the type in question that has implemented the interface `Iterator` deems internally to be a sequential manner) the elements of a collection.

7. Predict the output of the following code and explain the mechanism being utilized for the output:

```
import java.util.Iterator;  
import java.util.LinkedList;  
import java.util.List;  
  
public class InterfacesAndCollections {  
    public static void main(String[] args){  
        List<Integer> list01 = new LinkedList<>();  
        list01.add(1);list01.add(3);list01.add(5);list01.add(7);list01.add(9);  
        1:   Iterator<Integer> it01 = list01.iterator();  
        2:   while(it01.hasNext()){  
        3:       Integer j = it01.next();  
        4:       System.out.print(j + " ");  
        5:   }  
    }  
}
```

**Output:** 1 3 5 7 9

The mechanism being utilized for output is called an `Iterator`. In the code line labeled 1 above, we see an `Iterator<Integer>` reference being assigned an object known as an “iterator object” from the `List<T>` instance in question.

The `Iterator` object is one that facilitates the accessing in a sequential manner (sequential manner meaning

	<p>whatever the type in question that has implemented the interface <code>Iterator</code> deems internally to be a sequential manner) the elements of a collection.</p> <p>We see the use of the <code>Iterator</code> object in the lines labeled 2 to 5. We see the use of two of its methods, in particular the method <code>hasNext()</code>, which is used to determine whether there are any more elements in the Collection object through whose list of elements we are iterating and then we use the method <code>next()</code> which is both used to obtain the next element and also to advance the internal “pointer” of the <code>Iterator</code> object to the next element (if there is one).</p> <p>All <code>Collections</code> classes implement the <code>Iterator</code> interface.</p> <p>If not already done, the reader is invited to study the official documentation for the <code>Iterator</code> interface.</p>
8.	<p>Enhance the program in the preceding exercise to, using an <code>Iterator</code>, remove the number 5 from the <code>List&lt;Integer&gt;</code> presented. After the removal, print out the contents of the <code>List&lt;Integer&gt;</code> using an <code>Iterator</code> object.</p> <pre>public class InterfacesAndCollections {     public static void main(String[] args){         List&lt;Integer&gt; list01 = new LinkedList&lt;&gt;();         list01.add(1);list01.add(3);list01.add(5);list01.add(7);list01.add(9);          Iterator&lt;Integer&gt; it01 = list01.iterator();         while(it01.hasNext()){             Integer j = it01.next();             if(j == 5) it01.remove();         }         // Now printing the list out using an iterator         Iterator&lt;Integer&gt; it02 = list01.iterator();         while(it02.hasNext()){             Integer j = it02.next();             System.out.print(j + " ");         }     } }</pre>
9.	<p>Given a <code>List&lt;Integer&gt;</code> with entries 1, 3, 5, 7, 9 and 11, using an <code>Iterator</code>, add the value 1 to each of the entries and then, still using an <code>Iterator</code>, print out the contents of the <code>List&lt;Integer&gt;</code>.</p> <pre>public class InterfacesAndCollections {     public static void main(String[] args){         List&lt;Integer&gt; list01 = new LinkedList&lt;&gt;();         list01.add(1);list01.add(3);list01.add(5);list01.add(7);list01.add(9);          Iterator&lt;Integer&gt; it01 = list01.iterator();         while(it01.hasNext()){             Integer j = it01.next();             it01.set(j+1);         }          it01 = list01.iterator();      // NOTE: We have to acquire a new Iterator, because                                      // the last one used is now at the end of the List.         while(it01.hasNext()){             System.out.print(it01.next() + " ");         }     } }</pre>
<b>ListIterator</b>	
10.	<p>Compare and contrast the <code>Iterator</code> and the <code>ListIterator</code> interfaces.</p> <p>The <code>ListIterator</code> is a sub-interface of the <code>Iterator</code> interface and thus inherits all of its methods. However, the <code>ListIterator</code> goes beyond the capability of the <code>Iterator</code> interface, adding the ability to walk backwards through a Collection by the use of its <code>previous()</code> method. The <code>ListIterator</code> can also report the index of the entries. Also, while the <code>Iterator</code> can only remove elements, the <code>ListIterator</code> can also add or replace elements using its <code>add()</code> and <code>set()</code> methods.</p>

## Practice Your Java Level 1

	<p><code>ListIterator</code> is implemented in the classes which implement the interface <code>List</code>.</p>
11.	<p>I have a <code>LinkedList&lt;Integer&gt;</code> named <code>list01</code> with the values 1, 3, 5, 7, 9, 10 and 11. Write a program which uses a <code>ListIterator</code> to loop through the list and print its contents out.</p> <pre>public class InterfacesAndCollections {     public static void main(String[] args){         List&lt;Integer&gt; list01 = new LinkedList&lt;&gt;();         list01.add(1);list01.add(3);list01.add(5);list01.add(7);list01.add(9);         list01.add(10);list01.add(11);          // Now printing the list out using an iterator         ListIterator&lt;Integer&gt; it01 = list01.listIterator();         while(it01.hasNext()){             Integer j = it01.next();             System.out.print(j + " ");         }     } }</pre>
12.	<p>I have a <code>LinkedList&lt;Integer&gt;</code> named <code>list01</code> with the values 1, 3, 5, 7, 9, 10 and 11. Write a program which uses a <code>ListIterator</code> to loop <i>backwards</i> through the list and print its contents out.</p> <p><i>Hint: See method <code>List&lt;T&gt;.ListIterator(int index)</code></i></p> <pre>public class InterfacesAndCollections {     public static void main(String[] args){         List&lt;Integer&gt; list01 = new LinkedList&lt;&gt;();         list01.add(1);list01.add(3);list01.add(5);list01.add(7);list01.add(9);         list01.add(10); list01.add(11);          // Now printing the list out backwards using an iterator         // Place the listIterator at the last position of the List         ListIterator&lt;Integer&gt; it01 = list01.listIterator(list01.size());         while(it01.hasPrevious()){             Integer j = it01.previous();             System.out.print(j + " ");         }     } }</pre>
13.	<p>Repeat exercise 9, this time using a <code>ListIterator</code>. Go through the <code>List&lt;Integer&gt;</code> only once to do both the addition and the printing out of its contents.</p> <pre>public class InterfacesAndCollections {     public static void main(String[] args){         List&lt;Integer&gt; list01 = new LinkedList&lt;&gt;();         list01.add(1);list01.add(3);list01.add(5);list01.add(7);list01.add(9);         ListIterator&lt;Integer&gt; it01 = list01.listIterator();          while(it01.hasNext()){             Integer j = it01.next();             it01.set(j+1);             //Alright, now go back one and come back to print it out             if(it01.hasPrevious()){                 System.out.print(it01.previous() + " ");                 it01.next(); //we need to move forward 1 since we moved back 1             }         }     } }</pre>
14.	<p>Compare and contrast the function of the <code>Iterator</code> and <code>ListIterator</code> vis à vis an enhanced <code>for</code> loop with respect to classes which implement the <code>List&lt;T&gt;</code> interface.</p> <p>Actually, the enhanced <code>for</code> loop uses the <code>Iterator</code> functionality! In that sense, the enhanced <code>for</code> loop is just a cover for the <code>Iterator</code>. One advantage the <code>Iterator</code> has over the enhanced <code>for</code> loop is that it supports the</p>

	removal of elements via its <code>remove()</code> method. However, with respect to the <code>ListIterator</code> , in addition to the functionality of the <code>Iterator</code> , the <code>ListIterator</code> also has the ability to do the following for objects of classes which implement it: (1) add elements, (2) return element indexes (3) replace elements and also to (4) go backwards through the elements.
15.	<p><i>Inserting into a Collection using a ListIterator</i></p> <p>I have a <code>LinkedList&lt;Integer&gt;</code> which starts out being filled with only odd numbers. Using a <code>ListIterator</code>, go through the <code>LinkedList&lt;Integer&gt;</code> and insert after each odd number therein the even number just after it. Print the resulting <code>LinkedList&lt;Integer&gt;</code> out.</p> <p>For example if the contents of the <code>LinkedList&lt;Integer&gt;</code> are <code>{1,3,15,7}</code>, then after the insertions the content will end up being <code>{1,2,3,4,15,16,7,8}</code>.</p> <pre>import java.util.LinkedList; import java.util.List; import java.util.ListIterator;  public class InterfacesAndCollections {     public static void main(String[] args){         List&lt;Integer&gt; list01 = new LinkedList&lt;&gt;();         list01.add(1);list01.add(3);list01.add(15);list01.add(7);          ListIterator&lt;Integer&gt; it01 = list01.listIterator();          while(it01.hasNext()){             Integer j = it01.next();             if(j % 2 == 1){                 it01.add(j+1);             }         }         ListIterator&lt;Integer&gt; it02 = list01.listIterator();         while(it02.hasNext()){             Integer j = it02.next();             System.out.print(j + " ");         }     } }</pre>
E1	I have a <code>List&lt;Integer&gt;</code> with <u>at least</u> two different numbers in it, in ascending order. Write a program which, using a <code>ListIterator</code> , will insert all of the missing numbers in-between the numbers that appear in the <code>List&lt;Integer&gt;</code> . For example, if the original contents of the <code>List&lt;Integer&gt;</code> are <code>{1, 5, 8}</code> , after your program has run its contents should be <code>{1,2,3,4,5,6,7,8}</code> .



# Chapter 28. Properties, Computer Environment & Computer Information

This chapter presents exercises with which to practice and enhance your knowledge on how to use built-in Java methods to determine information about your computer. Exercises on such topics as the concept of properties, determination of available JVM runtime status, available processor count, drive and filesystem information, operating system version, computer name, system uptime and other such related topics are presented in this chapter.

The key classes utilized in this chapter include `Runtime` and `System`. Some of the exercises also call for the use Java Beans to get system information.

1.	<p>What is the concept of <i>properties</i> in Java?</p> <p>The concept of properties in Java is a simple one. It simply refers to specific named environmental constants that have been assigned values. Properties are accessible whenever a JVM is running. Properties exist for your program to use as you see fit. Note that the value assigned to a property can also be modified.</p> <p>The JVM obtains these properties in at least the following ways:</p> <ul style="list-style-type: none"><li>• Obtaining them from the underlying operating system</li><li>• Obtaining them from the command line when the JVM is started (i.e. you the user can specify values for properties on the command line)</li><li>• New properties can be created directly from within your program</li></ul> <p>The system properties pertain largely to the specification of the hardware that the JVM is being run on, the operating system, the user profile that the JVM is being run on and the JVM itself.</p> <p>You the programmer can also create properties from within your program.</p> <p>Run the program below to obtain a list of the current set of properties and their values for your JVM.</p> <pre>import java.util.Properties; public class PracticeYourJava {     public static void main(String[] args) {         Properties p = System.getProperties();         p.list(System.out);     } }</pre>
2.	<p><i>Java installation and configuration information</i></p> <p>Using the <code>System.getProperty</code> method, print out the following information: (1) the Java version that you are currently running, (2) the directory in which the Java runtime is installed and (3) the classpath of the JAR files.</p> <p><i>Hint: properties <code>java.version</code>, <code>java.home</code> and <code>java.class.path</code></i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Java version: " + System.getProperty("java.version"));         System.out.println("JRE installation directory: " + System.getProperty("java.home"));         System.out.println("Java CLASSPATH: " + System.getProperty("java.class.path"));     } }</pre>
3.	<p><i>Java runtime information</i></p> <p>The value <code>&lt;Runtime&gt;.MaxMemory()</code> defines the maximum amount of RAM that the JVM is permitted to request. Explain how to interpret the value returned from this method.</p> <p>The value in question is to be interpreted as follows: If the value returned = <code>Long.MAX_VALUE</code>, then there is no actual constraint on how much memory the JVM can request. If it is any other value then that value is the</p>

## Practice Your Java Level 1

	maximum amount of memory that the JVM can request.
4.	Using an instance of the class <code>Runtime</code> returned from the method <code>Runtime.getRuntime()</code> , print out the following information: (1) the amount of RAM that is currently available to the JVM (2) the amount of RAM used by the JVM and (3) the maximum amount of RAM that the JVM is allowed to request (keeping in mind the solution to the preceding exercise); if there is no limit, then state so. <pre>public class PracticeYourJava {     public static void main(String[] args) {         Runtime rt = Runtime.getRuntime();         System.out.println("Currently available RAM to JVM: " + rt.totalMemory() + " bytes");         long usedMem = rt.totalMemory() - rt.freeMemory();         System.out.println("RAM currently utilized by the JVM: " + usedMem + " bytes");         long maxRequestable = rt.maxMemory();         if(maxRequestable != Long.MAX_VALUE)             System.out.println("Max requestable memory: " + maxRequestable + " bytes");         else             System.out.println("No maximum enforced for requestable memory ");     } }</pre>
5.	Print out the working directory that the JVM is currently using. <i>Hint: property user.dir</i> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("JVM working directory: " + System.getProperty("user.dir"));     } }</pre>
6.	<i>JVM Vendor information</i> Print out the following information: (1) the JVM vendor name and (2) the vendor URL. <i>Hint: properties java.vendor, java.vendor.url</i> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Java vendor: " + System.getProperty("java.vendor"));         System.out.println("JRE vendor URL: " + System.getProperty("java.vendor.url"));     } }</pre>
7.	<i>Operating System Information</i> Print out the following: (1) the name of the operating system that your JVM is currently running on, (2) the operating system architecture and (3) the internal denotation of the operating system version. <i>Hint: properties os.name, os.arch, os.version</i> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Operating System (OS): " + System.getProperty("os.name"));         System.out.println("OS Architecture: " + System.getProperty("os.arch"));         System.out.println("OS version (internal): " + System.getProperty("os.version"));     } }</pre>
8.	<i>Computer Information</i> Using a <code>Runtime</code> object, print out how many processors are available to the JVM for use. <i>Hint: &lt;Runtime&gt;.availableProcessors()</i> <pre>public class PracticeYourJava {     public static void main(String[] args) {         Runtime rt = Runtime.getRuntime();         System.out.println("Number of processors available to the Java VM = " +                            rt.availableProcessors());     } }</pre>

	Note: If the processor is hyperthreaded, each CPU core may be presented to the operating system as a separate processor.
9.	<p><i>User information</i></p> <p>Write a program which will determine the user account name that your JVM is running with.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("User name: " + System.getProperty("user.name"));     } }</pre> <p><b>IMPORTANT NOTE:</b> This method of determining the user name is <u>not preferred at all</u>, because properties can be specified on the command line when starting a Java program and can thus be purposely spoofed. A better method for obtaining the user name is to check using a security authentication module. That topic is beyond the scope of this volume.</p>
10.	<p>Print out the home directory of the user.</p> <p><i>Hint: property user.home</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("User home directory: " + System.getProperty("user.home"));     } }</pre> <p><b>IMPORTANT NOTE:</b> See the note in the solution to the previous question as it applies to this property too.</p>
11.	<p>Using the <code>getHomeDirectory</code> method of a <code>FileSystemView</code> object obtained via the method <code>FileSystemView.getFileSystemView()</code>, write a program which will determine and print out the <i>Desktop</i> directory of the user account that the JVM is running with.</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         FileSystemView fileSys = FileSystemView.getFileSystemView();         String homeDir = fileSys.getHomeDirectory().toString();         System.out.println("The home directory is " + homeDir);     } }</pre>
12.	<p><i>Path Separators</i></p> <p>The directory path separator on Windows is the forward slash, “\”, on UNIX it is the backwards slash “/”. Now given that your Java program should generally be able to run on either operating system, write a program which obtains and prints the file separator for the system on which you are running.</p> <p><i>Hint: property file.separator</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("File separator : " + System.getProperty("file.separator"));     } }</pre>
13.	<p>Write a program which prints out the path separator used on your system.</p> <p>(The path separator is the character used for concatenating a set of paths, for example for your Java CLASSPATH variable).</p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("Path separator : " + System.getProperty("path.separator"));     } }</pre>
14.	<p><i>Using beans to find system information: The operating System Bean</i></p> <p>Using an interface reference of <code>OperatingSystemMXBean</code> to an object returned by the <code>ManagementFactory</code> static method <code>ManagementFactory.getOperatingSystemBean</code>, determine and print out the following information:</p>

## Practice Your Java Level 1

	<ul style="list-style-type: none"> <li>The processor architecture</li> <li>Number of available processors</li> <li>Operating System Name</li> <li>Operating System version</li> </ul> <pre>import java.lang.management.OperatingSystemMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         OperatingSystemMXBean osmxbean = ManagementFactory.getOperatingSystemMXBean();         System.out.println("The system architecture = " + osmxbean.getArch());         System.out.println("#Available processors = " + osmxbean.getAvailableProcessors());         System.out.println("Operating System(OS) Name = " + osmxbean.getName());         System.out.println("OS Version = " + osmxbean.getVersion());     } }</pre> <p><b>Note:</b> Recall that we can also determine the number of processors on a system using the method <code>&lt;Runtime&gt;.availableProcessors</code>, as previously seen in exercise 8.</p>
15.	Using the same object as in the preceding exercise, determine the <i>average system load</i> in the last minute, if this value is supported by the operating system that you are running on.
	<pre>import java.lang.management.OperatingSystemMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         OperatingSystemMXBean osmxbean = ManagementFactory.getOperatingSystemMXBean();         float loadAvg = (float)osmxbean.getSystemLoadAverage();         if(loadAvg &lt; 0)             System.out.println("Load Avg reporting is not supported");         else             System.out.println("Avg. system load in last minute = " + osmxbean.getSystemLoadAverage());     } }</pre>
16.	Using an interface reference variable of type <code>RuntimeMXBean</code> to an object returned from the static method <code>ManagementFactory.getRuntimeMXBean()</code> , determine and print out the following information: <ul style="list-style-type: none"> <li>The JVM vendor</li> <li>The JVM specification version</li> <li>The JVM implementation version</li> </ul>
	<pre>import java.lang.management.ManagementFactory; import java.lang.management.RuntimeMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         RuntimeMXBean rtmxbean = ManagementFactory.getRuntimeMXBean();         System.out.println("JVM vendor name = " + rtmxbean.getSpecVendor());         System.out.println("JVM spec version = " + rtmxbean.getSpecVersion());         System.out.println("JVM implementation version = " + rtmxbean.getVmVersion());     } }</pre> <p><b>Note:</b> The information printed out above actually corresponds to the properties <code>java.vm.vendor</code>, <code>java.specification.version</code> and <code>java.vm.version</code>.</p>
17.	Using the same object as in the previous exercise, determine the following: <ul style="list-style-type: none"> <li>The CLASSPATH</li> <li>The Boot ClassPath of the JVM</li> <li>The Java library path</li> </ul>
	<pre>import java.lang.management.RuntimeMXBean; import java.util.*;</pre>

	<pre>public class PracticeYourJava {     public static void main(String[] args) {         RuntimeMXBean rtmxbean = ManagementFactory.getRuntimeMXBean();         System.out.println("ClassPath           = " + rtmxbean.getClassPath());         System.out.println("Boot ClassPath     = " + rtmxbean.getBootClassPath());         System.out.println("Java Library Path = " + rtmxbean.getLibraryPath());     } }</pre>
18.	Using the same object as in the previous exercise, determine the following: the instance name of the JVM on which the program is running (you can have many JVMs running on the same computer). <pre>import java.lang.management.RuntimeMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         RuntimeMXBean rtmxbean = ManagementFactory.getRuntimeMXBean();         System.out.println("JVM instance name = " + rtmxbean.getName());     } }</pre>
19.	Using the same object as in the previous exercise, print out the uptime of the JVM itself in milliseconds. <pre>import java.lang.management.RuntimeMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         RuntimeMXBean rtmxbean = ManagementFactory.getRuntimeMXBean();         System.out.println("The JVM has been up for: " + rtmxbean.getUptime() + "milliseconds.");     } }</pre>
20.	Using the same object as in the previous exercise, print out the command line arguments that pertain to the JVM itself. <pre>import java.lang.management.RuntimeMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         RuntimeMXBean rtmxbean = ManagementFactory.getRuntimeMXBean();         System.out.println("The JVM arguments are: " + rtmxbean.getInputArguments());     } }</pre>
21.	Write a program which, using an operating system bean returned from the static method <code>ManagementFactory.getOperatingSystemMXBean</code> , obtains the size of the computer memory and prints it out. <pre>import java.lang.management.ManagementFactory; import com.sun.management.OperatingSystemMXBean;  public class PracticeYourJava {     public static void main(String[] args) {         OperatingSystemMXBean osbean =             (com.sun.management.OperatingSystemMXBean)ManagementFactory.getOperatingSystemMXBean();         long memSize = osbean.getTotalPhysicalMemorySize();         System.out.println("Computer memory = " + memSize + " bytes");     } }</pre>
22.	<p><i>Updating/Modifying a Property</i></p> <p>Using the method <code>System.setProperty</code>, update the property <code>user.country</code> to a different country. To ensure that it was modified, print the property out after you set it.</p> <p><i>Hint: System.setProperty</i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {</pre>

## Practice Your Java Level 1

	<pre>        System.setProperty("user.country", "UK"); //We'll use the UK in our example here         System.out.println(System.getProperty("user.country"));     } }</pre>
23.	<p><i>Creating a new System property</i></p> <p>Create a new property named <code>user.greeting</code>. Assign it a value of choice. To ensure that it was created, print the value of the property out after you create and set it.</p> <p><i>Hint: <code>System.setProperty</code></i></p> <pre>public class PracticeYourJava {     public static void main(String[] args) {         System.setProperty("user.greeting", "Hello!!!");         System.out.println(System.getProperty("user.greeting"));     } }</pre>
<b>E1</b>	Write a program which determines the system uptime in days, hours, minutes and seconds.
<b>E2</b>	Write a program which obtains the hostname of the computer you are running on.

# Chapter 29. File System I: Path, Directory and File Handling using NIO/NIO.2

This is a key chapter in confirming and enhancing your skills with and knowledge in computer filesystem handling using Java. Among the exercises in this chapter are exercises on directory creation, directory path handling, symbolic links, file archives (zip files), the manipulation (moving, renaming, deleting, etc.) of the same as well as the manipulation files.

The title of this chapter states that the filesystem exercises in this chapter use NIO/NIO.2 (which stands for New Input Output version 2). Prior to NIO, Java filesystem handling was handled by the functionality in the `java.io` package. In version 1.4 of Java, NIO was introduced and in version 7 was further enhanced and at present is called NIO.2. NIO/NIO.2 do provide a number of enhancements over the `java.io` package. While indeed the `java.io` package is still available in Java and is still used by many, it is advised that Java programmers familiarize themselves with the new package which has been present in Java for a number of versions now. With that in mind, this chapter has been written to exercise and enhance your filesystem handling skills using the latest package provided in Java for such.

NIO/NIO.2 present quite a number of classes and enumerations. In this chapter, you will exercise and enhance your skills in using them. The exercises in this chapter employ the use of the NIO classes `Path`, `Paths`, `File`, `FileSystem`, `FileStore`, `FileTime`, the interfaces `DirectoryStream`, `UserPrincipal` and `BasicFileAttributes`, along with the exception handling that Java mandates for filesystem operations. Various Java concepts, including the use of interfaces, generics and anonymous classes are necessary tools in solving some of the exercises presented in this chapter.

1.	<p>In Java NIO, what does a <code>Path</code> object represent?</p> <p>A <code>Path</code> object in Java NIO generally represents a directory path to either a directory, a file or a symbolic link. In such scenarios, the last element contained in a given <code>Path</code> object is the name of the directory, file or symbolic link that the path refers to.</p> <p>In addition to representing absolute paths, a <code>Path</code> object can represent a relative path, or even just part of a path that you might want to combine with other paths.</p> <p>We summarize the answer given above as follows “A <code>Path</code> object can represent the full path to a directory, a file, a symbolic link or even just part of a directory path”.</p>
2.	<p>Write a program that obtains and prints out the path of the default working directory that Java uses when running your programs.</p> <p><i>Hint: Property user.dir</i></p> <pre>import java.util.*; public class PracticeYourJava {     public static void main(String[] args) {         System.out.println("JVM working directory: " + System.getProperty("user.dir"));     } }</pre>
3.	<p>Explain in general terms the relationship of the class <code>Path</code> to the class <code>Paths</code>.</p> <p>As described earlier, the <code>Path</code> class can represent the full path to a directory, a file, a symbolic link, or even just part of a directory path. The function of the <code>Paths</code> class is to form a <code>Path</code> object out of strings that denote the individual elements of a path. For example, on a Windows system, the path to a given file might be defined as: <code>\Users\PYJ\Desktop\abc.txt</code> whereas, on a UNIX derived system the path to the file in question might be defined as <code>/Users/PYJ/Desktop/abc.txt</code>. When passed the individual elements <code>Users</code>, <code>PYJ</code>, <code>Desktop</code>, <code>abc.txt</code> of the path in sequence, <code>Paths</code> knows how to concatenate them appropriately for the given system that it is on (it pays attention to the property <code>path.separator</code>) to form a <code>Path</code> object.</p> <p>The <code>Paths</code> class is also used to create Uniform Resource Indicators (URI's) in the same way. (Exercises on URI's</p>

## Practice Your Java Level 1

	<p>appear later in this chapter).</p> <p>In summary, we can say that the <code>Paths</code> class is really just a helper class for the class <code>Path</code> to help form “paths”. The <code>Paths</code> class consists only of static methods with which to perform its tasks.</p>
4.	<p><i>Obtaining Path objects using the Paths class and system properties</i></p> <p>Write a program which obtains a <code>Path</code> object that represents the users’ home directory. Print the <code>Path</code> object out to the console.</p> <p><i>Hint: Obtain the desired Path object, using <code>Paths.get</code> applied to the property <code>user.home</code>.</i></p> <pre>import java.nio.file.*; public class PracticeYourJava {     public static void main(String[] args) {         Path homeDir = Paths.get(System.getProperty("user.home"));         System.out.println("The home directory is " + homeDir);     } }</pre>
5.	<p>Write a program which obtains a <code>Path</code> object that represents the users’ <code>Desktop</code> directory. Print the <code>Path</code> object out to the console.</p> <p><i>Hint: Obtain the desired Path object, using <code>Paths.get</code> applied to the property <code>user.home</code> and the string “<code>Desktop</code>”. Also, note that <code>Paths.get</code> takes a variable number of arguments.</i></p> <pre>import java.nio.file.*; public class PracticeYourJava {     public static void main(String[] args) {         Path desktop = Paths.get(System.getProperty("user.home"), "Desktop");         System.out.println("The directory path of the Desktop is " + desktop);     } }</pre>
6.	<p>Set the property that represents the current user working directory to the <code>Desktop</code>. Afterwards, print the path of the current working directory out to confirm that it has indeed been modified.</p> <p><i>Hint: <code>System.setProperty</code>, property <code>user.dir</code></i></p> <pre>import java.nio.file.*; public class PracticeYourJava {     public static void main(String[] args) {         Path desktopDir = Paths.get(System.getProperty("user.home"), "Desktop");         System.setProperty("user.dir", desktopDir);         System.out.println("JVM working directory is now : " + System.getProperty("user.dir"));     } }</pre>
7.	<p>Write a program which determines whether a directory named <code>Dir_A</code> exists on the <code>Desktop</code> and inform the user as to your findings.</p> <p><i>Hint: Obtain a <code>Path</code> object to the target directory. Then check its existence with the method <code>Files.exists</code></i></p> <pre>import java.nio.file.*; public class PracticeYourJava {     public static void main(String[] args) {         String targetDir = "Dir_A";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", targetDir);         if (Files.exists(targetPath))             System.out.printf("The directory %s exists on the desktop", targetDir);         else             System.out.printf("The directory %s does not exist on the desktop", targetDir);     } }</pre>
8.	<p>Create each of the following directories on the <code>Desktop</code>: <code>Dir_A</code>, <code>Dir_B</code>, <code>Dir_C</code>, <code>Dir_D</code> and <code>Dir_E</code>, after first confirming that they do not already exist.</p> <p><i>Hint: Using a loop, create a <code>Path</code> object for each of the listed directories (put the directory names into a <code>String[]</code>) and then</i></p>

	<p>use <code>Files.createDirectory</code> to do the actual creation.</p> <pre>import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         String[] dirNames = new String[]{"Dir_A","Dir_B","Dir_C","Dir_D","Dir_E"};         for(int i=0;i &lt; dirNames.length; i++){             Path targetDir = Paths.get(System.getProperty("user.home"), "Desktop", dirNames[i]);             if (Files.exists(targetDir))                 System.out.printf("The directory %s already exists on the desktop\n", dirNames[i]);             else{                 try{                     Files.createDirectory(targetDir);                     System.out.printf("The directory %s has been created on the desktop\n", dirNames[i]);                 }                 catch(IOException e){                     System.out.println("Couldn't create directory " + dirNames[i]);                 }             }//end else         }//end for     } }</pre>
9.	<p>Write a program which will create the subdirectories Dir_A_1, Dir_A_2, Dir_A_3, Dir_A_4 and Dir_A_5 inside the earlier created Dir_A.</p> <p><i>Hint: This exercise is very similar to the preceding exercise.</i></p> <pre>import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         String baseDir = "Dir_A";         String[] dirNames = new String[]{"Dir_A_1","Dir_A_2","Dir_A_3","Dir_A_4","Dir_A_5"};         for(int i=0;i &lt; dirNames.length; i++){             Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", baseDir, dirNames[i]);             if (Files.exists(targetPath))                 System.out.printf("The directory %s already exists \n", targetPath);             else{                 try{                     Files.createDirectory(targetPath);                     System.out.printf("The directory %s has been created \n", targetPath);                 }                 catch(IOException e){                     System.out.println("Couldn't create directory " + targetPath);                 }             }//end else         }//end for     } }</pre>
10.	<p><i>Creating nested directories at one go</i></p> <p>Create the following directory tree on the Desktop: A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/Y/Z</p> <p><i>Hint: <code>Files.createDirectories</code>. Also ensure that you replace the “/” with the appropriate path separator for your system (<code>property file.separator</code>).</i></p> <pre>import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {      public static void main(String[] args) {         String targetDir = "A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/Y/Z";         //replace the backslash with the appropriate path separator for the system you are on     } }</pre>

## Practice Your Java Level 1

	<pre>//The path separator is in the property file.separator targetDir = targetDir.replace("/", System.getProperty("file.separator")); //Now create the Path Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", targetDir);  if (Files.exists(targetPath))     System.out.printf("The directory tree %s already exists on the desktop", targetDir); else{     try{         Files.createDirectories(targetPath);     }     catch(IOException e){         System.out.println("Couldn't create directory " + targetPath);     }     System.out.printf("The directory tree %s has been created on the desktop", targetPath); } }</pre>
11.	<p>Delete the earlier created empty subdirectory <code>Dir_A_5</code>. In your <code>IOException</code> handler, print out the stack trace of the potential <code>IOException</code> exceptions that may result. <i>Hint: <code>Files.deleteIfExists</code></i></p> <pre>import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         String targetDir = "Dir_A_5";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", "Dir_A", targetDir);          try{             boolean result = (Files.deleteIfExists(targetPath));             if(result==true) //can also be written as if(result)                 System.out.printf("The directory %s has been deleted.\n", targetPath);             else                 System.out.printf("The directory %s was non-existent.\n", targetPath);         }         catch(IOException e){             System.out.printf("Exception; could not delete the directory " + e.toString());             e.printStackTrace();         }     } }</pre>
12.	<p>Manually create a text file in the previously created directory <code>Dir_A_4</code>. Then modify the code from the preceding exercise to attempt to delete the directory <code>Dir_A_4</code>. Comment on the output of the program. An exception, <code>DirectoryNotEmptyException</code>, occurred and was caught by its super exception <code>IOException</code>. (If we want, we can handle this exception separately in its own <code>catch</code> block).</p>
13.	<p>What is the function of the <code>DirectoryStream</code> interface? The <code>DirectoryStream</code> interface is one that facilitates the reading of the names and information about all of the contents of a specified directory. A <code>DirectoryStream</code> is opened using the method <code>Files.newDirectoryStream</code> against a given <code>Path</code> object. The resulting <code>DirectoryStream</code> object can then be iterated through to analyze each of the items that are returned. It is also possible to specify a <code>DirectoryStream</code> filter (<code>DirectoryStream.Filter</code>) if we want only specific subsets of the directory contents; for example, you might want to list only the files with an extension of ".txt" (specified as <code>*.txt</code>), or another example might be all files that both have an extension of ".jpg" and have only three letters in their name before the file extension (specified as <code>???.jpg</code>). Note: the word "Stream" here is not to be confused with the Java 8 concept of Streams; think of stream here to mean "pouring forth the items items in the directory as if a stream is flowing".</p>
14.	The code below is using a <code>DirectoryStream</code> object to obtain and print out a listing of all the files & directories in

the *Desktop* directory. Describe briefly what the code is doing.

```
import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {
        Path desktopPath = Paths.get(System.getProperty("user.home"), "Desktop");
        try{
            DirectoryStream<Path> dirStream = Files.newDirectoryStream(desktopPath);
            for (Path item: dirStream) {
                System.out.println(item);
            }
            dirStream.close(); //make sure you close the stream
        }
        catch(IOException e){}
    }
}
```

**Explanation:** The code starts off with obtaining a `Path` object for the *Desktop* directory.

We then instantiate a `DirectoryStream<Path>` reference to a `DirectoryStream` object. This object is initialized and populated by the method `Files.newDirectoryStream` which is passed the `Path` object for the target directory whose contents we want to list, the directory in this case being the *Desktop* directory.

The object having been populated, we simply loop through it and print out its contents.

Later we will see exercises on filtering the data that is produced by the method `newDirectoryStream`.

15. Write a program which, using a `DirectoryStream`, will do the following:

- Obtain the contents of the *Desktop* directory
- Will output the list of contents, doing the following:
  - Indicate, using one of the following strings “F→” or “D→” or “S→” at the front of the entry, whether the entry is a file or a directory or a symbolic link.
  - If the entry is a directory, the program should put the path separator at the end of the entry.

*Hint: Apply the following methods to the output from the preceding exercise: Files.isRegularFile, Files.isDirectory, Files.isSymbolicLink*

```
import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {
        Path desktopPath = Paths.get(System.getProperty("user.home"), "Desktop");
        String pathSeparator = System.getProperty("path.separator");
        try{
            DirectoryStream<Path> dirStream = Files.newDirectoryStream(desktopPath);
            for (Path item: dirStream) {
                if(Files.isRegularFile(item))
                    System.out.println("F-> "+item);
                else if(Files.isDirectory(item))
                    System.out.println("D-> "+item+pathSeparator);
                else if(Files.isSymbolicLink(item))
                    System.out.println("S-> "+item+pathSeparator);
                else
                    System.out.println(item);
            }
            dirStream.close(); //make sure you close the stream
        }
        catch(IOException e){}
    }
}
```

16. Write a program which, using a `DirectoryStream` will obtain the contents of the *Desktop* directory. The program should print out the files first, then a blank line and then print out the directories, putting the path separator at the end of each of the directory listings. Ignore the symbolic links.

*Hint: The solution to this exercise is a modification of the solution to the preceding exercise.*

## Practice Your Java Level 1

```

import java.nio.file.*;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class PracticeYourJava {
    public static void main(String[] args) {
        Path desktopPath = Paths.get(System.getProperty("user.home"), "Desktop");
        String pathSeparator = System.getProperty("path.separator");

        List<Path> dirList = new ArrayList<>();
        // What we do in the 1st loop is to loop through the DirectoryStream object dirStream
        // and print out the files but put the directory Path objects into the ArrayList object
        // dirList; we print out the contents of dirList in the following loop.
        try{
            DirectoryStream<Path> dirStream = Files.newDirectoryStream(desktopPath);
            for (Path item: dirStream) {
                if(Files.isRegularFile(item))
                    System.out.println(item);
                else if(Files.isDirectory(item))
                    dirList.add(item);
            }
            System.out.println();
            for (Path item: dirList){
                System.out.println(item+pathSeparator);
            }
            dirStream.close(); //make sure you close the stream
        }
        catch(IOException e){}
    }
}

```

### 17. Filtering DirectoryStream data

#### “Globbing”/Pattern Matching

Given the filters below, which of the items on the right would each of the filters specified below catch?

- |                     |                      |
|---------------------|----------------------|
| 1. a?c.txt          | a) abc.txt           |
| 2. a??c.txt         | b) abc.doc           |
| 3. a*               | c) def.txt           |
| 4. *.doc            | d) abc               |
| 5. A12*.doc         | e) abbd.txt          |
| 6. *.{txt,doc,docx} | f) abcd.txt          |
|                     | g) A1223432e3.doc    |
|                     | h) A12c.doc          |
|                     | i) abcdc.txt         |
|                     | j) country_list.docx |
|                     | k) breed_list.docx   |

*Hint:* The question mark (?) matches any single character. The \* matches any range of characters.

1. a
2. e, f
3. a, b, d, e, f, i
4. g, h
5. h
6. a, b, c, e, f, g, h, i, j, k

### 18. Write a program which will list all of the files on the Desktop that have the extension “.txt”.

*Hint: Use the overloaded method Files.newDirectoryStream to filter the data.*

```

import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {
        Path desktopPath = Paths.get(System.getProperty("user.home"), "Desktop");
        try{
            DirectoryStream<Path> dirStream = Files.newDirectoryStream(desktopPath, "*.txt");

```

	<pre> for (Path item: dirStream){     System.out.println(item); } dirStream.close(); //make sure you close the stream } catch(IOException e) {} } </pre>
19.	<p>Write a program which will list all of the files on the <i>Desktop</i> with the extension “.doc”, “.docx” or “.txt”.</p> <p>Modify the <code>DirectoryStream</code> instantiation line in the solution to the preceding exercise to the following:</p> <pre>DirectoryStream&lt;Path&gt; dirStream = Files.newDirectoryStream(directoryPath, "*.{txt,doc,docx}");</pre>
<i>File system information</i>	
20.	<p>List each of the root directories that are on your system.</p> <p><i>Hint: Study the class <code>FileSystems</code> and in particular its method <code>FileSystems.getDefault().getRootDirectories</code>. To solve this exercise, use the generic interface <code>Iterable</code>, as <code>Iterable&lt;Path&gt;</code> in conjunction with the method stated above.</i></p> <pre> import java.nio.file.*;  public class PracticeYourJava {     public static void main(String[] args) {         Iterable&lt;Path&gt; rootDirs = FileSystems.getDefault().getRootDirectories();         for(Path rootDir: rootDirs){             System.out.println(rootDir);         }     } } </pre>
21.	<p>For each of the filestores on your system, print out the following information: The filestore name, type, total drive space, unallocated drive space, usable drive space, available space.</p> <p><i>Hint: Study the class <code>FileSystems</code> and in particular its method <code>FileSystems.getDefault().getFileStores</code>. To solve this exercise, use the generic interface <code>Iterable</code>, as <code>Iterable&lt;FileStore&gt;</code> and the method stated above.</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         Iterable&lt;FileStore&gt; filestores = FileSystems.getDefault().getFileStores();         try{             for(FileStore fs: filestores){                 System.out.println("drive name = " + fs.name());                 //will print out the actual drive/partition name e.g. XT9720055 if assigned.                 System.out.println("drive type = " + fs.type()); //NTFS, FAT                 System.out.println("Total drive space      = " + fs.getTotalSpace() + " bytes");                 System.out.println("Unallocated drive space = " + fs.getUnallocatedSpace() + " bytes");                 System.out.println("Usable drive space     = " + fs.getUsableSpace() + " bytes");                 long availSpace = fs.getTotalSpace() - fs.getUnallocatedSpace();                 System.out.println("Available drive space = " + availSpace + " bytes");                 System.out.println();             }         catch(IOException e){}     } } </pre>
22.	<p>Extend the solution to exercise 20 to directly print the drive space information for each root directory.</p> <p><i>Hint: In exercise 20 we obtained an <code>Iterable&lt;Path&gt;</code>, which iterates over the list of <code>Path</code> objects that represent the root directories in the filesystem. Now, for <u>each</u> of the <code>Path</code> objects, obtain a <code>FileStore</code> object against it (see exercise 21) using the method <code>Files.getFileStore(&lt;Path&gt;)</code> and request the same information as in exercise 21.</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) { </pre>

## *Practice Your Java Level 1*

	<pre> Iterable&lt;Path&gt; rootDirs = FileSystems.getDefault().getRootDirectories(); for(Path rootDir: rootDirs) {     try{         FileStore fs = Files.getFileStore(rootDir);         double totalSpaceInMegs = (double)fs.getTotalSpace()/(double)(1024*1024);         System.out.println(rootDir);         System.out.printf("Total space = %.4f megs\n",totalSpaceInMegs);         double availSpaceInMegs = (double)(fs.getTotalSpace()-             fs.getUnallocatedSpace())/(double)(1024*1024);         System.out.printf("Available space = %.4f megs\n",availSpaceInMegs);         System.out.println();     }     catch(FileSystemException e){         System.out.println(e.getMessage());     }     catch(IOException e){} } } } </pre>
23.	<p>Print out, in megabytes, the available space for the drive on which the home directory resides.</p> <p><i>Hint: Files.getFileStore(&lt;desktop directory path&gt;)</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         Path desktopPath = Paths.get(System.getProperty("user.home"));         try{             FileStore fs = Files.getFileStore(desktopPath);             double totalSpaceInMegs = (double)fs.getTotalSpace()/(double)(1024*1024);             System.out.printf("Total space = %.4f megs\n",totalSpaceInMegs);              double availSpaceInMegs = (double)(fs.getTotalSpace()-                 fs.getUnallocatedSpace())/(double)(1024*1024);             System.out.printf("Available space = %.4f megs\n",availSpaceInMegs);         }         catch(IOException e){}     } } </pre>
	<p><i>File Information</i></p> <p>The questions in this section pertain to determining information about individual files.</p>
<b>M</b>	<p>Create a file named <code>abc.txt</code> on the <i>Desktop</i>. Enter any data into it. This file will be referenced in the exercises immediately below.</p>
24.	<p><i>File size</i></p> <p>Write a program which will print out the size of the earlier created file. Print the size out in bytes and megabytes. Ensure that the file exists before proceeding to check its size.</p> <p><i>Hint: Files.size</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         long   fileSize;         float  fileSizeMegs;         String fileName = "abc.txt";         Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);          try{             if(Files.notExists(targetFile)) {                 System.out.printf("The file %s does not exist. Exiting...\n", targetFile);                 System.exit(0);             }         }         catch(IOException e){             System.out.println(e.getMessage());         }     } } </pre>

	<pre>         }         fileSize = Files.size(targetFile);         fileSizeMegs = (float)fileSize/(float)(1024*1024);         System.out.printf("The size of %s = %d bytes, or %f megs.\n", targetFile, fileSize,                            fileSizeMegs);     }     catch(IOException e){} } } </pre>
25.	<p><i>Hidden files</i></p> <p>Write a program which will ascertain whether the earlier created file is <u>hidden</u> or not.</p> <p>After running the program, manually modify the permissions/properties on the earlier created file to “Hidden”, then rerun the program to validate its findings. Un-hide the file afterwards.</p> <p><i>Hint: Files.isHidden</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {          String fileName = "abc.txt";         Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);         try{             if(Files.notExists(targetFile)) {                 System.out.printf("The file %s does not exist. Exiting...\n", targetFile);                 System.exit(0);             }             if(Files.isHidden(targetFile))                 System.out.printf("The file %s is hidden\n", targetFile);             else                 System.out.printf("The file %s is NOT hidden\n", targetFile);         }         catch(IOException e){}     } } </pre>
26.	<p><i>Read-only files</i></p> <p>Write a program which will ascertain whether the earlier created file is <u>writeable</u> or not.</p> <p>After running the program, manually modify the permissions/properties on the earlier created file to “read-only”, then rerun the program to validate its findings. Do remember to leave the file in a writeable state afterwards.</p> <p><i>Hint: Files.isWritable</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {          String fileName = "abc.txt";         Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);         if(Files.notExists(targetFile)) {             System.out.printf("The file %s does not exist. Exiting...\n", targetFile);             System.exit(0);         }         if(Files.isWritable(targetFile))             System.out.printf("The file %s is writeable\n", targetFile);         else             System.out.printf("The file %s is NOT writeable\n", targetFile);     } } </pre>

## Practice Your Java Level 1

### 27. File executability

Write a program which will ascertain whether the file `abc.txt` is an executable file or not.

Note: Understand that concept of file executability does not necessarily mean that the file in question is a program file.

*Further testing:* If desired, On UNIX-like systems, to test this further, set the execute bit of the file using `chmod`. On computers running the Windows operating system, go into the file properties for the file and then go to `/Security/Edit` for your particular user name to modify the read&execute permission for the file.

*Hint: Files.isExecutable*

```
import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {
        String fileName = "abc.txt";
        Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);
        if(Files.notExists(targetFile)) {
            System.out.printf("The file %s does not exist. Exiting...\n", targetFile);
            System.exit(0);
        }
        if(Files.isExecutable(targetFile))
            System.out.printf("The file %s is executable\n", targetFile);
        else
            System.out.printf("The file %s is NOT executable\n", targetFile);
    }
}
```

### 28. File readability

Create a file named `ReadTest.txt` on your *Desktop* On UNIX based systems, set its read bit to unreadable using `chmod`. On Windows, go to the file properties and set its Read permission to Deny.

Write and test a program which will ascertain whether the specified target file is readable or not.

Make the file readable again and then rerun the program. It should yield the opposite result.

*Hint: Files.isReadable*

```
import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {
        String fileName = "ReadTest.txt";
        Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);
        if(!Files.exists(targetFile)){
            System.out.printf("The file %s does not exist.\n", targetFile);
            System.exit(0);
        }
        if(Files.isReadable(targetFile))
            System.out.printf("The file %s is readable.\n", targetFile);
        else
            System.out.printf("The file %s is NOT readable\n", targetFile);
    }
}
```

### 29. File access time

Print out the last time that the earlier created file `abc.txt` was modified.

*Hint: Files.getLastModifiedTime*

```
import java.nio.file.*;
import java.io.IOException;
import java.nio.file.attribute.FileTime;

public class PracticeYourJava {
    public static void main(String[] args) {
```

	<pre> String fileName = "abc.txt"; Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);  if(Files.notExists(targetFile)) {     System.out.printf("The file %s does not exist. Exiting...\\n", targetFile);     System.exit(0); }  try {     FileTime ft = Files.getLastModifiedTime(targetFile);     System.out.println("The last time the file was modified was: " + ft); } catch (IOException e) {     e.printStackTrace(); } } } </pre>
30.	<p>Using the interface <code>BasicFileAttributeView</code>, determine and print out the following about the file <code>abc.txt</code>:</p> <ul style="list-style-type: none"> <li>creation time</li> <li>last access time</li> <li>modification time</li> </ul> <p><i>Hint: <code>Files.readAttributes</code></i></p> <pre> import java.nio.file.*; import java.io.IOException; import java.nio.file.attribute.BasicFileAttributes;  public class PracticeYourJava {     public static void main(String[] args) {          String fileName = "abc.txt";         Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);          if(Files.notExists(targetFile)) {             System.out.printf("The file %s does not exist. Exiting...\\n", targetFile);             System.exit(0);         }         try {             BasicFileAttributes bfa = Files.readAttributes(targetFile, BasicFileAttributes.class);             System.out.println(bfa.creationTime());             System.out.println(bfa.lastModifiedTime());             System.out.println(bfa.lastAccessTime());         } catch (IOException e) {             e.printStackTrace();         }     } } </pre>
31.	<p>Print out the owner of the file <code>abc.txt</code>.</p> <p>If desired, you can modify the owner and rerun the program.</p> <p><i>Hint: <code>Files.getOwner</code></i></p> <pre> import java.nio.file.*; import java.io.IOException; import java.nio.file.attribute.UserPrincipal;  public class PracticeYourJava {     public static void main(String[] args) {          String fileName = "abc.txt";         Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);          if(Files.notExists(targetFile)) {             System.out.printf("The file %s does not exist. Exiting...\\n", targetFile);             System.exit(0);         }         try { </pre>

## Practice Your Java Level 1

```
UserPrincipal owner = Files.getOwner(targetFile);
System.out.printf("The owner of the file is: " + owner);
} catch (IOException e) {
    e.printStackTrace();
}
}
```

32. Print out the name of the volume on which the file `abc.txt` is stored.

*Hint: Files.getFileStore*

```
import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {

        String fileName = "abc.txt";
        Path targetFile = Paths.get(System.getProperty("user.home"), "Desktop", fileName);

        if(Files.notExists(targetFile)) {
            System.out.printf("The file %s does not exist. Exiting...\n", targetFile);
            System.exit(0);
        }

        try {
            FileStore fs = Files.getFileStore(targetFile);
            System.out.printf("The filestore of the file is: " + fs);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

### File Operations

In this section exercises on file operations such as *copying, deleting, moving and suchlike* are presented.

*Hint: Files.copy*

33. *Copying Files*

Write a program which will **copy** the *Desktop* file `abc.txt` to the earlier created directory `Dir_A`.

*Hint: Files.move*

```
import java.nio.file.*;
import java.util.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {

        String targetDir = "Dir_A"; //the directory we want to move it to
        String fileToCopy = "abc.txt";
        Path sourcePath, targetPath;

        sourcePath = Paths.get(System.getProperty("user.home"), "Desktop", fileToCopy);
        targetPath = Paths.get(System.getProperty("user.home"), "Desktop", targetDir, fileToCopy);

        //the actual file to move
        if(Files.notExists(sourcePath)){
            System.out.println("The source file does not exist. Exiting...");
            System.exit(0);
        }
        if(Files.exists(targetPath)){
            System.out.println("A file with the same name already exists at the target location.
                                Exiting...");
            System.exit(0);
        }
        if(!Files.isWritable(targetDirPath)){
            System.out.println("The target directory is not writeable. Exiting...");
        }
    }
}
```

	<pre>         //Now do the copying         Files.copy(sourcePath, targetPath);         System.out.println("The copy has been done.");     } } </pre>
34.	<p><i>Renaming/Moving Files</i></p> <p>Write a program which will rename the <i>Desktop</i> file <b>abc.txt</b> to <b>def.txt</b>.</p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {          String oldFileName = "abc.txt";         String newFileName = "def.txt";          Path sourcePath = Paths.get(System.getProperty("user.home"), "Desktop", oldFileName);         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", newFileName);          if(Files.notExists(sourcePath)){             System.out.println("The source file does not exist. Exiting...");             System.exit(0);         }         if(Files.exists(targetPath)){             System.out.println("A file with the same name already exists at the target location.                 Exiting...");             System.exit(0);         }         if(!Files.isWritable(targetPath)){             System.out.println("The target directory is not writeable. Exiting...");         }         //Now do the move         try {             Files.move(sourcePath, targetPath);         } catch (IOException e) {             System.out.println("Error: Could not move the file");             e.printStackTrace();         }         System.out.println("The move/rename has been done.");     } } </pre> <p><b>Note:</b> The same code applies for moving or renaming files.</p>
35.	<p><i>Deleting files</i></p> <p>Write a program which will delete the <i>Desktop</i> file <b>def.txt</b>.</p> <p><i>Hint: Files.deleteIfExists</i></p> <pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {          String fileName = "def.txt";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", fileName);         boolean deletionResult=false;         try {             deletionResult = Files.deleteIfExists(targetPath);         } catch (IOException e) {             System.out.println("Could not delete file!");             e.printStackTrace();             System.exit(0);         }         if(deletionResult == false)             System.out.println("The deletion could not be done.");     } } </pre>

## Practice Your Java Level 1

```
        else
            System.out.println("The file has been deleted.");
    }
}
```

**Note:** If the file does not exist, this program using the method `Files.deleteIfExists` will tell you “the deletion could not be done”, but not tell you why.

### Recursive Directory Operations

#### 36. Subfolder traversal operations (traversing, deleting)

Describe how you would traverse a directory tree and print out the names of each of the files and directories therein.

One way in which to perform the described task is to use as a concrete class or as an anonymous class the class `SimpleFileVisitor`, an instance of which is passed to the built-in directory path traversal method `Files.walkFileTree`.

`Files.walkFileTree` is designed to walk recursively through a path that is passed to it. `Files.walkFileTree` also takes an instance of `SimpleFileVisitor` as noted above and performs the actions in this class.

`SimpleFileVisitor` has the following methods which you can modify as desired:

- `postVisitDirectory` – this is invoked by `Files.walkFileTree` after it has recursively visited every descendant directory and file of the current directory that it is in and then it performs the actions specified in this method.
- `preVisitDirectory` – this is invoked by `Files.walkFileTree` before it has recursively visited every descendant directory and file of the current directory that it is in and performs the actions specified in this method.
- `visitFile` – this method is invoked by `Files.walkFileTree` for each file that it visits.
- `visitFileFailed` – this method is invoked by `Files.walkFileTree` when it cannot “visit” a file.

If for example you wanted to print out the names of the directories and files that you visit, you would put that code into the appropriate listed method. If you wanted to delete the directories and files that you visit, you would put that code into the appropriate one of these methods.

#### 37. The program below is trying to obtain the total number of files and the total number of sub-directories in a given directory path, recursively. Explain how the program achieves this objective.

```
import java.nio.file.*;
import java.io.IOException;
import java.nio.file.attribute.BasicFileAttributes;

public class SimpleFileVisitor2 extends SimpleFileVisitor<Path> {

    public int fileCount=0;
    public int directoryCount=0;

    public FileVisitResult visitFile(Path pathItem, BasicFileAttributes attr){
        fileCount++;
        return(FileVisitResult.CONTINUE);
    }
    public FileVisitResult postVisitDirectory(Path pathItem, IOException e){
        directoryCount++;
        return(FileVisitResult.CONTINUE);
    }
}

public class PracticeYourJava {
    public static void main(String[] args) {

        String dirName = "A"; //The directory we'd created on the desktop earlier
        Path pathToDir = Paths.get(System.getProperty("user.home"), "Desktop", dirName);

        System.out.println("The target directory =" + pathToDir);

        //Okay the new stuff to really pay attention to
        SimpleFileVisitor2 sfv2 = new SimpleFileVisitor2();
        FileVisitor<Path> fv = sfv2;
```

```

    Files.walkFileTree(pathToDir, (FileVisitor<Path>)sfv2);
    System.out.println("File count      = " + sfv2.fileCount);
    System.out.println("Directory count = " + sfv2.directoryCount);
}
}

```

The code shows that we are trying to walk the file tree recursively, passing to the method `Files.walkFileTree` a `Path` object and a `FileVisitor<Path>` object.

The `Path` object in question points to the directory “A” that we’d created earlier on the *Desktop*.

The `FileVisitor<Path>` points to an object `sv2` which we created, which is an instance of the new class `SimpleFileVisitor2` that we just created.

The class `SimpleFileVisitor2` extends `SimpleFileVisitor<Path>`. In particular it does the following:

- It has two variables for holding the sum of files and the sum of directories respectively.
- It has modified the method `postVisitDirectory` to increment the directory counter.
- It has modified the method `FileVisitResult` to increment the file counter.

Back in `main`, we simply invoked `Files.walkFileTree` with the `Path` object in question and an instance of the class `SimpleFileVisitor` (or its descendant class `SimpleFileVisitor2` in this case), let it run and then printed out the results that we want.

38. Print out the contents of the earlier created directory “A”, recursively. Ensure that the contents of any given directory are indented by a single space below the name of their containing directory. Put the path separator at the end of the directory entries so that we can tell that it is a directory.

```

import java.nio.file.*;
import java.io.IOException;
import java.nio.file.attribute.BasicFileAttributes;

public class PracticeYourJava {
    public static void main(String[] args) {
        String dirName = "A";
        Path pathToDir = Paths.get(System.getProperty("user.home"), "Desktop", dirName);
        String pathSeparator = System.getProperty("path.separator");

        System.out.println("The target directory = " + pathToDir);
        FileVisitor<Path> fv = new SimpleFileVisitor<Path>() {
            //we create an anonymous class here...good example of need for one.
            int count=0; //we use this to count how far down in the tree, for indentation
            public FileVisitResult visitFile(Path pathItem, BasicFileAttributes attr) {
                if(count > 0) {
                    for(int x=0;x<=(count+1);x++)
                        System.out.print(" ");
                }
                System.out.println(pathItem);
                return(FileVisitResult.CONTINUE);
            }

            public FileVisitResult preVisitDirectory(Path pathItem, BasicFileAttributes attr) throws
                IOException {
                count++;
                if(count > 0) {
                    for(int x=0;x<=count;x++)
                        System.out.print(" ");
                }
                System.out.println(pathItem+pathSeparator);
                return(FileVisitResult.CONTINUE);
            }
        };
        try{
            Files.walkFileTree(pathToDir, fv);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

## Practice Your Java Level 1

39. Write a program to delete the sub-directory tree B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/Y/Z that is in the directory “A” that you previously created on the *Desktop*. Implement this without full exception handling.

**Note:** Great care must be taken in writing and executing this code, as it performs recursive deletions.

*Hint: Set the working directory to Desktop/A and then perform the delete operation there, recursively, using the tree walking feature.*

```
import java.nio.file.*;
import java.io.IOException;
import java.nio.file.attribute.BasicFileAttributes;

public class PracticeYourJava {
    public static void main(String[] args) {

        String dirName = "A/B";
        String pathSeparator = System.getProperty("file.separator");
        dirName = dirName.replace("/", pathSeparator);
        Path pathToDir = Paths.get(System.getProperty("user.home"), "Desktop", dirName);

        System.out.println("The target directory = " + pathToDir);

        FileVisitor<Path> fv = new SimpleFileVisitor<Path>(){
            //anonymous class used here...
            public FileVisitResult visitFile(Path pathItem, BasicFileAttributes attr) throws
                IOException{
                Files.delete(pathItem);
                return(FileVisitResult.CONTINUE);
            }
            public FileVisitResult postVisitDirectory(Path pathItem, IOException e) throws
                IOException{
                Files.delete(pathItem);
                return(FileVisitResult.CONTINUE);
            }
        };
        try{
            Files.walkFileTree(pathToDir, fv);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
}
```

**Note:** In a more complete version of this, we would check to see if the files can be deleted and if not, then we are not able to delete their containing directory. We would also check to see if a given directory is accessible to us at all for reading and then for deletion.

### Symbolic link operations

The concept of symbolic links, which while well-known in the Unix/Linux world, is not as well known in the Windows space. The exercises that follow deal with this topic and how symbolic links are handled using Java.

40. What are symbolic links?

In general terms a symbolic link is a “named pointer” in the filesystem to another entry in the filesystem. The entry pointed at can be a file a directory or even another symbolic link.

There are two types of symbolic links; soft links and hard links (we look closer at these in another exercise).

The number of symbolic links to a given file/directory is only limited by the operating system.

Deleting a symbolic link has no effect on the file that the link is pointing to.

**Note:** In Windows, you have to have administrator privileges in order to create a symbolic link. Therefore, if you want to create one from a program running directly in your IDE, you have to start the IDE with Administrator privileges (In Unix/Linux systems, the creation of symbolic links to files that you have access to does not generally require elevated permissions). Otherwise, you can run your symbolic-link creating program directly in a command prompt window that was started with Administrator privileges.

41. Manually create a file named `rhyme.txt` on the *Desktop*. Enter some text into it. Then write a program that uses the method `Files.createSymbolicLink` to create a soft symbolic link on the *Desktop* named `lNK01` to it.

**Note:** If you are running on Windows, see the constraint noted in the preceding solution.

```

import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {

        String fileName = "rhyme.txt";
        String symLinkName = "lnk01";
        Path targetPath = Paths.get(System.getProperty("user.home"), fileName);
        Path symlinkPath = Paths.get(System.getProperty("user.home"), symLinkName);
        try {
            Files.createSymbolicLink(symlinkPath, targetPath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

**Note:** open the newly created symbolic link with whichever application you would use to read rhyme.txt; you will note that indeed it is the contents of rhyme.txt that appear in your application.

42. Write a program which will check whether the link created in the preceding exercise is a symbolic link or not. Print your findings out.  
Note: Windows users: If you were unable to create the link programmatically, you can create one in a command window with Administrator privileges, using the command `mklink`.

*Hint: Files.isSymbolicLink*

```

import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {

        String symLinkName = "lnk01";
        Path symlinkPath = Paths.get(System.getProperty("user.home"), symLinkName);
        if(Files.isSymbolicLink(symlinkPath))
            System.out.printf("%s is a symbolic link\n", symlinkPath);
        else
            System.out.printf("%s is NOT a symbolic link\n", symlinkPath);
    }
}

```

43. Write a program which will print out the name of the actual target of the symbolic link of the preceding exercise.

*Hint: Files.readSymbolicLink*

```

import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {

        String symLinkName = "lnk01";
        Path symlinkPath = Paths.get(System.getProperty("user.home"), symLinkName);
        if(!(Files.isSymbolicLink(symlinkPath))){
            System.out.printf("%s itself is not a symbolic link, so nothing to detect..Exiting\n",
                symlinkPath);
            System.exit(0);
        }
        Path actualTarget;
        try {
            actualTarget = Files.readSymbolicLink(symlinkPath);
            System.out.printf("%s is pointing to %s\n", symlinkPath, actualTarget);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

## Practice Your Java Level 1

	<p><b>Output:</b> The program will simply output the path of the target relative to the link. For example, if the link and the target are in the same directory, then the program will simply output the name of the target.</p>
44.	<p>What is the difference between a hard symbolic link and a soft symbolic link? Also, explain the effect on soft and hard links of the following actions on the file being referenced 1) updating, 2) renaming and 3) deleting</p> <p>This solution requires some background knowledge on how filesystems work. We present first a <u>simplified view</u> of how filesystems can track entries in order to answer the question properly.</p> <p>The filesystem has a “file directory” (think of a phone directory here), where it stores the following tuples of information about the files on the filesystem in a manner akin to the following:</p> <p><i>(entry#, entry name, entry type, actual file location on disk, target)</i></p> <p>In our example, the field <b>entry type</b> will have the following entries F(file), D(Directory), S(Soft Symbolic Link), H(Hard Symbolic Link).</p> <p>For the purposes of this illustration, think of <b>entry name</b> as the fully resolved name of the directory entry in question.</p> <p>A file <b>abc.txt</b> on the <i>Desktop</i> would be entered in our file directory as follows:</p> <pre>(#1, "C:/Users/PYJ/Desktop/abc.txt", F, 1500001, "")</pre> <p>A soft symbolic link named <b>softLink01</b> to this file would be entered as:</p> <pre>(#2, "softLink01", S, , "C:/Users/PYJ/Desktop/abc.txt")</pre> <p>Observe that the soft symbolic link does not have an entry indicating the actual location of the file on disk.</p> <p>A hard symbolic link named <b>hardLink01</b> to this file would be entered as:</p> <pre>(#3, hardLink01, H, 1500001, "")</pre> <p>Observe that the hard symbolic link does not have an entry indicating the name of the file being pointed to (as the soft symbolic link does), but rather has an entry pointing to the actual location of the file on disk. Also, be aware that the operating system stays aware of who is pointing to the actual physical location of a file on disk.</p> <p><i>File Access mechanism for soft link vs hard link</i></p> <p>When a file is accessed through a <u>soft symbolic link</u>, the operating system looks at the name of what the soft link is pointing at, goes to the actual entry for that item and from the original entry determines the actual location on disk of the file and accesses the file.</p> <p>When a file is accessed through a <u>hard symbolic link</u>, the operating system looks at the disk location that the hard link itself is directly pointing at and then accesses the file in that way.</p> <p><i>Effect on links of updating a file</i></p> <p>If we update the file contents, whether the file is accessed through either a hard link or a soft link, the access mechanism is as described above. The file contents can also be updated directly or through the links.</p> <p><i>Effect on links of renaming a file</i></p> <p>Now, if the file <b>abc.txt</b> is renamed to <b>def.txt</b>, entry #1 becomes the following:</p> <pre>(#1, "C:/Users/PYJ/Desktop/def.txt", F, 1500001, "")</pre> <p>Observe that on renaming a file the actual location of the file (1500001) is not moved if it renamed on the same filesystem (C: in this example). <u>Understand that on renaming a file neither the soft links nor the hard links to the file are updated.</u></p> <p><i>Effect on soft link</i></p> <p>Entry #2 (the soft link) is now a “dead link” as there is no longer a C:/Users/PYJ/Desktop/abc.txt to point to! The file can no longer be accessed through the soft link.</p> <p><i>Effect on hard link</i></p> <p>The file content is still accessible through the hard link <b>hardlink01</b> because the filesystem knows from the entry of the hard link the actual location of the data that the hard link is pointing to.</p> <p><i>Effect on links of deleting a file</i></p> <p>According to our previous explanation, this means that entry #2 is deleted from the file directory.</p> <p><i>Effect on soft link</i></p>

	<p>This results in entry #2 (the soft link) being a “dead link” as there is no longer a <code>C:/Users/PYJ/Desktop/abc.txt</code> to point to! The file can no longer be accessed through the soft link.</p> <p><i>Effect on hard link</i></p> <p>Interestingly, <u>the file content is still accessible through the hard link <code>hardLink01</code></u> because the filesystem knows from the entry of the hard link the actual location on the disk of the file/data that the hard link is pointing to. Recall that it was stated earlier that the operating system keeps a record of who is actually pointing to data locations on this disk, i.e. the operating system is mindful that there is still someone (in our case <code>hardLink01</code>) pointing the file location <code>1500001</code>. Therefore, when the file <code>abc.txt</code> was deleted, only its entry in the directory listings was removed, but not the actual data that it pointed to, as someone else was also pointing to that data.</p> <p>This is the explanation of the functioning of soft symbolic links and hard symbolic links.</p>
45.	<p>Create a <u>hard symbolic link</u> named <code>lnk02</code> to the earlier linked to file <code>rhyme.txt</code>.</p> <p><i>Hint: <code>Files.createLink</code></i></p> <pre>import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {          String fileName = "rhyme.txt";         String symLinkName = "lnk02";         Path targetPath = Paths.get(System.getProperty("user.home"), fileName);         Path symlinkPath = Paths.get(System.getProperty("user.home"), symLinkName);         if(Files.notExists(targetPath)){             System.out.println("The target file does not exist. Exiting...");             System.exit(0);         }         try {             Files.createLink(symlinkPath, targetPath);         } catch (IOException e) {             e.printStackTrace();         }     } }</pre>
46.	<p>Do a directory listing of the directory where the created links are; observe and comment on the difference in the way the soft links vs. the hard links appear in the directory listing.</p> <p>It is clear to see from the entry for the soft link that this represents a link. The entry for the hard link however does not appear to be a link, but rather looks like a “copy” of the target file!</p>
47.	<p><i>File Operations with Options</i></p> <p>What is the effect if you use the <code>Files.copy</code> method without any extra options to attempt to overwrite an already existing file?</p> <p>An exception, <code>java.nio.file.FileAlreadyExistsException</code>, will result.  <b>Note:</b> If you try to copy a file to itself, the request is quietly ignored.</p>
48.	<p>How do you effect an overwriting of an existing file using the method <code>Files.copy</code>?</p> <p><i>Hint: enumeration <code>StandardCopyOption</code></i></p> <p>You would apply the option <code>StandardCopyOption.REPLACE_EXISTING</code> to the <code>Files.copy</code> command. So instead of just,</p> <pre>Files.copy(Path sourcePath, Path targetPath);</pre> <p>You would do the following:</p> <pre>Files.copy(Path sourcePath, Path targetPath, StandardCopyOption.REPLACE_EXISTING);</pre>
49.	<p>What is the function of the <code>StandardCopyOption.COPY_ATTRIBUTES</code> enumeration option?</p> <p>This option ensures that, as far as the underlying operating system permits, that the attributes of the original file are copied to the copy of the file. These attributes include but are not limited to file timestamps (creation, modification, last access time) and POSIX attributes.</p>

## Practice Your Java Level 1

50.	<p><i>Path objects as Uniform Resource Identifiers</i></p> <p>What is the concept of a Uniform Resource Identifier (URI)?</p> <p>The concept of a URI is a string of characters used to identify a resource. Examples of URI's are the strings which allow you to paste a "URL-like" string that refers to a file or directory directly into a web browser on your computer and have it bring up the file or directory to which it refers. A URL is in fact a type of URI.</p> <p>The structure of a URI is: a scheme name followed by a colon and then the network/computer location of the resource. Take for example a URL which is of the form <code>http://&lt;location&gt;</code>. The URI scheme in this case is "http", which is followed by a colon; the rest of the line indicates the location of the resource.</p> <p>There are a number of URI schemes; for example <code>http</code> (the HyperText Transfer Protocol, used for accessing web pages), <code>https</code> (secure http access) and <code>file</code> (used for accessing files on local computers or across the network), among others. For example, a file on your computer might have the following URI:</p> <p style="text-align: center;"><code>file:///C:/Users/PYJ/Desktop/abc.txt</code></p> <p>Applications which receive URIs know how to access and process the resource that the URI refers to. For example, a web browser knows how to interpret the URI's with a URI scheme of <code>http</code>, <code>https</code> and at least <code>file</code>.</p> <p>A fuller explanation of URI's can be found in the Internet Engineering Task Force document RFC3986 which is currently obtainable at <a href="https://tools.ietf.org/html/rfc3986">https://tools.ietf.org/html/rfc3986</a>.</p>
51.	<p>Given an existing file <code>abc.txt</code> on your <i>Desktop</i>, write a program to obtain its URI equivalent. Print the obtained URI out and test it by pasting it into a web browser.</p> <p><i>Hint: get a Path for the file and then use the &lt;Path&gt;.toUri() method to get the URI.</i></p> <pre>import java.net.URI; import java.nio.file.*;  public class PracticeYourJava {     public static void main(String[] args) {         String fileName = "abc.txt";         Path sourcePath = Paths.get(System.getProperty("user.home"), "Desktop", fileName);         URI uri01 = sourcePath.toUri();         System.out.println(uri01);     } }</pre> <p><b>Note:</b> For further practice, you can create a URI to a directory on your computer and then paste that URI into a web browser.</p>

### Temporary Directories and Files

There are times when your application requires temporary directories and files for its processing. Certain operating systems have a designated area where these temporary items are stored. Java provides means by which temporary directories and files can be created in the default temporary file space.

52.	<p>Create a temporary file with the name prefix ABC and the name suffix FFF in the default temporary directory. Print out the name of the resulting temporary file.</p> <p><i>Hint: Files.createTempFile</i></p> <pre>import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {         String prefix = "ABC";         String suffix = "FFF";         Path filePath;         try {             filePath = Files.createTempFile(prefix, suffix);             System.out.println("The name of the new temp file = " + filePath);         } catch (IOException e) {             e.printStackTrace();         }     } }</pre>
-----	---

53. Create a temporary directory in the default temporary directory with the name prefix PYJ in the default temporary directory. Print the resulting Path out.

Also, put a temporary file with the name prefix ABC and the name suffix FFF into this directory.

```
import java.nio.file.*;
import java.io.IOException;

public class PracticeYourJava {
    public static void main(String[] args) {

        String dirPrefix = "PYJ";
        String filePrefix = "ABC";
        String fileSuffix = "FFF";

        try {
            Path dirPath = Files.createTempDirectory(dirPrefix); //simple!
            System.out.println("The path of the new temp dir = " + dirPath);
            //Now create the temporary file inside this dir. In the previous question
            //we just created the file directly in the default temporary directory
            Path filePath = Files.createTempFile(dirPath, filePrefix, fileSuffix);
            System.out.println("The name of the new temp file = " + filePath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

### Zip Files

54. Describe the structure of a program which creates a zip file in Java.

In broad terms, the steps are: (1) create a zip archive `FileSystem` object (2) using a `Map` object, specify what action you want to apply to this object, whether creation of a zip archive, or extraction therefrom (3) extract or insert files to/from the zip archive. We expatiate on the steps below.

The first step is to create a new `FileSystem` object, using one of the overloaded versions of the method `FileSystems.newFileSystem`. The two potential versions we look at have the following method definitions:

```
newFileSystem(Path, Map, null);
and
newFileSystem(URI, Map);
```

The description of the contents of the variable types in the methods is as follows:

**Path:** The last element of the `Path` object must be the name of the target zip file which MUST end in the extension `.zip`. This requirement is to enable the method to determine what kind of `FileSystem` object it is creating; it will successfully determine that the intent is to create a zip file archive by noting the file extension.

**Map:** This is a `Map<String, String>` object which contains specific values that a zip filesystem uses to determine what actions we want to undertake; to create a zip file, we have to insert into the `Map` object the `String, String` tuple `"create", "true"`. When we are extracting from the zip file we ensure that the pair is `"create", "false"`.

With the preceding values, we can create our `FileSystem` variable using the first method shown above.

In order to use the second version of the method, the location of the intended zip file must first be converted to a `URI` object for a zip file as follows:

Convert the target `Path` object of the intended zip file to a `URI`, using the `<Path>.toUri()` method

After that, the program then has to convert this `URI` back to a path, using the `<uri>.toPath()` method

The full URI of the zip file is determined to be the prefix `"jar:file"` to which the earlier created URI is appended.

All of the above can be written as:

```
URI uri_instance = URI.create("jar:file: " + targetPath.toUri().getPath());
```

Where `targetPath` is the `Path` object which contains the path of the intended zip file.

The same `Map<String, String>` described above is what should be used for the method variant that takes URIs.

With this we can create a `FileSystem` object using the second overloaded method.

## Practice Your Java Level 1

	<p><i>Insertion into the zip file</i></p> <p>After obtaining a <code>FileSystem</code> object, insertion into the zip file is done easily by using the <code>Files.copy</code> method as follows:</p> <ol style="list-style-type: none"> <li>1. The first parameter is the <code>Path</code> object of the file that we are putting into the archive</li> <li>2. The second parameter is a <code>Path</code> object representing the location in the zip file of the object that we are inserting</li> <li>3. The third parameter is the <code>StandardCopyOption</code> enum that we want to use; for example, we might be okay, or not, with overwriting a file with the same name that might already be existent in the archive.</li> </ol>
55.	<p><i>Creating a zip file and inserting into it</i></p> <p>Manually identify a file of choice that you want to put into a zip file. (We choose here the file <code>abc.txt</code> that is on the <i>Desktop</i>).</p> <p>Create a zip file, named <code>zip01.zip</code> on the <i>Desktop</i>. Insert the previously identified file into it. If any such file is already existent in the archive in the same location, replace it.</p> <pre>import java.nio.file.*; import java.io.IOException; import java.net.URI; import java.util.HashMap; import java.util.Map;  public class PracticeYourJava {     public static void main(String[] args) {          Map&lt;String, String&gt; env = new HashMap&lt;&gt;();         env.put("create", "true");         String zipName      = "zip01.zip";         String fileToInsert = "abc.txt";          Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", zipName);          try {             URI uri = URI.create("jar:file:" + targetPath.toUri().getPath());             FileSystem zipFileSystem01 = FileSystems.newFileSystem(uri, env);             Path inputFile = Paths.get(System.getProperty("user.home"), "Desktop", fileToInsert);             Path pathInZipfile = zipFileSystem01.getPath(fileToInsert);             //copy the file into the zip file             Files.copy(inputFile, pathInZipfile, StandardCopyOption.REPLACE_EXISTING );             zipFileSystem01.close();         }         catch(IOException e){             System.out.println("Could not put the file into zip directory");             System.out.println(e.getStackTrace());         }     } }</pre>
56.	<p><i>Extracting from an already existing zip file</i></p> <p>Extract the file that was previously inserted into the zip file and put in on the <i>Desktop</i> with the name <code>def.txt</code>.</p> <pre>import java.nio.file.*; import java.io.IOException; import java.net.URI; import java.util.HashMap; import java.util.Map;  public class PracticeYourJava {     public static void main(String[] args) {          Map&lt;String, String&gt; env = new HashMap&lt;&gt;();         env.put("create", "false");         String zipName      = "zip01.zip";         String fileToExtract = "abc.txt";         String fileNewName = "def.txt";          Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", zipName);         System.out.println(targetPath.toUri().getPath());     } }</pre>

```

URI uri = URI.create("jar:file:" + targetPath.toUri().getPath());
try (FileSystem zipfs = FileSystems.newFileSystem(uri, env)) {
    Path locationToExtractFileTo = Paths.get(System.getProperty("user.home"), "Desktop",
        fileNewName);
    Path pathInZipfile = zipfs.getPath(fileToExtract);
    Files.copy(pathInZipfile, locationToExtractFileTo);
}
catch(IOException e){}
}
}

```

**Path Handling... more details**

In this section we look at a few miscellaneous details regarding the handling of `Path` objects.

	<p>57. Given the following path:  Windows: C:\Users\PYJ\PracticeDir\testfile01.txt  *NIX: /home/PYJ/PracticeDir/testfile01.txt  Extract the end element and print it out.  <i>Hint: Path.getFileName</i></p>
	<pre> import java.nio.file.*; import java.io.IOException;  public class PracticeYourJava {     public static void main(String[] args) {          Path targetPath = Paths.get("C:\\\\Users\\\\PYJ\\\\PracticeDir\\\\testfile01.txt"); //on Windows          Path endElement = targetPath.getFileName();         System.out.println("The end element is : " + endElement);     } } </pre> <p><b>Note:</b> It does not matter whether the end element is a file or a directory, the method <code>getFileName</code> retrieves it.</p>
	<p>58. With respect to the same paths as in the preceding exercise, print the root element of the path.  <i>Hint: Path.getRoot</i></p>
	<pre> import java.nio.file.*;  public class PracticeYourJava {     public static void main(String[] args) {          Path targetPath = Paths.get("C:\\\\Users\\\\PYJ\\\\PracticeDir\\\\testfile01.txt");          Path rootElement = targetPath.getRoot();         System.out.println("The root element is : " + rootElement);     } } </pre>
	<p>59. With respect to the same path as in the preceding exercise, write a program which prints out how many elements there are in the path.  <i>Hint: Path.getNameCount</i></p>
	<pre> import java.nio.file.*;  public class PracticeYourJava {     public static void main(String[] args) {          Path targetPath = Paths.get("C:\\\\Users\\\\PYJ\\\\PracticeDir\\\\testfile01.txt");          int numberOfElements = targetPath.getNameCount();         System.out.println("The number of elements in the path is : " + numberOfElements);     } } </pre>
	<p>60. With respect to the same paths as in the preceding exercise, write a program which determines what the parent directory of the end element in the path is.  <i>Hint: Path.getParent</i></p>
	<pre> import java.nio.file.*;  public class PracticeYourJava { </pre>

## Practice Your Java Level 1

	<pre> public static void main(String[] args) {     Path targetPath = Paths.get("C:\\\\Users\\\\PYJ\\\\PracticeDir\\\\testfile01.txt");     Path parentPath = targetPath.getParent();     System.out.println("The parent's path is : " + parentPath); } } </pre>
61.	<p>Write a program which will remove the redundant elements from the following path. Print the resulting path out.</p> <p>Windows: C:\\Users\\PYJ\\PracticeDir\\..\\PracticeDir\\testfile01.txt      *NIX: \\home\\PYJ\\PracticeDir\\..\\PracticeDir\\testfile01.txt</p> <p><i>Hint: Path.normalize</i></p> <pre> import java.nio.file.*;  public class PracticeYourJava {     public static void main(String[] args) {         Path targetPath = Paths.get("C:\\\\Users\\\\PYJ\\\\PracticeDir\\\\..\\\\PracticeDir\\\\testfile01.txt");         Path normalizedPath = targetPath.normalize();         System.out.println("The normalized path is : " + normalizedPath);     } } </pre>
E1	Write a program which will put a complete folder/directory into a zip file.
E2	Write a program which will extract all of the components of a zip file into a directory of choice.
E3	Write a program which goes through a directory recursively and lists all of the files with the extension *.txt or *.doc.
E4	Rewrite the solution to exercise 39 (recursive deletion of directories), this time, ensuring that checks for file and directory accessibility and deletability are done.
E5	Investigate the purpose and function of the method <code>Files.probeContentType</code> .
E6	If you are on a UNIX based system, write a program with which to set the group user and other permissions on a file ( <i>Hint: see class PosixFilePermissions</i> ).

# Chapter 30. File System II: File Input & Output

In the preceding chapter, we looked at file handling, however didn't address the contents of the files. This chapter presents exercises which deal with the reading and writing of files. Two different pairs of ways of reading/writing files using NIO/NIO.2 are presented in this chapter. The main classes utilized in this chapter include the classes `Files`, `Path`, `Paths`, `BufferedReader` and `BufferedWriter`.

1.	<p><i>Reading the contents of a text file using <code>File.readAllLines</code></i></p> <p>Manually create a file <code>abc.txt</code> on the <i>Desktop</i>. Using a text editor of your choosing, place into it any text of your choosing. Now, write a program which uses the method <code>Files.readAllLines</code> to read the contents of the file and output the contents to the console.</p>
	<pre>import java.io.IOException; import java.nio.file.*; import java.util.List;  public class FileHandling {     public static void main(String[] args) {          String filename = "abc.txt";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", filename);         List&lt;String&gt; fileContents = null;          if(Files.isReadable(targetPath)){             try {                 fileContents = Files.readAllLines(targetPath);             } catch (IOException e) {                 e.printStackTrace();             }             for(String s:fileContents){                 System.out.println(s);             }         }         else             System.out.println("The file is not readable \n");     } }</pre>
2.	<p><i>Writing to a text file using <code>Files.write</code></i></p> <p>Write the first three lines of the rhyme “Mary had a little lamb” to a file named <code>rhyme.txt</code> (put this file on the <i>Desktop</i>). Look at the contents of the file after running the program in order to confirm that the file was written to.</p>
	<pre>import java.io.IOException; import java.nio.file.*; import java.util.ArrayList; import java.util.List;  public class FileHandling {     public static void main(String[] args) {          List&lt;String&gt; rhyme = new ArrayList&lt;&gt;();         rhyme.add("Mary had a little lamb");         rhyme.add("little lamb, little lamb");         rhyme.add("Mary had a little lamb that was as white as snow");          String filename = "rhyme.txt";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", filename);          try{             Files.write(targetPath, rhyme);             System.out.println("The data has been written to the file");         }</pre>

## Practice Your Java Level 1

	<pre>        }     catch(IOException e){         System.out.println(e.getMessage());     } }</pre>
3.	If you removed from the program in the preceding exercise the code which writes the first two lines of the rhyme to the <code>List&lt;String&gt;</code> , leaving the single line <code>rhyme.add("Mary had a little lamb that was as white as snow")</code> and then ran the modified program, what would the contents of the file <code>rhyme.txt</code> then be? <b>Findings:</b> The contents of the file are now only that third line, “Mary had a little lamb that was as white as snow”, i.e. the contents were overwritten, not appended to! This is because the default action of the method <code>Files.write(Path, List&lt;&gt;)</code> is to overwrite the data in files. We will see later how to append data to files if so desired.

### Buffered Files

4.	<p>Explain the concept of buffered reading and writing of files.</p> <p><i>The concept of buffered reading</i></p> <p>We explain by example. Imagine we have a file which we would like to read, whose sole contents are the string “The sun rises in the east” and we are mandated to read and process the data character by character from disk. The processing in pseudocode would be similar to the following:</p> <pre>while(there is another character to read) {     read_character_from_file_on_disk(); // We are reading character by character     process character; }</pre> <p>The method <code>read_character_from_file_on_disk</code> really does the following:</p> <pre>read_character_from_file_on_disk {     wait your turn among all programs waiting to read from the disk     move the disk read head to the position on disk of the next character     read what is at that position. }</pre> <p>Looking at the pseudocode above, we see that this sequence of operation is relatively slow!</p> <p>One solution to this slow data access is that when a request is made to read data from a file, a reasonable amount of the file should be read into a space in memory instead of character by character reading. When each request for a character is made, the <code>read_character_from_file_on_disk()</code> method would then go to the space in memory to fetch the next character, rather than going to the disk. The area of memory where the portion of the file is read into is called a <i>buffer</i>. Then our method <code>read_character_from_file_on_disk()</code> would now look like the following:</p> <pre>read_character_from_file_on_disk{     are desired file contents already in buffer?     no → then read a reasonable portion of the file into the buffer, starting from the           desired character     return character from buffer }</pre> <p>This is the concept of buffered reading.</p> <p>This is a standard in disk reading. As a general rule, operating systems read whole disk sections into memory at once.</p> <p><i>The concept of buffered writing</i></p> <p>The concept of buffered writing is similar. Similar to the issue with reading files from disk, it is not optimal to write each data item we wish to write to disk at the exact time that the program says that it should be written; rather, when an appropriate amount of data is gathered to be written, then we write the data to the disk at that time. An appropriate analogy is that of a bus waiting to gather a sufficient number of passengers before moving on to the destination; clearly moving each passenger to the destination one by one is not efficient. The bus will</p>
----	---

	<p>not move until it is full, or has an appropriate number of customers, or some other constraint compels it to move at once (see the options <code>SYNC</code> and <code>DSYNC</code> in exercise 11).</p> <p>Buffered reading and writing are appropriate in cases where we do not or cannot read/write the whole file from/to disk in one go as with the methods shown earlier.</p>
5.	<p><i>Reading Buffered Files</i></p> <p>Explain in detail, the objective, function and reasoning of the numbered sections of code in the code fragment shown below.</p> <pre>#1  String filename = "rhyme.txt";     Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", filename);      String nextLine; #2  BufferedReader nbr = Files.newBufferedReader(sourcePath); #3  while((nextLine = nbr.readLine()) != null){         System.out.println(nextLine);     } #4  nbr.close();</pre> <p>#1. The code labeled #1 is simply obtaining a <code>Path</code> object named <code>sourcePath</code> for a given file on the <code>Desktop</code> that is named in the <code>String</code> variable <code>targetFile</code>.</p> <p>#2. The static method <code>Files.newBufferedReader(Path)</code> returns a <code>BufferedReader</code> object that is associated with the <code>Path</code> object that is passed to it.</p> <p>#3. The contents of a <code>BufferedReader</code> object can be read line by line, using the <code>readLine</code> method of the <code>BufferedReader</code> class, as is done in this code. In this section of the code we are saying, “keep looping while we can successfully read another line from the <code>BufferedReader</code> object”. We print each line that is read out to the console. The return value from <code>readLine</code> is <code>null</code> when there is nothing else to be read.</p> <p>It should be noted that the class <code>BufferedReader</code> actually belongs to the package <code>java.io</code> which belongs to the older Java mechanism for file I/O; however this class is still valid for NIO file operations.</p> <p>#4. This line causes the release of all resources associated with the <code>BufferedReader</code> object. One must always close <code>BufferedReader/Writer</code> objects when finished with them.</p> <p>In summary, we have just a short number of things to do to write data to a file using a <code>BufferedWriter</code>. These things are: (1) obtain a <code>Path</code> object to the source file location, (2) obtain a <code>BufferedReader</code> object against the <code>Path</code> in question, (3) read the desired data from the <code>BufferedReader</code> object and (4) close the <code>BufferedReader</code> object.</p> <p><b>Notes:</b> (Re #2) The method <code>Files.newBufferedReader(Path)</code> is actually just a NIO wrapper method for the method <code>BufferedReader</code> of class <code>java.io.BufferedReader</code> of Java’s pre-NIO file handling mechanism. The <code>newBufferedReader</code> takes a <code>Path</code> object. Due to the fact that this is simply a wrapper method, that line can be written in the old style as:</p> <pre>BufferedReader nbr = new BufferedReader(new FileReader(sourcePath.toString()));</pre> <p>However, the new means of writing the same code as shown in line #2 is more appropriate when using the NIO/NIO.2 package.</p>
6.	<p>Enhance the code in the preceding exercise to make it fully functional. Put it in a class named <code>FileHandling</code>. Test the code by manually creating a file named <code>rhyme.txt</code> (put your favorite nursery rhyme into the file) on the <code>Desktop</code> and then use your program to read the file and print its contents out to the console.</p> <pre>import java.io.BufferedReader; import java.io.IOException; import java.nio.file.*;  public class FileHandling {     public static void main(String[] args) {          String fileToRead = "rhyme.txt";         Path sourcePath = Paths.get(System.getProperty("user.home"), "Desktop", fileToRead);         String nextLine;         BufferedReader nbr = null;          try{             nbr = Files.newBufferedReader(sourcePath);             while((nextLine = nbr.readLine()) != null){                 System.out.println(nextLine);             }         } catch (IOException e) {             e.printStackTrace();         } finally {             if (nbr != null) {                 try {                     nbr.close();                 } catch (IOException e) {                     e.printStackTrace();                 }             }         }     } }</pre>

## *Practice Your Java Level 1*

	<pre>         }         nbr.close();     }     catch(IOException e){         System.out.println("IOException = " + e);     } } </pre>
7.	<p><i>Writing Buffered Files</i></p> <p>Explain in detail, the objective, function and reasoning of the numbered sections of code in the code fragment shown below.</p> <pre> #1 String targetFile = "out.txt"; Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", targetFile);  #2 BufferedWriter nbw = Files.newBufferedWriter(targetPath); #3 nbw.write("The sun rises in the East"); #4 nbw.write(System.getProperty("line.separator")); #5 nbw.write("The sun settles in the West"); #6 nbw.close(); </pre> <p>#1. This code is simply obtaining a <code>Path</code> object named <code>targetPath</code> for a given file on the <code>Desktop</code> that is specified in the <code>String</code> variable <code>targetFile</code>.</p> <p>#2. The static method <code>Files.newBufferedWriter(Path)</code> returns a <code>BufferedWriter</code> object that is associated with the <code>Path</code> object that is passed to it.</p> <p>#3. We are using the <code>write</code> method of the <code>BufferedWriter</code> class to write a <code>String</code> to the target file.</p> <p>#4. We are using the <code>write</code> method of the <code>BufferedWriter</code> class to write a line separator to the file. The line separator is not necessarily "<code>\n</code>", that is why it is safe to explicitly use the <code>line.separator</code> property.</p> <p>#5. We are using the <code>write</code> method of the <code>BufferedWriter</code> class to write another <code>String</code> to the target file.</p> <p>#6. This line causes the release of all resources associated with the <code>BufferedWriter</code> object. It should always be used when one has finished operations with the <code>BufferedWriter</code> objects.</p> <p>In summary, we have just a short number of things to do to write data to a file using a <code>BufferedWriter</code>. These things are: (1) obtain a <code>Path</code> object to the target file location, (2) obtain a <code>BufferedWriter</code> object against the <code>Path</code> in question, (3) write the desired data to the <code>BufferedWriter</code> object and (4) close the <code>BufferedWriter</code> object.</p> <p><b>Notes:</b> (Re #2) The method <code>Files.newBufferedWriter(Path)</code> is actually just a NIO wrapper method for the method <code>BufferedWriter</code> of class <code>java.io.BufferedWriter</code> of Java's pre-NIO file handling mechanism. The <code>newBufferedWriter</code> takes a <code>Path</code> object. Due to the fact that this is simply a wrapper method, that line can be written in the old style as:</p> <pre>BufferedWriter nbw = new BufferedWriter(new FileWriter(targetPath.toString()));</pre> <p>However, the new means of writing the same code as shown in line #2 is more appropriate when using the package NIO.</p>
8.	<p>Enhance the code in the preceding exercise to make it fully functional. Put it in a class named <code>FileHandling</code>.</p> <pre> import java.io.BufferedReader; import java.io.IOException; import java.nio.file.*;  public class FileHandling {     public static void main(String[] args) {          String fileToWrite = "out.txt";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", fileToWrite);          try{             BufferedWriter nbw = Files.newBufferedWriter(targetPath);             nbw.write("The sun rises in the East");             nbw.write(System.getProperty("line.separator"));             nbw.write("The sun settles in the West");             nbw.close();         }         catch(IOException e){ </pre>

	<pre>         System.out.println("IOException = " + e);     } } } </pre>
9.	<p>If immediately after running the program in the preceding exercise I run the program below, what is the resulting content of the file <code>out.txt</code>?  (only the new <code>try</code> block in <code>main</code> is shown here; all other aspects of the program remain the same as in the preceding exercise).</p> <pre> try{     BufferedWriter nbw = Files.newBufferedWriter(targetPath);     nbw.write("What happens from the North then?");     nbw.close(); } </pre> <p>The content of the file is now:  <b>What happens from the North then?</b></p> <p>What has happened is that the contents of the file are actually <u>overwritten</u> by the new program.</p>
10.	<p><i>Appending to files</i>  Rerun exercise 8 to reset the contents of the file. Now modify the program in exercise 9 as shown below and comment on the contents of the file <code>out.txt</code>.  (only the new <code>try</code> block in <code>main</code> is shown here; all other aspects of the program remain the same as in the preceding exercise).</p> <pre> try{     BufferedWriter nbw = Files.newBufferedWriter(targetPath);     nbw.write(System.getProperty("line.separator"));     nbw.append("What happens from the North then?"); //we are now using the append method     nbw.close(); } </pre> <p>The content of the file is now:  <b>The sun rises in the East</b>  <b>The sun settles in the West</b>  <b>What happens from the North then?</b></p> <p>As can be seen, in contrast to exercise 9, the last line “<i>What happens from the north then?</i>” is <i>appended</i> to the existing contents of the file.</p> <p>The objective of this exercise was to show the use of the method <code>append</code> for writing to files.</p>

So far we've looked at two different pairs of mechanisms for reading and writing files. We've also looked at the concept of appending to files. We will now look at the `StandardOpenOption` enumeration, which when used to open file buffers facilitates greater control over file I/O.

11.	<p>The different values of the enumeration <code>StandardOpenOption</code>, when applied at the same time that a buffer is instantiated for a file, determines how the file is written to or read from. Explain what each of the following <code>StandardOpenOption</code> enumeration options implies: (1) <code>APPEND</code>, (2) <code>CREATE</code>, (3) <code>CREATE_NEW</code>, (4) <code>READ</code>, (5) <code>DSYNC</code>, (6) <code>SYNC</code>, (7) <code>TRUNCATE_EXISTING</code> and (8) <code>WRITE</code>.</p> <p><b>APPEND</b> – This option means that whatever data is going to be written to the file should be added to the end of the file. Looking back at exercise 9, if we had opened that file buffer with this option, then the data in the file would not have been overwritten, but rather appended to.</p> <p><b>CREATE</b> – This option states that if the intended file does not already exist then it should be created. If the file already exists it is okay too.</p> <p><b>CREATE_NEW</b> - This option states that if the intended file does not already exist then it should be created. However, if the file already exists generate an exception.</p> <p><b>READ</b> – This indicates that we want to read data from the file. This mode is implicitly used when using a <code>BufferedReader</code>.</p> <p><b>DSYNC</b> – This mode, used when writing files, requires that whenever the file is modified (through the appropriate methods such as <code>write/append</code>) the change must be written to disk immediately, rather than being buffered and</p>
-----	--

## Practice Your Java Level 1

	<p>written at a time determined by the buffering algorithm. The advantage is better data integrity (disk-wise), however this results in slower performance than when the data is buffered. Think of <b>DSYNC</b> as “Data-SYNC”. <b>SYNC</b> – This mode, is an enhancement of <b>DSYNC</b>; in addition to the properties of <b>DSYNC</b> (which concerns the file data only), this mode insists that metadata for the file be written to disk as soon as it is changed.</p> <p><b>TRUNCATE_EXISTING</b> – When used for a file that we intend to write to, this mode wipes out the existing contents of the file.</p> <p><b>WRITE</b> – This is an indication that we are opening the file for write access.</p>	
12.	<p><i>Applying file permissions on writing for a BufferedWriter</i></p> <p>Redo exercise 9, this time opening the file with the option <b>APPEND</b>. Before running your code, first rerun exercise 8 to reset the contents of the file.</p> <p><i>Note:</i> The reason for this exercise is to show that when the option <b>APPEND</b> is used when a <b>FileBuffer</b> is being opened and the <b>write</b> method is used, the data will be appended to the end of the file, rather than the data in the file being overwritten.</p> <pre>import java.io.BufferedWriter; import java.io.IOException; import java.nio.file.Files; import java.nio.file.Path; import java.nio.file.Paths; import java.nio.file.StandardOpenOption;  public class FileHandling {      public static void main(String[] args){         String fileToWrite = "out.txt";         Path targetPath = Paths.get(System.getProperty("user.home"), "Desktop", fileToWrite);          try{             BufferedWriter nbw = Files.newBufferedWriter(targetPath, StandardOpenOption.APPEND);             //This enumeration requires the following import: java.nio.file.StandardOpenOption;             nbw.write(System.getProperty("line.separator"));             nbw.write("What happens from the north then?");             nbw.close();         }         catch(IOException e){             System.out.println("IOException = " + e);         }     } }</pre>	
13.	<p>Explain the difference in function between the following lines of code:</p> <pre>BufferedWriter nbw =     Files.newBufferedWriter(targetPath, StandardOpenOption.APPEND); and BufferedWriter nbw =     Files.newBufferedWriter(targetPath, StandardOpenOption.WRITE, StandardOpenOption.APPEND);</pre> <p>There is no difference. The reason is that a <b>BufferedWriter</b> is opened for writing anyway, therefore explicitly stating that it should use the <b>WRITE</b> option has no effect on the code.</p>	
14.	<p>What is the result if you initialize a <b>BufferedWriter</b> with the <b>StandardOpenOption APPEND</b> for a file that is <i>not</i> already existent?</p>	An exception, <a href="#">java.nio.file.NoSuchFileException</a> , will be thrown.
15.	<p>What is the result if you initialize a <b>BufferedWriter</b> with the <b>StandardOpenOption CREATE</b> on an already existing file?</p>	The file will be overwritten.
16.	<p>What is the result if you initialize a <b>BufferedWriter</b> with the <b>StandardOpenOption CREATE_NEW</b> on an already existing file?</p>	An exception, <a href="#">java.nio.file.FileAlreadyExistsException</a> , will be thrown.
<i>Writing/ Reading data as bytes</i>		
17.	<p>What is the difference between the pairs of methods <b>Files.write(Path, String[])</b>/<b>Files.readAllLines</b> and <b>Files.write(Path, byte[])</b>/<b>Files.readAllBytes</b>?</p>	

	<p>The pair of methods <code>Files.write(Path, String[])</code>/<code>Files.readAllLines</code> are used to write/read <i>textual/human-readable strings</i> from a file line by line, whereas the methods <code>Files.write(Path, byte[])</code>/<code>Files.readAllBytes</code> write/read data from files simply as bytes. Exercise 1 is an example of the method <code>readAllLines</code> being used to read textual data. The methods <code>Files.write(Path, byte[])</code>/<code>readAllBytes</code> are used for non-textual data, examples of which would be jpeg files, mpeg files, png files, mp3 files and suchlike (these are clearly not textual data) which are simply bytes of data that are interpreted by the respective programs that understand how to interpret such. For example if you wrote a program with which to play mp3 files, you would use the method <code>Files.readAllBytes</code> to read an mp3 file into a <code>byte[]</code>.</p> <p><b>Note:</b> In actuality all data is stored as bytes, however <code>Files.readAllLines</code> has already applied an interpretive layer to the data being read to convert it to human readable form; its corollary method <code>Files.write(Path, String[])</code> applies the inverse of that interpretive layer to convert the textual/human-readable data to raw bytes. Common schemes/formats for converting textual data to bytes include ASCII and UTF-16.</p> <p>Stated another way, we can actually say that because textual data is so common, the creators of Java chose to create an interpretive layer for textual data within the methods <code>Files.write(Path, String[])</code>/<code>Files.readAllLines</code>. For other non-textual data (mp3 data, jpeg, etc), you have to create your own interpretive layer (which for example could be a jpeg viewer program or an mp3 player program).</p> <p>As stated earlier all data is stored in bytes and therefore, you can actually use <code>Files.readAllBytes</code> to read a textual file which was written by the method <code>Files.write(Path, String[])</code>; you would just have to interpret the bytes yourself.</p>
18.	<p>We know that the pair of NIO methods <code>Files.newBufferedWriter</code>/<code>Files.newBufferedReader</code> are used to write/read textual file data in a buffered manner. What are the corresponding methods for writing/reading <code>byte[]</code> data in a buffered manner?</p> <p>There are currently two ways to write/read buffered byte data to/from file in Java. These are:</p> <ol style="list-style-type: none"> <li>1. To use the <code>ByteBuffer</code> and <code>newByteChannel</code> functionality provided in NIO.</li> <li>2. To use the <code>BufferedInputStream</code>/<code>BufferedOutputStream</code> in the old <code>java.io</code> package</li> </ol> <p>Note: In this book, no further investigation is made of the NIO <code>ByteBuffer</code> functionality.</p>
E1	Using a <code>BufferedReader</code> and <code>BufferedWriter</code> , copy a file of choice to another file.
E2	Write a program which creates 50 empty files in a directory of your choice with the naming scheme <code>JPG0001</code> to <code>JPG0050</code> .
E3	I downloaded some pictures from a camera into a given directory on the <i>Desktop</i> . The camera named the files <code>JPGP0001...JPGP0050</code> . I then deleted one of the files, thus creating a hole in my numbering scheme. Write a program which shifts the names of the files after the one I deleted up by one, so that there is no longer a hole in my numbering scheme (Thus we will now have files named <code>JPG0001</code> to <code>JPG0049</code> ).



# Chapter 31. Enumerations II

This chapter, entitled *Enumerations II*, is a continuation of the chapter *Enumerations I*. In *Enumerations I*, exercises which ensured an understanding of the basic functioning of enumerations in Java was presented. This chapter extends the previous enumeration chapter by presenting exercises which pertain to the extended capabilities of enumerations in Java.

From the previous chapter on enumerations we know that in Java enumerations are classes and thus have most of the features of classes. The exercises in this chapter focus on expanding our skill and understanding enumerations more fully as the classes and instances that they are.

1. Explain what the code shown below is doing and what its implications are:

```
public enum Gender {  
    MALE(10), FEMALE(20);  
  
    private final int value;  
  
    public int getValue(){  
        return(value);  
    }  
  
    private Gender(int value){ //constructor  
        this.value = value;  
    }  
}
```

It was stated in the chapter *Enumerations I* that an enumeration class is a special kind of class which has all the objects that can be instantiated from it predefined within the class definition itself. It was also stated that objects thereof are considered to have an ordinality that is directly derived from their order of declaration within the enumeration class in question.

Since we stated that enumerations are objects of their particular enumeration class, like any class, an enumeration class can have constructors. Again, because enumerations are objects, the objects thereof can have initialization parameters passed to their constructors. Thus we see in this example two enumeration objects *MALE* and *FEMALE*, each being presented with the values that we want passed to their constructor; in the case of the enumeration object *Gender.MALE*, we want it to pass the value *10* to its constructor and for the enumeration object *Gender.FEMALE*, we want it to pass the value *20* to its constructor.

Again we see a unique thing about enumeration objects; their constructor values are hardcoded into the class definition!

Now, like for any other class, the initialization parameters can be used as we choose; in this case, we happen to be storing them in a variable within the class. Also a method named *getValue* has been created with which to access this variable.

**Again, think of an enumeration as just a special kind of class whose objects and their constructor parameters are specified within it at class creation time.** You can actually liken enumeration classes to a variant of a singleton pattern!

If we want to use any of the enum object initialization parameters to represent the relative ordering of the enum objects, we would have to override the *compareTo* method or do order comparisons using a method that we ourselves have created. For instance in this example we can use the *getValue* method that we created to do order comparisons. If we choose to use a method that we ourselves create to do order comparisons, then the natural ordinality of the enum objects is irrelevant to us and therefore we can insert objects into the enum class definition in any order that we choose.

Again, an enumeration is a class all of whose objects and their constructor initialization parameters are hardcoded into it.

**Note:** If this is a bit confusing, what you can do when creating an enumeration class is to first create the body of the class as with any other class definition and then lastly, add to the class the enumeration objects that you want created.

## Practice Your Java Level 1

2.	<p>Modify the enum <code>Gender</code> of exercise 1 above to use a <code>String</code> initialization value of “man” for the enum entry <code>MALE</code> and “woman” for the enum entry <code>FEMALE</code>.</p> <pre><code>public enum Gender {     MALE("man"), FEMALE("woman");      private final String value; //we changed this to a String      public String getValue(){         return(value);     }      private Gender(String value){         this.value = value;     } }</code></pre>
	<p><b>Note:</b> One objective of this exercise was for you to see that you can use any value (in this case a <code>String</code>) as an initialization value for an enum, as enumeration classes are very similar to regular classes.</p>
3.	<p>The continents and their areas are given as follows:  The list of continents, along with their sizes (in square kilometres) expressed in exponential form is as follows: (Europe, <math>1.018 \times 10^8</math>), (North America, <math>2.449 \times 10^8</math>), (Africa, <math>3.037 \times 10^8</math>), (Asia, <math>4.382 \times 10^8</math>), (Australia, <math>9.0085 \times 10^7</math>), (South America, <math>1.7840 \times 10^8</math>) and (Antarctica, <math>1.372 \times 10^8</math>). Create an enum named <code>Continents</code>, which will store the area and name of the continents in fields named <code>area</code> and <code>name</code> respectively and have methods named <code>getArea</code> and <code>getName</code> to return the area and name of the continent when asked.</p> <p><i>Hint: Like any object, an enum can have multiple initialization parameters for its constructor and can have multiple fields; again, an enum is basically a class with objects and their respective constructor parameters hardcoded in the class.</i></p>
	<pre><code>public enum Continents {     EUROPE(1.018e8, "Europe"),           NORTH_AMERICA(2.449e8, "North America"),     AFRICA(3.037e8, "Africa"),            SOUTH_AMERICA(1.784e8, "South America"),     AUSTRALIA(9.0085e7, "Australia"),     ANTARCTICA(1.372e8, "Antarctica");      private final double area;     private final String name;      public double getArea(){         return(area);     }      public String getName(){         return(name);     }      private Continents(double area, String name){         this.area = area;         this.name = name;     } }</code></pre>
4.	<p>Override the inherited <code>toString</code> method of the enumeration class <code>Continents</code> declared above. Have the <code>toString</code> method print out the following: &lt;name&gt;, size = &lt;size&gt;.</p>
	<p>Add this method to the enum <code>Continents</code>:</p> <pre><code>public String toString(){return(this.name + ", size = " + this.area);}</code></pre>
5.	<p><i>True or False:</i> An <code>enum</code> can have a subclass.</p>
	<p><i>False.</i> An <code>enum</code> cannot be a superclass.</p>
6.	<p><i>True or False:</i> An <code>enum</code> can implement an interface.</p>
	<p><i>True.</i></p>
7.	<p>We have an enumeration that represents salary bands, each enumeration object representing a salary range, which overlap from band to band in some cases. Create an enum <code>SalaryBand</code> and initialize each of the bands to the lower and upper limits specified below:</p>
	<p>BAND1 = 30,000 - 37,000, BAND2 = 33,000 - 45,000, BAND3 = 43,000 - 55,000, BAND4 = 50,000 - 60,000,</p>

BAND5 = 57,000 - 70,000, BAND6 = 67,000 - 80,000, BAND7 = 77,000 - 90,000, BAND8 = 88,000 - 100,000, BAND9 = 98,000 - 120,000, BAND10=120,000 - 150,000.

Also create the following methods:

1. A method `getBand` which will return which band a given salary amount falls into; if the salary amount appears in two different bands, then the method should return the lower band.
2. A method `getUpperLimit` which returns the upper limit of a band passed to it.
3. A method `getLowerLimit` which returns the lower limit of a band passed to it.

```
import java.math.BigDecimal;

enum SalaryBand {
    BAND1(30000, 37000), BAND2(33000, 45000), BAND3(43000, 55000), BAND4(50000, 60000),
    BAND5(57000, 70000), BAND6(67000, 80000), BAND7(77000, 90000), BAND8(88000, 100000),
    BAND9(98000, 120000), BAND10(120000, 150000);

    private BigDecimal lowerLimit;
    private BigDecimal upperLimit;

    public BigDecimal getLowerLimit() {return(lowerLimit);}
    public BigDecimal getUpperLimit() {return(upperLimit);}

    static SalaryBand getBand(BigDecimal valueToCheck) {
        //get the list of objects and check if the amount is in their bands
        for(SalaryBand x:SalaryBand.values())
        {
            BigDecimal lower = x.getLowerLimit();
            BigDecimal upper = x.getUpperLimit();
            if((valueToCheck.compareTo(lower)>=0) && (valueToCheck.compareTo(upper)<=0))
                return(x); //implicit exit from the method
        }
        //otherwise we don't have the number. Return null.
        return(null);
    }

    private SalaryBand(int lower,int upper) {
        lowerLimit = new BigDecimal(lower);
        upperLimit = new BigDecimal(upper);
    }
}
```



# Chapter 32. Graphical User Interfaces using Swing

This chapter presents basic exercises on graphical user interface implementation using the Java Swing library.

Graphics in Java is an extensive topic, exercises on which can easily fill multiple volumes. Also, it is quite likely that you will be creating the front end of graphical applications using a GUI builder, thus sidestepping some issues in GUI layout (for example details of Layout managers). The foregoing notwithstanding, there is benefit in having some understanding of what the GUI builders are doing. This chapter presents a carefully chosen set of exercises that confirm your basic understanding, of the structure of GUI applications, GUI components and event listeners in Java using Swing.

1.	Describe in general terms each of the following Java graphics packages and their relationship: AWT, Swing & JavaFX.  <b>Abstract Windowing Toolkit</b> , also known by its acronym AWT, is the original graphics package shipped with Java. It delegates a lot of its graphical component painting to the underlying operating system and therefore AWT applications have the look and feel of the operating system that they are running on. <b>Swing</b> , which itself is built on top of AWT, debuted in a later version of Java (version 1.2). It presents more advanced built-in graphical components than AWT. Swing itself is written in Java. Swing also has more control over the actual drawing and painting of components and thus their look and feel, thereby allowing you to specify a common look and feel for your application across all platforms. <b>JavaFX</b> is currently the next generation Java graphical toolkit. It is based on a scene graph model. A markup language, FXML, is provided for laying out JavaFX components. It also provides even more powerful built-in components than Swing (for example it also includes a chart library). The look and feel of JavaFX applications can also be customized using Cascading Style Sheets(CSS). JavaFX's design makes it easier to provide animations within your application. Also, if desired, the user can also embed JavaFX features within a Swing application through the Swing component <b>JFXPanel</b> .
2.	In general terms, what are “components” in Swing?  A component in Swing means a “graphical component” of your user interface. Well-known examples of built-in graphical components include buttons (push buttons), text areas (there are a number of different ones), radio buttons, check boxes, list boxes, combo boxes, scroll bars, menus, file choosers and tooltips among others. Each of these components can “listen” for user interaction with them and report this to the application for the appropriate processing. You can also built your own custom Swing components.
3.	Describe in broad terms the general components of and how to build a graphical application in Java Swing. In your description touch on the concepts of Frames, Panels, Layouts and Event Listeners.  A Java Swing application exists in a frame (think of this as a picture frame); this is realized by an object of class <b>JFrame</b> . A <b>JFrame</b> object has a number of distinct subareas, the ones of immediate pertinence being (1) the application frame itself (where the application title, program icon and the minimize/maximize/close buttons are), (2) a menu bar area (the use of which is optional) and (3) the content pane, <i>generally</i> of class <b>JPanel</b> , which is where the application components (such as buttons, pulldown menus, text areas, check boxes and suchlike) are. There is a default object assigned as the content pane object for a <b>JFrame</b> , you obtain it by the following code:  <code>&lt;JFrame&gt;.getContentPane();</code> You are free to replace the default content pane object with your own <b>JPanel</b> object. You would do that in the following manner:  <code>&lt;JFrame&gt;.setContentPane(new JPanel(...));</code>  <i>Layouts</i> How do you determine the layout of different components within a <b>JPanel</b> ? One way is to manually specify the individual positioning of components by $(x, y)$ coordinates, or to use one of the “layout managers” that Swing has provided to help ease the component layout process. Each of these layout managers has different

## Practice Your Java Level 1

	<p>characteristics.</p> <p><i>Event Listeners</i></p> <p>There are a number of built-in interfaces in Java whose task it is to facilitate the recognition and receipt of user input from Java GUI's. These are the event listener interfaces. For example, if you click on a button component using a mouse, your application can, if it has implemented the <code>MouseListener</code> interface, recognize which element has been clicked on by the mouse and direct the appropriate action within your application as you have programmed it to.</p> <p>The event listeners are actually quite easy to program.</p> <p>We summarize the above as follows: Each application exists within a frame. The frame has three areas, namely the application frame, the menu bar area and a content frame. Individual components are laid out in the content frame, either directly by specifying <math>(x, y)</math> coordinates, or by using a "layout manager". Each of the components can implement appropriate "event listeners". The event listener methods listen for user interaction and inform the application as to what the user is doing/attempting to do with the application so that your application can respond appropriately.</p> <p><b>Note:</b> In addition to the content pane, a <code>JFrame</code> has three other panes, namely the root pane, the layered pane and a pane called the glass pane. These three are not immediately important for basic construction of Swing applications. The order of the four panes from lowest to topmost is: root, layered, content and glass panes.</p>
4.	<p>Explain briefly the concept of event driven programming. Touch on the concept of event listeners.</p> <p>This is actually a concept that you are familiar with by practice.</p> <p>Event driven programming simply means a program which waits for user events to occur and then the program responds based on these events. Examples of these events include pressing a push button, scrolling a scroll bar, typing into a text area and suchlike. Each graphical component can be programmed to "listen" for specific events.</p> <p>To "hear" events, the application has to "listen" to the user input devices, mainly the keyboard and the mouse. Each graphical component that has registered to "listen" to the input devices will then be prompted when the device is utilized. A component has to be handled by the application event listener itself or directly extend the event listener interfaces through anonymous classes in order to listen for when it is impacted by the user input devices.</p> <p>Java has implemented a number of different event listener interfaces which each component that is desirous of listening for user input can implement. Each different type of event listener interface listens for different things and can pass "what it hears" to whoever is listening. For example, a <code>MouseListener</code> can tell you if the mouse is moving and where it is, a <code>ListSelectionListener</code> can tell you which item has been selected on a list.</p> <p>The above can be summarized as follows: each component that wants to listen for user input can listen using appropriate event listener interfaces and can react as desired.</p>
5.	<p>Below is the structure of a basic GUI application:</p> <pre>import javax.swing.*; public class graphicsPractice {     public void createGUI() {         JFrame frame01 = new JFrame("My Application Title");         frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          frame01.setVisible(true);     }      public static void main(String args[]) throws Exception{         SwingUtilities.invokeAndWait(new Runnable(){             public void run(){                 new graphicsPractice().createGUI(); //Here we create an object of the class   //and call a method in it on the same line.             }         }); //an anonymous class implementing an interface here     } }</pre>

	<p>Run the application and explain what it is doing. Also, explain why the application needs to be run in a thread.</p> <p>We look first at the method <code>createGUI</code>.</p> <p>In this method we create a <code>JFrame</code> object. It will be recalled from exercise 3 above that every graphical application starts off with a <code>JFrame</code> object. Next, we specify what should happen to the application when the close button on the application window is selected by the user; our choice is indicated by the constant <code>JFrame.EXIT_ON_CLOSE</code>. Following this, we direct the application to make the frame visible.</p> <p>GUI applications are run on what is known as the Swing “event dispatch thread”. The way the GUI application is run on the event dispatch thread is by an instance of the GUI class being invoked in the method <code>run</code> of a class that implements the <code>Runnable</code> interface. We see in our example above an anonymous class which implements <code>Runnable</code>. We implemented the interface method <code>run</code> and in this method we create an instance of our GUI class and kick off the method which creates our GUI.</p> <p>The “event dispatch thread” is responsible for proper scheduling and handling of event notifications in graphical applications.</p> <p><b>Observations:</b> On running the program, you see that the application window started with length and width of 0 and 0 pixels! This is because we did not specify a size for it using the method <code>&lt;JFrame&gt;.size(width, height)</code>.</p> <p><b>Note:</b> We have presented the use of threading in Java GUI’s here, using the interface <code>Runnable</code>. This rudimentary knowledge of threading is sufficient for our use in this text.</p>
6.	<p>Describe in general terms the different layout managers in Java Swing.</p> <p>It was stated earlier (see solution to exercise 3), that Swing provides the concept of layout managers to help ease the task of placing components in an ordered way into panels. Swing provides the following eight (8) layout manager classes:</p> <p><b>BorderLayout</b> – this manager defines the panel as 5 areas, namely top, bottom, left, right and center. <u>This is the default layout manager</u>. The constants that characterize each of these areas are <code>BorderLayout.PAGE_START</code>, <code>BorderLayout.PAGE_END</code>, <code>BorderLayout.LINE_START</code>, <code>BorderLayout.LINE_END</code> and <code>BorderLayout.CENTER</code> respectively.</p> <p><b>BoxLayout</b> – This is a layout manager that lays the desired components out in either a top/down order, or a left to right order (the constants for these respectively are <code>BoxLayout.PAGE_AXIS</code> and <code>BoxLayout.LINE_AXIS</code>).</p> <p><b>CardLayout</b> – This layout manager actually lets you put present multiple panels as candidates for a given frame; you can then select which panel you want displayed at any particular time. As the name alludes to, this layout manager is actually one which like playing cards in a stack has only the current top card (the top panel in this case) visible at any given time.</p> <p><b>FlowLayout</b> – The flow layout manager puts objects into a panel from left to right, expanding to more rows if necessary. It ensures that each row is centered.</p> <p><b>GridLayout</b> – This layout container divides the panel into equal sized grids. Each component is put into a space on the grid. This layout makes each component fill its grid position, i.e. each component is scaled to be the same size.</p> <p><b>GridBagLayout</b> – This layout manager has grids, yes, however, not each row has to be the same height and a given component can span many grid positions.</p> <p><b>GroupLayout</b> – This layout manager allows the user to specify groups of items to layout at the same time, requiring that the user specify both the horizontal grouping and the vertical grouping of the group of items.</p> <p><b>SpringLayout</b> – In this layout manager, component positioning is defined by its vertical and horizontal distance from other components.</p> <p><b>Absolute positioning:</b> In addition to the layout managers, you can place the components directly on a panel by <math>(x, y)</math> position. This is called absolute positioning.</p> <p>In addition to the above, Java allows you to create your own custom layout managers.</p>
7.	<p>Can you put <code>JPanel</code> objects into another <code>JPanel</code> object?</p> <p>Yes you can!</p>
8.	<p>Why might you want to put <code>JPanel</code> objects into another <code>JPanel</code> object?</p> <p>It might be optimal for your application to have different component groupings on the screen that different types of layout managers would be best for. This is why you would want different panels on a frame.</p> <p><b>Implementation:</b> Given that a frame has a single content panel, a layout manager would have to be selected for</p>

## Practice Your Java Level 1

	the content panel and then the “child-panels” would then be inserted into this layout manager just like any other component.
9.	<p><i>Sizing the application window start size</i></p> <p>Given the following code for determining the size of the screen:</p> <pre>Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize(); // the height is in screenSize.height, the width is in screenSize.width</pre> <p>modify the program in exercise 5 to set the initial size of the application to <math>\frac{1}{2}</math> the screen width and <math>\frac{1}{2}</math> the screen length.</p> <p><i>Hint: JFrame.setSize</i></p> <pre>public class graphicsPractice {     public void createGUI() {         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();         JFrame frame01 = new JFrame("My Application Title");         frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         frame01.setSize(screenSize.width/2, screenSize.height/2);         frame01.setVisible(true);     }     public static void main(String args[]) throws Exception{         SwingUtilities.invokeLater(new Runnable(){             public void run(){                 new graphicsPractice().createGUI();             }         });     } }</pre>
10.	We would like the application in the preceding exercise to be centered on the screen when it starts. Modify the application to do this. <i>Hint: JFrame.setLocationRelativeTo</i> Simply add the following line to the code before you make it visible: <code>frame01.setLocationRelativeTo(null);</code>
11.	Modify the application in exercise 9 question to appear at position (25, 25) when it starts. <i>Hint: JFrame.setLocation</i> Simply add the following line to the code before you make it visible: <code>frame01.setLocation(25, 25);</code>
12.	Modify the application in exercise 9 question to ensure that the application window is not resizable. <i>Hint: JFrame.setResizable</i> Simply add the following line to the code before you make it visible: <code>frame01.setResizable(false);</code>
<b>Individual components</b> The preceding exercises have focused on the general structure of GUI applications. We now go into exercises which use some of the built-in Swing components.	
13.	<p><b>JButton</b></p> <p>In a program with a layout manager type <code>FlowLayout</code>, create the following <code>JButton</code> objects:</p> <ol style="list-style-type: none"><li>one without any text on it</li><li>one with the text “Hello!” on it</li><li>one with the text “Bye!” in the following font: Arial, bold, italicized, size 15</li><li>one with an icon of your choice (for now, let the icon be a .png file in any directory on your computer)</li><li>one with both an icon and text of your choice</li></ol> <p><i>Hint: Classes and methods JButton, Image, ImageIO.read, Font</i> Here is an example of the creation of a Font object: <code>Font("Arial", Font.BOLD   Font.ITALIC, 15);</code> Here is an example of the creation of an Image object using a .png file as input: <code>Image img = ImageIO.read(new File("C:\\\\Users\\\\PYJ\\\\DIR_A\\\\a.png"));</code></p>

The `FlowLayout` does not resize the objects, so in using it in this exercise you can observe the default appearance of objects placed in it.

```
public class graphicsPractice {
    public void createGUI() {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        System.out.println("height = " + screenSize.height);
        System.out.println("width = " + screenSize.width);

        JFrame frame01 = new JFrame("My Application Title");
        frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel pane01 = new JPanel(new GridBagLayout());
        frame01.setContentPane(pane01);
        frame01.setSize(screenSize.width/2, screenSize.height/2);

        JButton button01 = new JButton("");
        JButton button02 = new JButton("Hello!");
        JButton button03 = new JButton("Bye!");
        Font font01 = new Font("Arial", Font.BOLD|Font.ITALIC, 15);
        button03.setFont(font01);
        JButton button04 = new JButton();
        JButton button05 = null;
        try {
            Image img = ImageIO.read(new File("C:\\\\Users\\\\PYJ\\\\Desktop\\\\a.png"));
            button04.setIcon(new ImageIcon(img));
            button05 = new JButton("Hello!", new ImageIcon(img));
        }
        catch(Exception e){
            System.out.println(e);
            System.out.println("Couldn't set the image");
        }
        pane01.add(button01);pane01.add(button02);pane01.add(button03);
        pane01.add(button04);pane01.add(button05);

        frame01.setVisible(true);
    }

    public static void main(String args[]) throws Exception{
        SwingUtilities.invokeLater(new Runnable(){
            public void run(){
                new graphicsPractice().createGUI();
            }
        });
    }
}
```

**Note:** With regard to images in a packaged application, you would normally package the images with your program and use them as a “Resource”. The topic of Resources is outside the scope of this volume.

14. `JLabel`

In a program with a layout manager type `GridBagLayout`, create the following `JButton` objects:

1. one without any text on it
2. one with the text “Hello!” on it
3. one with the text “Bye!” in the following font: Arial, bold, italicized, size 15
4. one with an icon of your choice (for now, let the icon be a `.png` file in any directory on your computer).
5. one with both an icon and text of your choice, with the icon trailing the text

*Hint: Classes and methods `JLabel`, `Image`, `ImageIO.read`, `Font`*

```
public class graphicsPractice{
    public void createGUI() {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        System.out.println("height = " + screenSize.height);
        System.out.println("width = " + screenSize.width);
```

## *Practice Your Java Level 1*

```
JFrame frame01 = new JFrame("My Application Title");
frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel pane01 = new JPanel(new GridBagLayout());
frame01.setContentPane(pane01);
frame01.setSize(screenSize.width/2, screenSize.height/2);

JLabel label01 = new JLabel("");
JLabel label02 = new JLabel("Hello!");
JLabel label03 = new JLabel("Bye!");
Font font01 = new Font("Arial", Font.BOLD|Font.ITALIC, 15);
label03.setFont(font01);
JLabel label04 = new JLabel();
JLabel label05 = null;
try{
    Image img = ImageIO.read(new File("C:\\\\Users\\\\AMA\\\\Desktop\\\\a.png"));
    label04.setIcon(new ImageIcon(img));
    label05 = new JLabel("Hello!", new ImageIcon(img), SwingConstants.TRAILING);
}
catch(Exception e){System.out.println(e); System.out.println("Couldn't set the image");}
pane01.add(label01);pane01.add(label02);pane01.add(label03);
pane01.add(label04);pane01.add(label05);

frame01.setVisible(true);
}

public static void main(String args[]) throws Exception{
    SwingUtilities.invokeAndWait(new Runnable(){
        public void run(){
            new graphicsPractice().createGUI();
        }
    });
}
```

15. JTextField

In a program with a layout manager type of `GridBagLayout`, create the following `JTextField` objects:

1. an empty JTextField 30 columns wide
  2. one with the text “Hello!”
  3. one with the text “Hello” in a field that is 30 columns wide
  4. one with the text “How are you!” in the following font: Arial, bold, italicized, size 25

Also, make this JTextField uneditable.

*Hint: Classes and methods JTextField, Font, <JTextField>.setEditable*

```
public class graphicsPractice {
```

```
public void createGUI() {  
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();  
    System.out.println("height = " + screenSize.height);  
    System.out.println("width   = " + screenSize.width);  
  
    JFrame frame01 = new JFrame("My Application Title");  
    frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    JPanel pane01 = new JPanel(new GridBagLayout());  
  
    frame01.setContentPane(pane01);  
    frame01.setSize(screenSize.width/2, screenSize.height/2);  
  
    JTextField textField01 = new JTextField(30);  
    JTextField textField02 = new JTextField("Hello!");  
    JTextField textField03 = new JTextField("Hello", 30);  
    JTextField textField04 = new JTextField("How are you?");  
    Font font01 = new Font("Arial", Font.BOLD | Font.ITALIC, 25);  
    textField04.setFont(font01);  
  
    textField04.setEditable(false);  
    pane01.add(textField01); pane01.add(textField02);
```

	<pre> pane01.add(textField03); pane01.add(textField04); frame01.setVisible(true); textField01.setText( }  public static void main(String args[]) throws Exception{ SwingUtilities.invokeAndWait(new Runnable(){     public void run(){         new graphicsPractice().createGUI();     } }); } } </pre> <p><b>Observations</b></p> <ol style="list-style-type: none"> <li>1. Note how in this layout manager the height of the <code>JTextField</code> was set to match the height of the font that it was to contain.</li> <li>2. Note that the width assigned to the field matches the length of the text assigned to it in the cases where we did not explicitly assign a width to the <code>JTextField</code>. What can be concluded from this is that if you want a particular width for a <code>JTextField</code>, you should assign that width explicitly.</li> </ol>
16.	Why is the use of a <code>JPasswordField</code> component advised for receiving passwords from users? The reason why such is advised is that the <code>JPasswordField</code> component masks data entered into it.
17.	Explain the use of the <code>JScrollPane</code> component. Certain objects can display multiple lines of text, however they do not provide any facility to scroll through the lines, thus some of the data is obscured from the user. There are two solutions to this: (1) ensure that the size of the displayed component is large enough to show all of the data it contains (this is not always feasible) or (2) put the component into a <code>JScrollPane</code> component. The <code>JScrollPane</code> has the ability to scroll through content.
18.	<code>JList</code> and <code>JScrollPane</code> Explain what a <code>JList</code> is and its relationship to <code>JScrollPane</code> . A <code>JList</code> is a component which contains and displays in a vertical fashion, one after the other, a list of items put into it. It offers the user the option to select single items, a set of contiguous items, or a set of non-contiguous items. The <code>JList</code> needs a <code>JScrollPane</code> to aid in viewing its entries if the number of entries exceeds the size of the displayed <code>JList</code> control.
19.	The following code snippet shows how to get the list of system fonts into a <code>String</code> array; <code>String fonts[] = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();</code> Display all of these fonts in a scrollable <code>JList</code> component which displays 10 items at a time. <pre> public class graphicsPractice{      public void createGUI() {          Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();         System.out.println("height =" + screenSize.height);         System.out.println("width  =" + screenSize.width);          JFrame frame01 = new JFrame("My Application Title");         frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          JPanel pane01 = new JPanel(new GridBagLayout());          frame01.setContentPane(pane01);         frame01.setSize(screenSize.width/2, screenSize.height/2);          String fontArray[] =             GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();          JList&lt;String&gt; fontList = new JList&lt;String&gt;(fontArray);         fontList.setVisibleRowCount(10);         JScrollPane fontListScrollPane = new JScrollPane();         //to get a scroll bar on a list, put it in a JScrollPane         fontListScrollPane.setViewportView(fontList);         pane01.add(fontListScrollPane);     } } </pre>

## Practice Your Java Level 1

	<pre>         frame01.setLocationRelativeTo(null);         frame01.setVisible(true);     }      public static void main(String args[]) throws Exception{         SwingUtilities.invokeAndWait(new Runnable(){             public void run(){                 new graphicsPractice().createGUI();             }         });     } } </pre>																					
20.	<p><b>JTable</b></p> <p>Put the following data, along with their titles, into a <b>JTable</b> component (which itself should be in a <b>JScrollBar</b> component) in a <b>BorderLayout</b> layout manager. Ensure that the table is not editable.</p> <p><b>Continent, Size, Population</b></p> <table border="1"> <tbody> <tr> <td>Asia</td> <td>43820000</td> <td>4164252000</td> </tr> <tr> <td>Africa</td> <td>30370000</td> <td>1022234000</td> </tr> <tr> <td>North America</td> <td>24490000</td> <td>542056000</td> </tr> <tr> <td>South America</td> <td>17840000</td> <td>392555000</td> </tr> <tr> <td>Antarctica</td> <td>13720000</td> <td>4500</td> </tr> <tr> <td>Europe</td> <td>10180000</td> <td>738199000</td> </tr> <tr> <td>Australia</td> <td>9008500</td> <td>29127000</td> </tr> </tbody> </table> <pre> public class graphicsPractice {     public void createGUI() {         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();         JFrame frame01 = new JFrame("My Application Title");         frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         JPanel pane01 = new JPanel(new BorderLayout ());         frame01.setContentPane(pane01);         frame01.setSize(screenSize.width/2, screenSize.height/2);         Object[] tableHeadings = {"Continent","Size","Population"};         Object[][][] tableData = {             {"Asia", 43820000, 4164252000L},             {"Africa", 30370000, 1022234000L},             {"North America", 24490000, 542056000L},             {"South America", 17840000, 392555000L},             {"Antarctica", 13720000, 4500L},             {"Europe", 10180000, 738199000L},             {"Australia", 9008500, 29127000L}         };         DefaultTableModel model = new DefaultTableModel(tableData,tableHeadings);         JTable table01 = new JTable(model);         JScrollPane tableScrollPane = new JScrollPane(table01);         tableScrollPane.setSize(100,50);         table01.setEnabled(false);         pane01.add(tableScrollPane);         frame01.setLocationRelativeTo(null);         frame01.setVisible(true);     }      public static void main(String args[]) throws Exception{         SwingUtilities.invokeAndWait(new Runnable(){             public void run(){                 new graphicsPractice().createGUI();             }         });     } } </pre>	Asia	43820000	4164252000	Africa	30370000	1022234000	North America	24490000	542056000	South America	17840000	392555000	Antarctica	13720000	4500	Europe	10180000	738199000	Australia	9008500	29127000
Asia	43820000	4164252000																				
Africa	30370000	1022234000																				
North America	24490000	542056000																				
South America	17840000	392555000																				
Antarctica	13720000	4500																				
Europe	10180000	738199000																				
Australia	9008500	29127000																				

21.	<p>Explain the relationship between the components <code>JMenuBar</code>, <code>JMenu</code> and <code>JMenuItem</code>.  This trio of classes represent the elements that are required to (1) create a menu bar (this using <code>JMenuBar</code>, which is usually at the top of the application), create pull-downs (<code>JMenu</code>) and create the individual menu items (<code>JMenuItem</code>).</p>
22.	<p>Create a drop down menu with the items Breakfast, Lunch and Dinner. The Breakfast menu should have a submenu with the following three options: Scrambled Eggs &amp; Toast, Cereal &amp; Milk and Muffin &amp; Coffee.</p> <pre data-bbox="241 382 1535 1453"> public class graphicsPractice {     public void createGUI() {         JFrame frame01 = new JFrame("Menu example");         frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         JPanel pane01 = new JPanel(new GridBagLayout());         frame01.setContentPane(pane01);         frame01.setSize(300, 300);          //We 1<sup>st</sup> create the menubar         JMenuBar menuBar01 = new JMenuBar();         //Then we create the individual menu         JMenu menu01 = new JMenu("Meals");         menuBar01.add(menu01);         //Note: Breakfast itself is not just an item, it is a "menu" because it has sub-items         JMenu breakfastMenu = new JMenu("Breakfast");         breakfastMenu.add("Scrambled Eggs &amp; Toast");         breakfastMenu.add("Cereal &amp; Milk");         breakfastMenu.add("Muffin &amp; Coffee");         menu01.add(breakfastMenu);         //Lunch and dinner are just menu items...         JMenuItem menuItem02 = new JMenuItem("Lunch");         menu01.add(menuItem02);         JMenuItem menuItem03 = new JMenuItem("Dinner");         menu01.add(menuItem03);          frame01.setJMenuBar(menuBar01); //Putting the whole thing on the frame         frame01.setLocationRelativeTo(null);         frame01.setVisible(true);     }      public static void main(String args[]) throws Exception{         SwingUtilities.invokeAndWait(new Runnable(){             public void run(){                 new graphicsPractice().createGUI();             }         });     } } </pre>
	<p><b>Notes:</b> The interested reader should also look at the components <code>JRadioButtonMenuItem</code>, <code>JCheckBoxMenuItem</code></p>
23.	<p><code>JCheckBox</code>  Create and display a set of four <code>JCheckBox</code> items, labelled <i>Shirt</i>, <i>Shorts</i>, <i>Socks</i> and <i>Shoes</i> respectively. Use the following font: Arial, Bold and Italic, with a size of 20.  <i>(the code fragment which implements this is shown below)</i></p> <pre data-bbox="241 1628 1535 1938"> Font font01 = new Font("Arial",Font.BOLD   Font.ITALIC, 20); JCheckBox cb01 = new JCheckBox("Shirt"); JCheckBox cb02 = new JCheckBox("Shorts"); JCheckBox cb03 = new JCheckBox("Socks"); JCheckBox cb04 = new JCheckBox("Shoes"); cb01.setFont(font01);cb02.setFont(font01); cb03.setFont(font01);cb04.setFont(font01); pane01.add(cb01); pane01.add(cb02); pane01.add(cb03); pane01.add(cb04); //add the components to your panel </pre>

## Practice Your Java Level 1

24.	<p>Give a summary description of each of the following components: <b>JTextField</b>, <b>JTextArea</b> and <b>JTextPane</b>.  A <b>JTextField</b> is a component that can only display a single line of text. The background color, the font color and size can be modified and its text can be given the effects bold and italic.  A <b>JTextArea</b> component can display multiple lines of text with all of the above noted effects.  A <b>JTextPane</b> really has capabilities similar to that of a word processor. It has all of the abilities of the just described components and also supports different fonts within the same <b>JTextPane</b> object. It allows different styles to be applied the individual character elements (or groups of them), including such features as the application of underlining (which the other two do not support), different fonts and super/subscripting among other features. It can also serve as an HTML editor.  These components are all editable.</p>
25.	<p>Create two <b>JTextField</b> objects with the following characteristics respectively:  The first should contain the String “I am a JTextField”. Its width should match the width of the string.  The second should contain the String “I am the 2nd JTextField”. Its width should match the width of the string.  Other parameters of the text in the second <b>JTextField</b> are: color=red, font=Lucida Console, bold, italic, size=25.  <i>(the code fragment which implements this is shown below)</i></p> <pre>String String01 = "I am a JTextField"; String String02 = "I am the 2nd JTextField";  Font font01 = new Font("Lucida Console", Font.BOLD   Font.ITALIC, 25);  JTextField textfield01 = new JTextField(String01, String01.length()); JTextField textfield02 = new JTextField(String02, String02.length()); textfield02.setForeground(Color.RED); textfield02.setFont(font01);  //then add the components to your panel</pre>
26.	<p>Create a <b>JTextArea</b>, of dimensions 2 rows by 30 characters and put it into a <b>JScrollPane</b> component. Put into the <b>JTextArea</b> the lines for the nursery rhyme “Mary had a little lamb”. Ensure that the text is in the font Lucida Console, bold, italic and with a size of 25.  <i>(the code fragment which implements this is shown below)</i></p> <pre>String String01 = "Mary had a little lamb\nlittle lamb\nlittle lamb"; String01 = String01 + "Mary had a little lamb\nthat was as white as snow\n";  JTextArea textarea01 = new JTextArea(String01, 2, 30); Font font01 = new Font("Lucida Console", Font.BOLD   Font.ITALIC, 25); textarea01.setFont(font01); //apply the font to the JTextArea JScrollPane jsp01 = new JScrollPane(); //then add the object jsp01 to your panel</pre>
27.	<p>Create a <b>JTextPane</b> object. The contents are the contents of the nursery rhyme “Mary had a little lamb”, subject to the following requirements:</p> <ol style="list-style-type: none"> <li>1. The first line should be in superscript. The font should be Lucida Console, size 20.</li> <li>2. The second line should be red and underlined. The font should be Arial, size 20.</li> <li>3. The third line should be italic with strikethrough. The font should be Courier New, size 15.</li> <li>4. All subsequent lines should be of font Garamond, size 25, with a cyan background.</li> </ol> <p><i>(the code fragment which implements this is shown below)</i></p> <pre>String[] rhyme = new String[5]; rhyme[0] = "Mary had a little lamb"; rhyme[1] = "little lamb\n"; rhyme[2] = "little lamb\n"; rhyme[3] = "Mary had a little lamb\n"; rhyme[4] = "that was as white as snow";  JTextPane textpane01 = new JTextPane(); Document doc = textpane01.getDocument(); textpane01.setSize(200, 200); SimpleAttributeSet attr0 = new SimpleAttributeSet();  attr0.addAttribute(StyleConstants.CharacterConstants.FontFamily, "Lucida Console"); attr0.addAttribute(StyleConstants.CharacterConstants.FontSize, 20);</pre>

```

attr0.addAttribute(StyleConstants.CharacterConstants.Superscript, Boolean.TRUE);
SimpleAttributeSet attr1 = new SimpleAttributeSet();
attr1.addAttribute(StyleConstants.CharacterConstants.FontFamily, "Arial");
attr1.addAttribute(StyleConstants.CharacterConstants.FontSize, 20);
attr1.addAttribute(StyleConstants.CharacterConstants.Foreground, Color.RED);
attr1.addAttribute(StyleConstants.CharacterConstants.Underline, Boolean.TRUE);

SimpleAttributeSet attr2 = new SimpleAttributeSet();
attr2.addAttribute(StyleConstants.CharacterConstants.Foreground, Color.BLUE);
attr2.addAttribute(StyleConstants.CharacterConstants.Italic, Boolean.TRUE);
attr2.addAttribute(StyleConstants.CharacterConstants.StrikeThrough, Boolean.TRUE);
attr2.addAttribute(StyleConstants.CharacterConstants.FontFamily, "Courier New");
attr2.addAttribute(StyleConstants.CharacterConstants.FontSize, 15);

SimpleAttributeSet attr3 = new SimpleAttributeSet();
attr3.addAttribute(StyleConstants.CharacterConstants.FontFamily, "Garamond");
attr3.addAttribute(StyleConstants.CharacterConstants.FontSize, 25);
attr3.addAttribute(StyleConstants.CharacterConstants.Background, Color.CYAN);

try{
    doc.insertString(0, rhyme[0], attr0);
    doc.insertString(doc.getEndPosition().getOffset(), rhyme[1], attr1);
    doc.insertString(doc.getEndPosition().getOffset(), rhyme[2], attr2);
    doc.insertString(doc.getEndPosition().getOffset(), rhyme[3], attr3);
    doc.insertString(doc.getEndPosition().getOffset(), rhyme[4], attr3);
}
catch( BadLocationException e){
}
//then add the object textpane01 component to your panel

```

**Note:** There are also corresponding methods of the class *StyleConstants* which set these attributes as desired.

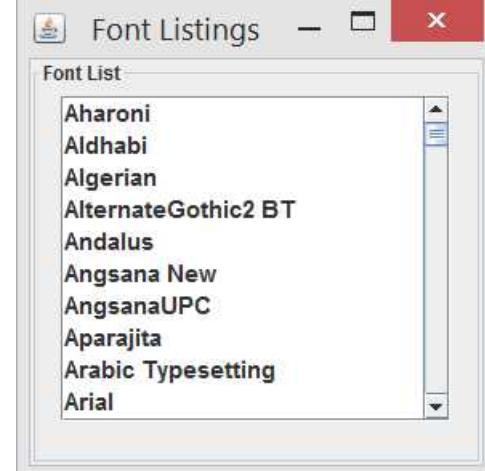
28. *A Pane within a Pane (nested panes)*

Write a program which lists all the fonts in a *JList* object (which itself is in a *JScrollBar* object).

Put this into a sub-pane to which you assign the title “Font List”.

Use the font Arial, Bold, with a size of 15 for the list content. Also, make the list a single selection list.

The expected GUI is as shown on the right.



```

public class graphicsPractice {
    public void createGUI() {
        JFrame frame01 = new JFrame("Font Listings");
        frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame01.setSize(300, 300);

        //The List of Fonts which we are going to put into the JList.
        String fontArray[] =
            GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
        JList<String> fontList = new JList<String>(fontArray);
        fontList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION); //only 1 selectable at a time
        fontList.setVisibleRowCount(10);
        fontList.setVisible(true);

        Font font01 = new Font("Arial", Font.BOLD, 15);
    }
}

```

## Practice Your Java Level 1

```
fontList.setFont(font01);

JScrollPane fontListScrollPane = new JScrollPane();
fontListScrollPane.setViewportView(fontList);
//Now put this in a pane that has a border & title
JPanel fontListPanel = new JPanel(new FlowLayout());
Border border01 = BorderFactory.createTitledBorder("Font List");
fontListPanel.setBorder(border01);
fontListPanel.add(fontListScrollPane);
frame01.add(fontListPanel);
frame01.setVisible(true);
}

public static void main(String args[]) throws Exception{
    SwingUtilities.invokeAndWait(new Runnable(){
        public void run(){
            new graphicsPractice().createGUI();
        }
    });
}
```

### 29. Event listeners

Describe the following two coding mechanisms for listening to events:

- 1) listening for events using anonymous classes applied to individual component objects, through the components' `add<x>Listener` set of methods
- 2) having the GUI class implement the listeners

Show code examples for both mechanisms for a `JButton` object that implements a `MouseListener`.

(Look at the solutions to exercises 3 and 4 for a summary on event listeners)

The differences between the two means of coding for the listening for events is: in one case, the methods for the particular event listener interface are implemented for each component individually, using anonymous classes to effect this. In the other case, there is single point of listening within the application. Doing it the second way results in all of your listening code being in the same place in your program, which you may prefer; it results in more compact code if the response for many of the objects in your GUI is the same and thus you will not end up writing the same anonymous class code for each and every object. If the behavior/response of each object to the events is completely different, it might be preferred to separate the code for each into its own anonymous class.

It should be understood that each Swing component is pre-supplied with methods for adding the objects that implement the listener interfaces that are pertinent to it. These methods are usually named `add<listener type>Listener`, for example, the method `addMouseListener` of class `JButton`. The interested reader can look at the class definition of `JButton` and other components to see the list of such listener adding methods that each component supports.

An example of each mechanism is shown below for a `JButton` object for which code is implemented for each of the methods of the `MouseListener` interface.

#### *Anonymous classes method for implementing listeners*

We show below an example of a `JButton` object which instantiates an anonymous class that implements the `MouseListener` interface:

```
JButton button01 = new JButton("Starting out");
button01.addMouseListener(new MouseListener(){
    public void mouseClicked(MouseEvent arg0) {
        button01.setText("You clicked me!");
    }
    public void mouseEntered(MouseEvent arg0) {
        button01.setText("You are hovering over me!");
    }
    public void mouseExited(MouseEvent arg0) {
        button01.setText("Phew! That mouse left!");
    }
    public void mousePressed(MouseEvent arg0) {
```

```

        button01.setText("Why are you pressing the mouse?");
    }
    public void mouseReleased(MouseEvent arg0) {
        button01.setText("Now you've released it!");
    }
});

```

*Having the GUI class implement the listener*

There are three steps using this method:

*Step 1*

The class will have to be declared as implementing the listener in question, for example as follows:

```
public class graphicsPractice implements MouseListener{
```

*Step 2*

Then you have to register for `MouseListener` events in the following simple manner:

```
<frame object>.addMouseListener(this);
```

or, if you want, register individually each object that should be listening for events, for example you can write the following for `button01`;

```
button01.addMouseListener(this);
```

*(Each implemented listener type has its own registration method. As you can see in this example we are only listening for mouse events)*

*Step 3*

Now, within the code we have to implement the same interface methods as in the previous code style.

Unless you specifically want to, you cannot just now write:

```
public void mouseClicked(MouseEvent arg0) {
    button01.setText("You clicked me!");
}
```

The reason you cannot is because when the class implements the interfaces, the interface methods respond for every single component that is on that page. We have to determine in this implementation style which object it is that was clicked. For this, we have to get the object name from the parameter `arg0` using the `getSource` method of the `MouseEvent` object in the following manner:

```
public void mouseClicked(MouseEvent arg0) {
    Object affectedObject = arg0.getSource();
    if(affectedObject.equals(button01)) {
        button01.setText("You clicked me!");
    }
    and subsequent if conditions to identify and code for
    each object in your application that you want to write a
    reaction for to this particular event.
}
```

You don't actually have to create an object reference `affectedObject`, you can simply say;

```
if(arg0.getSource().equals(button01)){
    do desired actions...
}
if(arg0.getSource().equals(button02)){
    do desired actions for when button02 is pressed...
}
```

**Note:** don't forget to declare the object references for the objects directly in the class and not in the method which calls for the instantiation of the GUI because these object references have to have the appropriate visibility.

The reader is urged to place these code segments into the previously developed solutions in order to observe their functioning.

## Practice Your Java Level 1

Create a program which functions according to the following specifications:

1. The program should have a `JTextBox` object whose initial content is the string “Just starting out”.
2. When the mouse is first moved onto the `JTextBox`, its text should change to “Hey mouse, I saw you enter...”.
3. When the mouse is clicked within it, it should change its text to “Don’t click please!!”
4. When a mouse button is held down within it, it should change its text to “Why are you holding the mouse down?”
5. When the mouse button is released, it should change its text to “You stopped holding it down. That’s good, but please leave!”
6. When the mouse leaves, it should change its text to “Phew...the mouse left!”

Present two solutions to this event, using each of the two presented means of structuring event listeners.

*Solution #1: Using an anonymous class*

```
public class graphicsPractice {  
    JTextField tf01;  
  
    public void createGUI() {  
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();  
  
        JFrame frame01 = new JFrame("My Application Title");  
        frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        JPanel pane01 = new JPanel(new GridBagLayout());  
        frame01.setContentPane(pane01);  
        frame01.setSize(screenSize.width/2, screenSize.height/2);  
  
        tf01 = new JTextField("Just starting out",50);  
        Font font01 = new Font("Lucida Console", Font.BOLD | Font.ITALIC, 20);  
        tf01.setFont(font01);  
  
        tf01.addMouseListener(new MouseListener(){  
            public void mouseEntered(MouseEvent arg0) {  
                tf01.setText("Hey mouse, I saw you enter...");  
            }  
            public void mouseClicked(MouseEvent arg0) {  
                tf01.setText("Don't click please!!!!");  
            }  
            public void mousePressed(MouseEvent arg0) {  
                tf01.setText("Why are you holding the mouse down?");  
            }  
            public void mouseReleased(MouseEvent arg0) {  
                tf01.setText("You stopped holding it down. That's good, but please leave!");  
            }  
            public void mouseExited(MouseEvent arg0) {  
                tf01.setText("Phew...the mouse left!");  
            }  
        });  
  
        frame01.add(tf01);  
        frame01.setVisible(true);  
    }  
  
    public static void main(String[] args) throws Exception{  
        SwingUtilities.invokeAndWait(new Runnable(){  
            public void run(){  
                new graphicsPractice().createGUI();  
            }  
        });  
    }  
}
```

*Solution #2: Using an anonymous class*

```
public class graphicsPractice implements MouseListener {
```

```

JTextField tf01;

public void createGUI() {
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

    JFrame frame01 = new JFrame("My Application Title");
    frame01.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel pane01 = new JPanel(new GridBagLayout());
    frame01.setContentPane(pane01);
    frame01.setSize(screenSize.width/2, screenSize.height/2);

    tf01 = new JTextField("Just starting out",50);
    Font font01 = new Font("Lucida Console", Font.BOLD | Font.ITALIC, 20);
    tf01.setFont(font01);
    tf01.addMouseListener(this); //register the object to listen to mouse events
                                //we could have registered the frame, but this is ok
    frame01.add(tf01);
    frame01.setVisible(true);
}

public void mouseClicked(MouseEvent arg0) {
    tf01.setText("Don't click please!!!!");
}

public void mouseEntered(MouseEvent arg0) {
    tf01.setText("Hey mouse, I saw you enter... ");
}

public void mouseExited(MouseEvent arg0) {
    tf01.setText("Phew...the mouse left!");
}

public void mousePressed(MouseEvent arg0) {
    tf01.setText("Why are you holding the mouse down?");
}

public void mouseReleased(MouseEvent arg0) {
    tf01.setText("You stopped holding it down. That's good, but please leave!");
}

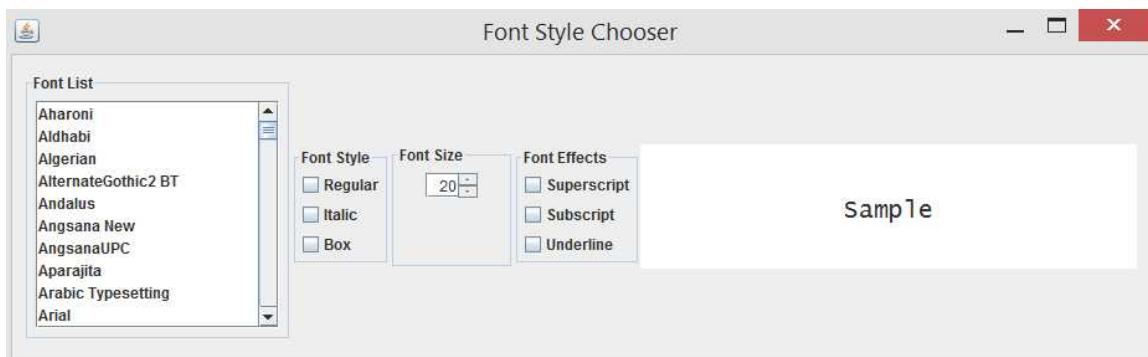
public static void main(String[] args) throws Exception{
    SwingUtilities.invokeAndWait(new Runnable(){
        public void run(){
            new graphicsPractice().createGUI();
        }
    });
}
}

```

E1 Study each of the following component classes: JComboBox, JToolTip, JSpinner.

#### Font chooser application

Write a “font chooser application” that looks like what is presented below.



The type of each of the major graphical elements therein is `JList`, `JComboBox`, `JSpinner` and `JTextPanel`.

The choice of any of the items should cause a modification of the word “Sample” that is in the `JTextPanel` to match what is chosen. Pay attention to mutually exclusive items (for example, superscript and subscript are

## *Practice Your Java Level 1*

	mutually exclusive). The <code>JSpinner</code> object should contain the range of font sizes that you want to support.
E2	The interested reader should study the concept of Dialogs.
E3	Study how to put other components, such as check boxes, into a <code>JTable</code> object.

# Chapter 33. Applets – a brief look

This chapter presents a very brief set of exercises on the topic of Java Applets. As we have already covered GUI's and given the fact that in summary an applet is a modified Java GUI application that runs in a browser, we present the following short set of exercises.

1.	(explain in a single line) What is a Java applet? An applet is a Java program that is written to run on a web page.
2.	Why was there originally a need for applets? One original reason for the existence of applets was the desire to provide a more graphics rich browsing experience for users; such graphics rich applications have also been termed “Rich Internet Applications”. Understand that applets predate web technologies like Cascading Style Sheets (CSS) and HTML 5 which now provide more advanced graphical options for webpage design.
3.	Explain the mechanism by which browsers support the running (not the loading) of Java applets. There are two main ways in which web browsers support the running of Java applets. These are: <ol style="list-style-type: none"><li>1. The embedding of a JVM within the browser itself</li><li>2. The installation of a Java browser plug-in (which works through the browser extension interface that many browsers provide). This plug-in then runs the applet on the webpage.</li></ol>
4.	What is a WebStart application? A WebStart application is a Java application that is loaded by the Java Network Launching Protocol (JNLP). This browser embedded protocol facilitates the loading of Java applets for running in a browser and also facilitates the loading of Java desktop applications over the web and running them as regular applications on a computer.
5.	Explain the mechanisms by which an applet is loaded into a browser using HTML. Basically, this launch is done by using the appropriate tags inside the HTML file. There are two different sets of tags that are used; the basic <code>applet</code> tag and the <code>jnlp</code> tag. <i>Using the Applet Tag</i> The name of the applet in question and its startup parameters (for example, startup size) are embedded in the HTML tag <code>&lt;applet&gt;</code> in an HTML page. The browser recognizes this tag and runs the indicated applet. For example, if we have a <i>local</i> applet (our example is one which we compiled in a class named <code>Hello</code> , we would embed it in an HTML page as follows: <pre>&lt;applet code="Hello.class" width="300" height="200"&gt;&lt;/applet&gt;</pre> A <i>remote</i> applet would need to specify the <code>codebase</code> parameter, for example, an applet that is to be loaded from xyzco.com's web server might look like this: <pre>&lt;applet codebase="http://www.xyzco.com/appletdir/" code="Hello.class" width="300" height="200"&gt;&lt;/applet&gt;</pre> <i>Using JNLP</i> JNLP uses a different set of html tags, primarily the tag <code>&lt;jnlp&gt;</code> . Also, JNLP specifies a particular way to package your applet. This book does not go into the details of JNLP, but wants that you be aware of this functionality. <b>Notes</b> <ol style="list-style-type: none"><li>1. For immediate testing purposes, you can use the <code>appletviewer</code> application that is packaged with Java. Also, some Java IDE's automatically recognize applet code when you try to run such and thus start your applet for you in the <code>appletviewer</code> application.</li><li>2. At present, <u>all applets</u>, whether local or not, have to be signed by a certificate authority before they are allowed to run. <b>However</b>, if you configure an exception into the Java Control Panel application (the executable <code>javacpl</code>) on your computer with the location of your local classfiles and also set browser permissions appropriately, you should be able to run your local applet in a browser that supports java applets. You would have to put the URI of the location of your class files into the exceptions list in the Java Control Panel application. (see Chapter 29, exercise 51 for an example of how to determine the URI of a directory or file).</li></ol>

## Practice Your Java Level 1

	3. Attempts have been made to deprecate the <code>&lt;applet&gt;</code> tag. In its place, the <code>&lt;object&gt;</code> tag or the <code>&lt;embed&gt;</code> tag can be used.
6.	<p>Explain the concept of a “security sandbox” for applet.</p> <p>The security sandbox refers to constraints put on a Java applet. For example, a Java applet that is running in the security sandbox cannot access the filesystem of the computer that it is on, nor can it access any network point except that from which it was downloaded (by name, not by ip address). It cannot even access the system clipboard. Also, sandboxed applets can only read a limited set of the available system properties.</p>
7.	<p>What is a “privileged applet”?</p> <p>A privileged applet is one that runs outside the sandbox and thus does not have the restrictions of a sandboxed applet. Any signed applet can request permission to run as a privileged applet.</p> <p>The user of course has to be careful in using such, as such applets have access to the components of the users’ computer.</p>
8.	<p>Explain the differences between an applet and a regular Java GUI application.</p> <p><i>Development differences</i></p> <p>A Java applet is a <code>public</code> class which extends the class <code>JApplet</code> (this for a Swing applet, an AWT applet would extend <code>Applet</code>), as opposed to a desktop GUI application which extends <code>JFrame</code>.</p> <p>A Java applet does not have the method <code>main</code>; rather it has a method named <code>init</code> which performs the same functions as <code>main</code>.</p> <p>The methods <code>setSize</code> and <code>setVisible</code> of class <code>JFrame</code> are not relevant for an applet, since it sets the size according to what is specified in the HTML tag and is automatically made visible by the browser.</p> <p>See the sample applet code below:</p> <pre>import java.awt.EventQueue; import javax.swing.JApplet; import javax.swing.JLabel;  public class AppletHello extends JApplet{      public void init(){         EventQueue.invokeLater(new Runnable(){             public void run(){                 JLabel label01 = new JLabel("Hello. How are you?");                 add(label01);             }         });     } }</pre> <p>On security: we have discussed the security constraints on such applets in the preceding exercises. With these in mind, we understand that any sandboxed applet shouldn’t attempt any file operations (It might have such written and compiled with the code, however an attempt to access forbidden areas will result in an exception).</p>

# Appendix A. Classes, Objects & Methods: An Introduction

This Appendix presents the concepts of classes and objects and then presents the implementation of these concepts in Java. The reader who is new to the concept of classes and objects should rest assured that the basic concepts of objects and classes are actually simple.

Following this, an introduction to the concept of methods in Java is made.

## Classes & Objects

We start with the question, *What is a class?* We explain by analogy, our first analogy using human beings.

The concept of human beings is a *class*

You the reader are an *instance* of the class of human beings

Your father is an instance of the class of human beings

Your mother is an instance of the class of human beings

We use dogs as another analogy. We'll assume that you have a dog and your neighbor has two dogs.

The *class* here is Dog

Your own dog is an instance of the class Dog

Your neighbors' two dogs are each individual instances of the class Dog

In Java code, the class that represents human beings can be written as follows:

```
class HumanBeing{  
}
```

We've just written the basic code for a class `HumanBeing` to represent human beings.

To create the instances of the class (you, father, mother), the code for each instance is simply as follows:

```
HumanBeing me = new HumanBeing();  
HumanBeing father = new HumanBeing();  
HumanBeing mother = new HumanBeing();
```

Think of a class as simply being a template. Each instance of the class has all the attributes that we listed in the template (we haven't put any in so far for the class `HumanBeing`).

For the `Dog` analogy we can have the following class definition:

```
class Dog{  
  
}  
  
Dog myDog = new Dog();  
Dog neighborDog1 = new Dog();  
Dog neighborDog2 = new Dog();
```

That really is the basic concept of classes and objects.

## Practice Your Java Level 1

### More details

Now, what characteristics/attributes does a human being have? They have for example, a first name, a last name, an age and a weight. We might want to have fields for these inside each instance. So we will enhance our Java class `HumanBeing` with these fields as follows:

```
class HumanBeing{  
    String firstName;  
    String lastName;  
    int age;  
    float weight;  
}
```

Looking at the enhanced definition of the class `HumanBeing`, observe that each field has a type specified for it. For example, our first names and last names are “string of characters” type data; string of character data can be represented in Java by the Java type `String`. Our ages are whole numbers (`integers`) and Java has many integer types that can handle this; we choose the type `int`. Our weight usually has fractions in it, so we need a number type that can handle fractions; `float` is a type that can handle such.

Now, how do we set the data fields for each person’s individual object? It’s simple. Let’s set the data for ourselves (Say our name is John Smith, we are 40 years old and weigh 178 pounds).

Recall we already created an object/instance that we called `me`, that represents ourself. All we now have to do is say the following:

```
me.firstName="John";  
me.firstName="Smith";  
me.age = 40;  
me.weight = 178.0;
```

Now, say Dad’s name is Singdon Smith, aged 67 and he weighs 163 pounds. You go ahead and fill in the object that represents Dad and then look at the answer below.

```
dad.firstName="Singdon";  
dad.firstName="Smith";  
dad.age = 67;  
dad.weight = 163.0;
```

You get the general picture. Of course there are some more details that can go into the picture but this is the essence of it.

### The concept of methods

Say we have a class named `Calculator`. I have one, my friend has his own. For now our calculator can perform only the operation `addition` for any two numbers given to it. How do we express the desired functionality within our `Calculator` class to do each of these operations?

The solution is methods. A method is a “named block of code” within a class. We can pass values to a method and the method will do whatever action it has been coded to do and can pass the solution back. Let us create a method named `addition` to do this:

```
class Calculator{  
    float addition(float a, float b){  
        float result = a + b;  
        return(result);  
    }  
}
```

You see that we created the method within the parentheses of the class `Calculator`. We named the method `addition` and as can be seen, it takes two values of type `float`, which we chose to give the names `a` and `b`. The variables names just stand in for whatever values we pass to it, there is nothing special or specific about the names.

When we pass values to a method we have to specify the type that the value is. We want to be able to add non-integer values, so we stated that we are passing a `float` value and another `float` value.

The computer needs to know what kind of value each method intends to send back to whoever invoked it; we are sending a value of type `float` back; that is the why the word `float` appears before the method name.

Inside the method you see that we created a variable named `result` which has a type of `float`, into which we put the result of our computation.

Then we have to tell the method to send a `float` back, because it did say it was going to send a `float` back. We do this using the built-in method `return` and we tell the built-in method `return` which `float` variables' value we want it to send back, which in this case is the value of the `float result`.

So how do we call/invoke our calculator addition method? Simple, we call it from another method! For example, we called the method `return` from within the method `addition`.

Let's write a full program where we will perform the addition of the numbers 5 and 7 on our instance of a `Calculator`.

```
class Calculator{
    float addition(float a, float b){
        float result = a + b;
        return(result);
    }

    static void main(String[] args){
        Calculator JohnsCalculator = new Calculator();
        float myResult;
        myResult = JohnsCalculator.addition(5.2, 7.2); //I am using my calculator object.
        //Now let's print it to the console
        System.out.println(myResult);
    }
}
```

You can actually take this program, put it in a file named `Calculator.java`, compile it and run the resulting `Calculator.class` file.

### *Some explanation*

Every class in Java that we want to run directly must have a method named `main`. This method is the one that Java looks for to run. The header line of the method `main` is generally presented in the fashion that we see it here (for more details see Chapter 1).

Think of the method `main` as a “governor” method for what we do with the other methods in the class.

See that we created our own `Calculator` instance (`JohnsCalculator`), in the method `main`.

Now, that we have a `Calculator` object that we named `JohnsCalculator`, we want to perform our addition on this calculator. So we simply call the `addition` method on our calculator (because if you want an addition on *your* calculator you press the buttons on *your* own very calculator, not on someone else's) as `JohnsCalculator.addition(5.2, 7.2)`. But remember that the method `addition` sends

## Practice Your Java Level 1

back a value which we want to capture; it sends a `float` type value back and we want to know what it is, so we create a `float` named `myResult` with which to receive what our `addition` method gives us.

Just like writing a formula  $y = \sin(x)$ , we wrote `myResult = JohnsCalculator(value, value)` and the result was returned.

Note something: what we call the parameters in the definition for the method `addition` (we called them `a` and `b`) does not matter, we could have called them anything. All the method `addition` is saying is “I take two numbers, I’ll call the first something and the second something else so I can keep track of them”.

We used the built-in method `System.out.println` to print the result out to the screen.

There in a nutshell is the concept of classes, class instances (called objects) and methods.

You’ve actually touched on more concepts than these in this short chapter. The topics you’ve learned are:

Classes, class instance (object) creation, class instance field definition, instance field assignment, instance method creation, instance method invocation, passing parameters to methods, returning parameters from methods, receiving parameters from methods, using the method `main`, printing a variable out to the console!

**Note:** For practice, the reader is encouraged to add more methods (for example methods for division, subtraction and multiplication) to the class `Calculator` and test them.

# Appendix B. Project

Here is a project which will exercise your skills in a number of the areas that have been presented in this volume. As you go through the exercise, you should be able to identify which particular Java features apply to the different parts of the exercise.

In this project we are going to simulate the operation of a bank. You will build the simulation to run for simulated periods of time of your choosing, the simulated period reflecting 1 day to 30 years of operations.

The bank will have a set of customers, who come in to the bank from time to time. The potential activities of the customers include the following:

## Customer Activities

1. Salary deposit: this is deposited every week in cash by the customers
2. Miscellaneous deposits
3. Requesting loans
4. Loan repayment (this is an automatic withdrawal by the bank and is always scheduled for the day after their salary deposit)
5. Cash withdrawals (if they have taken any loan, then the amount left in the account must suffice for the next loan repayment)
6. Statement of transactions for the day (this should include the ID# of the employee who served the customer)
7. Obtain a monthly bank statement for the last month which states the following:
  - a. total deposits they have
  - b. cash withdrawals in the given month
  - c. Each loan taken out and the interest rate
  - d. Total of loans taken out
  - e. Loan amounts repaid so far
  - f. Expected loan repayment completion date for each loan
  - g. Next loan repayment date

This statement should be available on demand. Also, whether it is requested or not, the bank should produce it on the first of the month and store it in a file directory for the customer.

The file directory structure should be `customer_id_number/year/monthly_statements/`

For each day that the customer comes into the bank, they should have a record of their daily activity stored in a directory of the following structure: `customer_id_number/year/daily_reports/`

## End of day duties

Your bank itself has the following duties at the end of the day

1. Produce a report stating the total deposits for the day
2. Produce a report stating the total loans issued that day

These should be stored in an appropriately named directory.

## Bank Loans

1. Maximum loan amount across all loans = 300% of salary
2. Interest rate: randomly selected value between 3.00% and 8.00%
3. Payment term: variable, from one month to 30 years
4. Aggregate loan repayment for a customer cannot exceed 50% of their salary

### More details

Your simulation should create the following:

**A list of bank customers.** One way to do this is to have a list of first names and also a list of last names and then randomly pair data from each list up in a way so as to generate as many customers as you want in your simulation. At the same time, assign customer ID numbers; sequential numbers might be optimal. Randomly assign each customer the following: a day of the month on which to deposit their pay and the amount of the pay. If that day falls on a day on which the bank is not open, then customer should make the deposit on the next business day.

Your customer class will have fields to represent at least the following: *id, first name, last name, salary, deposit date.*

**A list of bank tellers.** Specify to your program how many of these you want and it can create them from lists in the same manner in which the customer list is created. Assign them employee id numbers too. This employee id# should appear on the customer statement that is printed out at the end of the day.

**Bank hours.** Define your banking hours and days within your simulator. That will help determine the “end of the day”. On days on which the bank is not open, still go ahead and generate bank reports for the day to reflect the lack of activity on those days.

**Customer arrivals:** Randomly select which customers show up at the bank on any given day within the hours the bank is open. They should enter a queue and they will be serviced by the next available teller. Customers who are depositing their pay on a given day will not be selected randomly, but must be in the list of customers for the day.

A customer can have multiple activities that they want to perform on a given day. Select this set of activities randomly too. You might want to apply a weighting to certain activities, for example loan requests cannot be too often. Also, the customer is free to reject the loan if they find the offered interest rate to be unacceptable.

**Banking activity time:** Assign a time for each banking activity. This will help determine the time that each customer spends at the teller. The next customer on the queue should automatically be assigned to the next available teller.

**Day transaction statement:** On processing any customer, copy their statement of activity for the day to the outbox directory.

**Bank statement requests:** If the customer requests their statement of transactions for the month so far, copy this to the outbox directory.

### End of day

At the end of the banking day, run the bank reports for the day in question. Store them in an appropriately named directory.

At the end of the month, create a monthly statement for each of your customers. Also create a monthly statement for the bank itself.

Of course, as the developer, you will scale your unit of time appropriately to represent days, months and years!

# Index

## A

Accessor/Mutator methods, 200  
 Applet  
   (*definition*), 305  
   Browser plug-in, 305  
   *loading mechanisms*, 305  
   *privileged*, 306  
   *security sandbox*, 306  
   WebStart application (*definition*), 305  
 Arrays  
   (*definition*), 61  
   *initial index*, 61  
   Multi-dimensional, 67  
 auto-boxing, 81  
 auto-unboxing, 82

## B

Beans  
   *OperatingSystemMXBean*, 250  
   *RuntimeMXBean*, 250  
 BigDecimal, 99  
   MathContext, 101  
   precision, 100  
   Rounding, 103  
   rounding modes, 100  
   Scale, 103  
 BigInteger, 95  
 blank final (*definition*), 22  
**boolean**, 41  
 boolean testing  
   != (not equals), 42  
   & (and), 41  
   (not greater than), 42  
   < (less than), 42  
   <= (less than or equal to), 41  
   == (equal to), 43  
   > (greater than), 41  
   >= (greater than or equal to), 42  
   compound expressions, 43  
   short-circuiting operators, 43  
 boxing/autoboxing (see  
   unboxing/autounboxing), 81  
 Buffered Files, 278

## C

Casting, 18  
   boxing, 18  
   *implicit*, 19

objects, 18  
   unboxing, 18  
**char**, 157  
   char/hexadecimal conversion, 158  
   char/String relationship, 161  
 Class  
   (*definition*), 175  
   class access modifier  
     **package-private** (*implied modifier*), 175  
     **public**, 175  
   class type  
     **concrete class**, 189  
   class type modifiers, 188  
     **abstract**, 188  
     **final**, 188  
   class types  
     **final**, 197  
   constructor  
     (*definition*), 177  
     default constructor, 178  
     *explicit invocation*, 179  
     overloaded, 187  
     overloading, 177  
     parameterless constructor, 177  
     private constructors, 190  
   hiding methods, 195  
   *immutable classes*, 204  
   inheritance, 180  
     multiple (*concept*), 180  
     overriding superclass methods, 181  
   member access modifier  
     *none specified*, 176  
     **private**, 176  
     **protected**, 176  
     **public**, 176  
   member modifiers  
     **abstract**, 189  
     **final**, 189  
     **static**, 189  
   members, 175  
   nested class, 175  
   overriding methods, 195  
   polymorphism and method hiding, 195  
   polymorphism and method overriding, 195  
   subclass, 180  
   superclass, 175  
   top-level class, 175  
   type  
     nested class, 209  
     nested classes  
   types  
     abstract classes, 193  
     static classes, 189  
 Classes  
   **Arrays**, 63  
   **Boolean** (wrapper class), 89  
 Character Handling  
   **Character**, 157  
 Collections  
   **ArrayList**, 219  
   **Collections**, 224  
   **HashMap**, 228  
   **HashSet**, 228  
   **Stack**, 226  
   **TreeSet**, 227  
 Date/Time Handling  
   **DateTimeFormatter**, 132  
   **Duration**, 134  
   **Instant**, 134  
   **LocalDate**, 120  
   **LocalDateTime**, 117  
   **LocalTime**, 123  
   **MonthDay**, 127  
   **Year**, 127  
   **YearMonth**, 127  
   **ZonedDateTime**, 130  
   **ZoneId**, 130  
   **ZoneOffset**, 130  
 GUI classes  
   **GraphicsEnvironment**, 295  
   **JButton**, 292  
   **JCheckBox**, 297  
   **JComboBox**, 303  
   **JLabel**, 293  
   **JList**, 295  
   **JPanel**, 291  
   **JPasswordField**, 295  
   **JScrollPane**, 295  
   **JSpinner**, 303  
   **JTable**, 296  
   **JTextArea**, 298  
   **JTextField**, 294, 298  
   **JToolTip**, 303  
   **Toolkit**, 292  
 I/O  
   **BufferedReader**, 279  
   **BufferedWriter**, 280  
   **Files**, 254  
   **FileStore**, 259

# Practice Your Java Level 1

- F**  
`FileSystems`, 259  
`FileTime`, 262  
`Path`, 254  
`Paths`, 254  
`SimpleFileVistor`, 266  
Input/Output  
    `Console`, 147  
    `Scanner`, 150  
    System (see method `in`), 154  
members  
    instance members, 4  
    static members, 4  
Miscellaneous  
    `Enum`, 91  
    `FileSystemView`, 249  
    `Random`, 163  
    `Runtime`, 248  
    `System`, 247  
Numeric  
    `BigDecimal`, 99  
    `BigInteger`, 95  
    `Math`, 22  
    `MathContext`, 101  
    `Random`, 163  
    `StrictMath`, 110  
Numeric Wrapper Classes  
    `Double`, 87  
    `Float`, 87  
    `Integer`, 81, 82  
    `Long`, 87  
    `Short`, 87  
    `StrictMath`, 110  
String Handling  
    `String`, 47  
    `StringBuffer`, 167  
ultimate ancestor  
    `Object`, 185  
classfile, 2  
Collections  
    (*definition*), 219  
Classes  
    `Collections`, 224  
Generics, 219  
Types  
    `ArrayList`, 219  
    `HashMap`, 228  
    `HashSet`, 228  
    `Stack`, 226  
    `TreeSet`, 227  
Command Line Input, 155  
Comments, 3  
    `//`, 3  
    multi-line(`/*...*/`), 4  
Conditional Processing  
    `case`, 57
- I**  
I/O  
    Compression/`ZipFiles`, 273  
        *creating*, 274  
        *extraction*, 274  
    Creating directories, 255  
    Creating subdirectories, 255  
    Deleting directories, 256  
    Determining Desktop directory, 254  
    Determining file access time, 262  
    Determining file executability, 262  
    Determining file hidden state, 261  
    Determining file ownership, 263  
    Determining file readability, 262  
    Determining file size, 260  
    Determining file volume, 264  
    Determining home directory, 254  
    Determining working directory, 253  
    Directory operations, 266  
    Directory traversal, 266  
    File Information, 260  
File Operations  
    appending to files(buffered), 281  
    buffered files, 278  
    reading files, 277  
    reading files(buffered), 279  
    writing files, 277  
    writing files(buffered), 280  
File system information, 259  
Files  
    copying, 264  
    deleting, 265  
    moving, 265  
    renaming, 265  
    Symbolic links  
        *creating (hard)*, 271  
        *creating (soft)*, 269
- D**  
Date/Time Handling, 117  
    date/time formatting, 132  
Default/defender methods (Interface), 236  
Defender/default methods (Interface), 236
- E**  
Enumerations, 91  
    (*advanced*), 285  
    Date/Time Handling  
        `ChronoUnit`, 126  
        `DayOfWeek`, 127  
        `Month`, 127  
        *declaring, assigning*, 92  
    I/O  
        `StandardOpenOption`, 281  
Event Listeners, 300  
Exceptions, 137  
    catching, 137  
    checked, 141  
    `finally` block, 142  
    rethrowing, 143  
    `throw`, 139  
    throwing deliberately, 139  
    *try with resources*, 145  
    `try/catch` block, 137  
    user-defined, 140
- G**  
Generics  
    (*definition*), 219  
    getter/setter methods, 200  
Globbing (see *Filtering directory stream data*), 258  
Graphical User Interface  
    nested panes, 299  
Graphical User Interfaces  
    Abstract Windowing Toolkit (AWT), 289  
Components(definition), 289  
Event Listeners, 289

*determining*, 269  
*determining actual target*, 269  
*hard vs soft*, 270  
*Filtering directory stream data*, 258  
*pattern matching/globbing*, 258  
*Listing filestores*, 259  
*Listing root directories*, 259  
Paths  
*root determination*, 275  
*sub-element determination*, 275  
*Setting current working directory*, 254  
Symbolic links, 268  
*Temporary Files/Directories*, 272  
*Testing directory existence*, 254  
URI, 272  
*Creating*, 272  
inheritance, 180  
Initializer  
*instance initializer*, 198  
*static class initializer*, 197  
Input/Output, 147  
Interface  
*(benefits)*, 231  
*(definition)*, 231  
*(usage)*, 231  
default/defender methods, 236  
Interfaces  
Collections Interfaces  
*Deque*, 241  
*IListIterator*, 243  
*Iterator*, 242  
*List*, 241  
*Map*, 241  
*Queue*, 241  
*Set*, 241  
*DirectoryStream*, 256  
Iterations  
*do/while loop*, 74  
enhanced *for* loop, 73  
*for* loop, 70  
keywords  
*break*, 76  
*break*, 77  
*continue*, 76  
*labeled break*, 76  
labels, 76  
*while* loop, 69  
Iterator  
*(definition)*, 242  
*Iterator*, 242  
*ListIterator*, 243

**J**

Java Virtual Machine (JVM), 2  
JVM (*see Java Virtual Machine*), 2

**K**

Keywords  
*break*, 57, 76  
*case*, 57  
*catch*, 137  
*continue*, 76  
*default*, 57  
*do*, 74  
*extends*, 180  
*extends (interface)*, 238  
*final*, 182  
*final (applied to constants)*, 21  
*finally*, 142  
*for*, 70  
*implements*, 231  
*import*, 1  
*package*, 1  
*return*, 3  
*strictfp*, 109  
*super*, 183  
*switch*, 57  
*this*, 178  
*invoking overloaded constructor*, 178, 188  
*parameter disambiguation*, 178  
*self reference for passing \i*, 178  
*throw*, 139  
*try*, 137  
*void*, 3  
*while*, 69

**L**

labels, 76

**M**

**main**, 2  
*potential declarations*, 3  
Methods  
**main**, 2  
method declaration, 5  
method signature, 4  
overloading, 5  
static method, 5  
variable number of arguments (*varargs*), 6  
Mutator/Accessor methods, 200

**N**

NIO/NIO.2, 253  
Number types  
*BigDecimal*, 17  
*BigInteger*, 17

binary numbers, 32  
byte, 12  
Complex Numbers, 28  
constants  
*Byte.MIN\_VALUE*,  
*Byte.MAX\_VALUE*,  
*Byte.SIZE*, 12  
*Float.MIN\_VALUE*,  
*Float.MAX\_VALUE*,  
*Float.SIZE*, 14  
*Float.NEGATIVE\_INFINITY*, 28  
*Float.POSITIVE\_INFINITY*, 28  
*Infinity(∞)*, 27  
*-Infinity(-∞)*, 27  
*Integer.MIN\_VALUE*,  
*Integer.MAX\_VALUE*,  
*Integer.SIZE*, 12  
*Long.MIN\_VALUE*,  
*Long.MAX\_VALUE*,  
*Long.SIZE*, 13  
*NaN (Not A Number)*, 27  
*Short.MIN\_VALUE*,  
*Short.MAX\_VALUE*,  
*Short.SIZE*, 13  
double, 15  
fidelity of floating point types, 16  
float, 14  
formatting numeric output, 33  
format specifiers, 33  
hexadecimal numbers, 32  
int, 11  
long, 13  
precision, 15  
primitives  
*byte, short, int, long, float, double*, 11  
short, 12  
suffixes, 17  
unsigned numbers, 111  
wrapper classes, 11

**O**

Object  
*(definition)*, 175  
Objects  
*passing*, 206  
Operator  
*new*, 81  
Operators  
*-- (decrement)*, 28  
*- (subtraction)*, 22  
*% (modulus)*, 22  
*/ (division)*, 27

+ (addition), 22  
++ (increment), 28  
<> Diamond Operator, 219  
>>> (right shifting unsigned integers), 113  
bitwise operators, 38  
  << (left-shift), 39  
  << (right-shift), 39  
boolean operators  
  != (not equals), 42  
  & (and), 41  
  < (less than), 42  
  <= (less than or equal to), 41  
  > (greater than), 41  
  >= (greater than or equal to), 42  
compound operators  
  && (and), 43  
  ^ (xor), 45  
  || (or), 44  
logical operators, 38  
  & (AND), 38  
  ^ (XOR), 38  
  | (OR), 38  
  ~ (one's complement), 38  
shorthand, 29  
ternary operator (?), 57  
Output/Input, 147  
Overflow (integer), 29  
Overflow/Underflow avoidance  
  **Exact** methods, 30  
  **Exact** methods (BigInteger), 98

## P

package  
  (*definition*), 1, 175  
Package  
  accessing from another, 199  
Polymorphism, 195  
precision of number types, 15  
Primitive types  
  boolean, 41  
  byte, short, int, long, float,  
    double, 11  
primitive variable  
  (*definition*), 201  
primitives  
  char, 157  
Properties, 247  
  creating, 252  
  reading, 247  
  setting/ modifying, 251

## R

random number generation, 33, 163

## S

setter/getter methods, 200  
Singleton pattern, 192  
StrictMath, 109  
String/char relationship, 161  
Strings, 47  
  *equality*, 49

formatting, 54  
immutability, 48  
substrings, 49

Symbolic links (*see I/O, symbolic links*), 268  
System Properties (*see Properties*), 247  
**System.out.println**, 7

## T

Temporary Files/Directories, 272  
type  
  *definition*, 201  
  primitive type  
    (*definition*), 201  
reference type  
  *definition*, 201

## U

unboxing, 82  
unboxing/autounboxing(*see boxing/autoboxing*), 82  
Underflow (integer), 30  
Unicode, 157  
Unicode/UTF-16, 157  
Unsigned number handling, 111  
URI  
  *see I/O, URI*, 272

## V

*varargs* (variable number of arguments), 6