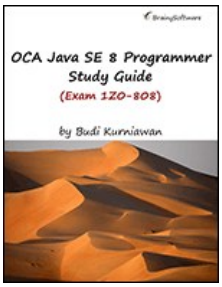

OCA Java SE 8 Programmer Study Guide (Exam 1Z0-808)

by Budi Kurniawan

Appendix B: Mock Exam 2

The following are answers to the questions in the second mock exam. Each question may have more than one correct answer and you have to choose all the correct answers.

1 Question (1.3. Create executable Java applications with a main method; run a Java program from the command line; including console output) ?

Consider the following code.

```
package demo;
public class MainDemo {
    public static void main(String[] args) {
        System.out.println(args[0]);
    }
}
```

It is invoked with this command line command:

```
java demo.MainDemo 20+30
```

What is printed on the standard out?

A. 50

B. 20

C. 30

D. 20+30

2 Question (1.3. Create executable Java applications with a main method; run a Java program from the command line; including console output) ?

Consider the following code snippet:

```
1.   package test;
2.
3.   public class Device {
4.       void main() {
5.       }
6.   }
```

What will happen if you try to compile this class? (Choose all that apply)

A. You will get a compile error at line 4 because **main** must be static.

B. You will get a compile error at line 4 because **main** must be public and static.

C. You will get a compile error at lines 3 and 4.

D. The class compiles without issues.

3 Question (1.3. Create executable Java applications with a main method; run a Java program from the command line; including console output) ?

Given

```
package smalldevice;
public class Counter {
    public static void main(String[] args) {
        System.out.print(args[1]);
        System.out.print(args[2]);
    }
}
```

The program is invoked using this command.

```
java smalldevice.Counter zero "one" two
```

What is printed on the console?

A. zeroone

B. zero"one"

C. onetwo

D. "one"two

4 Question (1.4. Import other Java packages to make them accessible in your code) ?

Which sets of **import** statements would allow you to use **java.util.List, java.util.ArrayList** and **java.lang.System** in your class?

A. import java.util.*;

B. import java.util.List; import java.util.ArrayList;

C. import java.util.List; import java.util.ArrayList; import java.lang.System;

D. import java.lang.*;

E. import java.lang.System;

5 Question (1.4. Import other Java packages to make them accessible in your code) ?

Which sets of **import** statements would allow you to use **java.util.List, java.util.ArrayList** and **java.lang.annotation.Documented** in your class?

A. import java.util.*;

B. import java.util.List; import java.util.ArrayList;

C. import java.util.*; import java.lang.annotation.Documented;

D. import java.util.*; import java.lang.annotation.*;

E. import java.lang.*;

6 Question (4.1. Declare, instantiate, initialize and use a one-dimensional array) ?

Assuming **stringArray** is a one-dimensional array, which of the following expressions returns the number of element in **stringArray?**

A. stringArray.size

B. stringArray.size()

C. stringArray.length

D. stringArray.length()

E. stringArray.capacity

F. stringArray.capacity()

7 Question (2.1. Declare and initialize variables (including casting of primitive data types)) ?

What is the output of this program?

```
 1. public class Utility {
 2.     public static void main(String[] args) {
 3.         int i = 9;
 4.         char c = '3';
 5.         i = c;
 6.         c = i;
 7.         System.out.print(i);
 8.         System.out.print(c);
 9.     }
10. }
```

A. The code prints 93

B. Compile error at line 5

C. Compile error at line 6

D. Compile error at lines 5 and 6

8 Question (7.5. Use abstract classes and interfaces) ?

What is the result of compiling this code?

```
interface Viewable {
    public List<String> getAllScreenNames();
    public static int getWidth() {
        return 1024;
    }
    public default int getScreenCount() {
        return 1;
    }
    String getDefaultScreenName();
}
```

A. Compile error because an interface may not contain static methods

B. Compile error because an interface may not contain method implementations

C. Compile error because the **getDefaultScreenName** method has no access modifier

D. It compiles without issues

9 Question (9.5. Write a simple Lambda expression that consumes a Lambda Predicate expression) ?

What is the possible outcome of this code?

```
package test;
import java.util.function.Predicate;
public class Tester {
    public static void main(String[] args) {
        Predicate<String> isNumber = x -> {
            for (int i = 0; i < x.length(); i++)
                if ("0123456789".indexOf("" + x.charAt(i)) == -1)
                    return false;
            return true;
        };
        System.out.println(isNumber.test(args[0]) ? "TRUE" : "FALSE");
    }
}
```

A. The code may print TRUE

B. The code may print FALSE

C. The code may throw an exception

D. The code will not compile

10 Question (9.1. Manipulate data using the StringBuilder class and its methods) ?

What does this code fragment print on the console?

```
StringBuilder buffer = new StringBuilder("Reversible");
for (int i = 0; i < buffer.length(); i++)
    if (buffer.charAt(i) == 'e')
        buffer.replace(i - 1, i, "E");
System.out.println(buffer);
```

A. REvErsiblE

B. REVERSIBLE

C. EeEersibEe

D. EEEErsibEE

11 Question (2.5. Develop code that uses wrapper classes such as Boolean, Double, and Integer) ?

What is the outcome of the following class?

```
package example.test;
public class SmartUtility {
    public static double average(Integer... numbers) {
        long temp = 0;
        for (Integer i : numbers)
            temp += i;
        return temp / numbers.length;
    }
    public static void main(String[] args) {
        int[] ints = {1, 2, 6, 7};
```

```
        System.out.println(average(ints));
    }
}
```

A. The class prints 4.0

B. The code generates a compile error

C. The code throws a runtime exception

D. The class prints 4

12 Question (6.1. Create methods with arguments and return values; including overloaded methods) ?

Consider the following code.

```
class Car extends java.lang.Object {
    private void brake() {}
    protected void brake(float decelaration) {}
    public int run(int speed) {
        int currentSpeed = 100;
        return currentSpeed < speed? currentSpeed : speed;
    }
    public static int run(int speed, int accelaration) {
        return speed + accelaration;
    }
    public String toString() {
        return "Car";
    }
}
```

Which statement(s) is or are true regarding the code?

A. The run methods are overloaded methods

B. The run methods are not overloaded methods since you cannot overload a static method with an instance method

C. The brake methods are overloaded methods.

D. The toString method in Car is an overload of the toString method in java.lang.Object.

E. The Car class will generate a compile error because of duplicate method names

13 Question (1.2. Define the structure of a Java class) ?

Which of the following can appear within a class? (Choose all that apply)

A. Comments

B. Methods

C. Fields

D. Embedded files

E. Import statements

14 Question (1.5. Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.) ?

Which of the following are features of Java?

A. Java is just like the conventional programming paradigm, after a program is compiled you get an executable file

B. Java programs can run in multiple platforms

C. To achieve platform independence, Java programs are compiled into different forms that work in multiple platforms

D. Java programs are compiled into bytecode

15 Question (2.1. Declare and initialize variables (including casting of primitive data types)) ?

Given

```
int i = 257;
byte b = (byte) i;
```

What is the value of **b** after the code is executed?

A. 255

B. 256

C. 257

D. 1

E. -1

16 Question (3.2. Test equality between Strings and other objects using == and equals()) ?

Consider this code.

```
Double d = new Double(12.34);
Double e = new Double(12.34);
Double f = 12.34;
Double g = 12.34;
```

Which of the following expressions return true? (Choose all that apply)

A. d==e

B. f==g

C. d.equals(e)

D. f.equals(g)

17 Question (2.2. Differentiate between object reference variables and primitive variables) ?

Consider this code:

```
Long a = new Long(1000);
Long b = new Long(1000);
long c = 1000;
long d = 1000;
```

Which of the following expressions evaluate to true? (Choose all that apply)

A. a==b

B. a==c

C. c==d

D. a!=d

18 Question (8.4. Create and invoke a method that throws an exception) ?

What is the output of the following program?

```
package test.math;
public class MathTester {
    public static double average(Integer... numbers) throws Exception {
        double total = 0;;
        for (Integer i : numbers) {
            if (i == null) {
                throw new Exception("Number cannot be null");
            }
            total += i;
        }
        return total/numbers.length;
    }

    public static void main(String[] args) {
        Integer[] numbers = {Integer.MIN_VALUE, Integer.MAX_VALUE, 0, 0};
        try {
            System.out.println(average(numbers));
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

A. 0.00

B. -0.00

C. -0.25

D. Number cannot be null

19 Question (7.2. Develop code that demonstrates the use of polymorphism; including overriding and object type versus reference type)

Consider this class declaration:

```
class Entertainment implements java.io.Serializable {
}
```

Which of the following expressions evaluates to true?

A. (new Entertainment() instanceof Object)

B. (new Entertainment() instanceof Serializable)

C. (new Entertainment() instanceof Entertainment)

D. (new Object() instanceof Entertainment)

E. (new Object() instanceof Serializable);

20 Question (9.2. Creating and manipulating Strings)

Which of the following expressions case-insensitively compare two strings?

A. a.equals(b)

B. a.toUpperCase().equals(b.toUpperCase())

C. a.toLowerCase().equals(b.toLowerCase())

D. a.equalsIgnoreCase(b)

21 Question (3.3. Create if and if/else and ternary constructs)

What is printed on the console if the following class is executed without an argument?

```
package test;
public class Argument {
    public static void main(String[] args) {
        String comment = args.length > 0 ? args[0].equals("file") ?
                "accept" : "ignore" : "invalid";
        System.out.println(comment);
    }
}
```

A. file

B. accept

C. ignore

D. invalid

22 Question (5.1. Create and use while loops)

Given

```
package exercise;
public class EternalLoop {
    public static void main(String[] args) {
        int[] a = { 10, 20, 30, 40, 50, 10 };
        int i = a.length - 3;
        while (i > 0) {
            System.out.print(a[a.length - i]);
            i--;
        }
    }
}
```

What is printed on the console?

A. 123456

B. 102030405010

C. 405010

D. 10203040

E. 2030405010

23 Question (2.4. Explain an object's lifecycle (creation, "dereference by assignment" and garbage collection)) ?

Given

```
1.  public class Skill {
2.      public static void main(String[] args) {
3.          Skill s1 = new Skill();
4.          Skill s2 = new Skill();
5.          if (args.length > 0) s1 = null;
6.          System.out.println(s1);
7.          if (args.length > 1) s2 = new Skill();
8.          System.out.println(s2);
9.      }
10. }
```

Which of the following statements are true?

A. Two objects will be garbage collected at line 8.

B. One object will be garbage collected at line 6.

C. The object referenced by **s1** may be eligible for garbage collection at line 5.

D. The object referenced by **s2** is eligible for garbage collection at line 7.

24 Question (6.3. Create and overload constructors; including impact on default constructors) ?

Given the following code

```
package sample;
class Event {
    private String name;
    private String description;
    public Event(String name) {
        this.name = name;
    }
    public Event(String name, String description) {
        this.name = name;
        this.description = description;
    }
    public String getName() {
        return name;
    }
    public String getDescription() {
        return description;
    }
    public String toString() {
        return name + ", " + description;
    }
}

public class EventOrganizer {
    public static void main(String[] args) {
        Event e1 = new Event("New Year's Eve", "Sponsored event");
        Event e2 = new Event("101th Birthday");
        Event e3 = new Event(); // empty event
        System.out.print(e2);
        System.out.print(e3);
    }
}
```

Which of the following statements are correct?

A. The code will print "New Year's Eve, Sponsored event 101th Birthday"

B. The code will not compile

C. The code will print an empty string

D. The code will throw a runtime exception

25 Question (1.1. Define the scope of variables) ?

Consider the following code snippet that contains named regions A, B, C, D, E and F.

```
int m = 0;
long x = 1001;
// ------- A --------
if (m == 0) {
    // ------- B --------
    for (int n = 0; n < 9; n++) {
        // ------- C --------
        System.out.println(x - n);
        System.out.println(n);
        if (n == 2) m++;
        // ------- D --------
    }
    // ------- E --------
    x++;
}
// ------- F --------
```

Which of the following statements are true?

A. *n* cannot be used in region F.

B. *m* and *n* can be used in region B

C. *m* and *n* can be used in region D

D. *m* and *n* can be used in region F

26 Question (2.3. Know how to read or write to object fields) ?

Consider the following code.

```
package com.example;
class Event {
    public String name;
    private String description;
    public Event(String name) {
        this.name = name;
    }

    public Event(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = this.description;
    }
    public String toString() {
        return name + ", " + description;
    }
}

public class Organizer {
    public static void main(String[] args) {
        Event e = new Event("Summer Festival", "New York & Toronto");
        e.name = "Winter is coming early";
        e.setDescription("Summer Festival canceled");
        System.out.println(e);
    }
}
```

What is the outcome of the code?

A. The code generates a compile error

B. The code prints "Summer Festival, New York & Toronto"

C. The code prints "Winter is coming early, Summer Festival canceled"

D. The code prints "Winter is coming early, New York & Toronto"

27 Question (3.1. Use Java operators; including parentheses to override operator precedence) ?

What is the output of the following code snippet?

```
public class Tester {
    public static void main(String[] args) {
        int i = 3, j = 3;
        System.out.print(++i + i);
        System.out.print(" ");
        System.out.print(j++ + j);
        System.out.print(" ");
        System.out.print(i);
        System.out.print(" ");
        System.out.print(j);
    }
}
```

A. 8 8 4 4

B. 7 7 4 4

C. 7 8 4 4

D. 8 7 4 4

28 Question (3.4. Use a switch statement) ?

What is the output of the following code?

```
package base;

public class SwitchTester {
    public static void main(String[] args) {
        int n = 100;
        switch (++n) {
        case 100:
            System.out.print("100");
        default:
            System.out.print("default");
        }
    }
}
```

A. 100default

B. 100

C. default

D. default100

29 Question (4.2. Declare, instantiate, initialize and use multidimensional array) ?

Consider the following class:

```
public class Test100 {
    public static void main(String[] args) {
        int[][][] matrix = {{{1, 2, 3},{4, 5, 6}}, {
            {10, 20, 30},{40, 50, 60}}};
        System.out.print(matrix[0][0][0] * matrix[1][1][1]);
    }
}
```

What is printed on the console when the class is executed?

A. 8

B. 10

C. 40

D. 50

E. 80

F. 120

30 Question (5.2. Create and use for loops including the enhanced for loop) ?

Given

```
package test;
import java.util.ArrayList;
import java.util.List;
public class Test200 {
    public static void main(String[] args) {
        String[] names = { "Henri", "Borousa" };
        String[] cities = { "Melbourne", "Wilton" };
        List<String> mix = new ArrayList<>();
        for (String name : names) {
            mix.add(name);
            for (String city : cities) {
                mix.add(city);
            }
        }
        String[] namesCities = new String[]{};
        namesCities = mix.toArray(namesCities);
        System.out.println(namesCities.length);
    }
}
```

What is printed on the console when the class is run?

A. 2

B. 4

C. 6

D. 7

E. 8

31 Question (9.3. Create and manipulate calendar data using classes from java.time.LocalDateTime, java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period) ?

Consider the following code.

```
package tester;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

public class FlightTime {
    public static void main(String[] args) {
        LocalDate depDate = LocalDate.of(2015, 12, 12);
        LocalTime depTime = LocalTime.of(10, 20);
        LocalDateTime departure = LocalDateTime.of(depDate, depTime);
        LocalDateTime arrival = departure.plusHours(8).plusMinutes(30);
        System.out.println(arrival.format(
            DateTimeFormatter.ofPattern("yyyy mm dd hh:MM")));
    }
}
```

What is printed on the console?

A. 2015 12 12 06:50

B. 2015 50 12 06:12

C. 2015/12/12 06:50

D. 2015/06/12 12:50

E. 2015 12 12 18:50

32 Question (9.4. Declare and use an ArrayList of a given type) ?

Consider the following class:

```
package com.brainysoftware.server;
import java.util.ArrayList;
public class Gauge {
    public static void main(String[] args) {
        ArrayList<Long> collection = new ArrayList<>();
        collection.add(459);
        collection.add(new Long("123"));
```

```
        for (long l : collection) {
            System.out.print(l);
        }
    }
}
```

What is the result of this class?

A. Compilation error

B. Runtime error

C. It prints 459123 when run

D. It prints 459 when run

33 Question (7.1. Describe inheritance and its benefits) ?

Which of the following statements are true? (Choose all that apply)

A. Inheritance is a feature of object-oriented programming

B. You implement inheritance using the extend keyword

C. All methods in a parent class can be used in a child class

D. Certain fields in a parent class can be accessed in a child class.

34 Question (7.3. Determine when casting is necessary) ?

Consider this code.

```
long l = 432123839843423434L;
int i = (int) l;
long l2 = (long) i;
byte b = (byte) i;
short s = (short) b;
short t = (short) s;
```

Which explicit castings are not optional? (Choose all that apply)

A. int i = (int) l;

B. long l2 = (long) i;

C. byte b = (byte) i;

D. short s = (short) b;

E. short t = (short) s;

35 Question (8.5. Recognize common exception classes (such as NullPointerException. ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException)) ?

Which of the following are runtime exceptions? (Choose all that apply)

A. CannotLoadException

B. NullPointerException

C. IllegalArgumentException

D. NumberFormatException

E. OutOfMemoryError

36 Question (8.3. Describe the advantages of Exceptions handling) ?

Which of the following statements are true?

A. A potential error can be isolated using the **try** statement

B. The **try-with-resources** statement closes all resources that implements **AutoCloseable**

C. You should catch all errors in your program

D. All exception classes are subclasses of **java.lang.Exception**

37 Question (8.2. Create a try-catch block and determine how exceptions alter normal program flow) ?

Consider the following code fragment.

```
package rewardprogram;

public class Reward {
    public static void main(String[] args) {
        int j = 0;
        try {
            j = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            System.out.println("Incorrect number format");
        } catch (NullPointerException e) {
            System.out.println("Null argument");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("No argument found");
        }
        System.out.println(j);
    }
}
```

What is printed on the console if the class is invoked without an argument.

A. No argument found

B. Null argument

C. 0

D. Incorrect number format

38 Question (7.4. Use super and this to access objects and constructors) ?

Given

```
package test;
class Super {
    @Override
    public String toString() {
        return "Super " + this.toString();
    }
}

class Sub extends Super {
    @Override
    public String toString() {
        return "Sub " + super.toString();
    }
}

public class Tester {
    public static void main(String[] args) {
        Sub sub = new Sub();
        System.out.println(sub);
    }
}
```

What happens when the class **Tester** is executed?

A. An error will be thrown

B. It will print "Sub Super Object"

C. It will print "Sub Super"

D. It will print "Sub Super" followed by a random string

39 Question (5.3. Create and use do/while loops) ?

Given

```
package com.example.test;
import java.util.ArrayList;
import java.util.List;

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

```
        List<String> list = new ArrayList<>();
        String[] favCities = { "Berlin", "Boston", "Chicago" };
        for (String city : favCities) list.add(city);
        do {
            System.out.print(list.get(0) + "--");
            list.remove(0);
        } while (list.size() > 0);
    }
}
```

What is the result of running this class?

A. Berlin--Boston--Chicago--

B. An empty string

C. Berlin--Boston--

D. Berlin--

40 Question (3.4. Use a switch statement) ?

What is the output of the following code if ?

```
public class Util {
    public static void main(String[] args) {
        int a = args.length;
        switch (a) {
        case 0:
            System.out.print("No argument");
        case 1:
            System.out.print("One argument");
            break;
        case 2:
            System.out.print("Two arguments or more");
            break;
            default:
                System.out.print("Default");
        }
    }
}
```

A. No argument

B. One argument

C. No argumentOne argument

D. Default

41 Question (5.5. Use break and continue) ?

What is the output of the following class?

```
public class FunnyGame {
    public static void main(String[] args) {
        int b = 15;
        Loop1:
        for (int a = 0; a < 5; a += 2) {
            Loop2:
                b--;
                for (int c = 6; c > 0; c--) {
                    if (c==4) continue;
                    System.out.print(c+" ");
                    if (c==3) continue Loop1;
                    if (c==2) break;
                }
        }
    }
}
```

A. 6 5 3 2 6 5 3 2 6 5 3 2

B. 6 5 3 6 5 3 6 5 3

C. 6 5 3 6 5 3 6 5 3 5

D. 6 5 3 6 5 3 6 5 2

42 Question (5.4. Compare loop constructs) ?

Consider the following code snippet that is supposed to print 234:

```
1. ...
2. {
3.     System.out.print(a++);
4. }
```

Which of the following statements can be inserted into line 1 so that the code does what is expected of it?

A. for (int a=2; i<5;)

B. int a=2; while(a<5)

C. for (int a=2; a<5; a++)

D. for (int a=2; i<5; i++)

43 Question (6.2. Apply the static keyword to methods and fields) ?

What will happen if you try to compile and run this code?

```
package test;
class Bottle {
    public static int weight = 10;
}
public class Bartender {
    public static void main(String[] args) {
        Bottle bottle = new Bottle();
        bottle.weight = 123;
        Bottle.weight = 654;
        System.out.println(bottle.weight);
    }
}
```

A. You will get a compilation error for trying to access a static field as an instance field

B. The code will compile fine and it will print 123

C. The code will compile fine and it will print 654

D. The code will throw a runtime error

44 Question (6.4. Apply access modifiers) ?

Which of the following statements are true? (Choose all that apply)

A. Public methods in a protected class can only be accessed from subclasses of the class

B. The private methods of a class cannot be accessed from other top-level classes

C. Static private methods in a class can be access from other classes

D. Methods with the default access can be called from the class's subclasses and other classes in the same package

45 Question (6.5. Apply encapsulation principles to a class) ?

Which of the following are steps to implementing encapsulation? (Choose all that apply)

A. All fields are made private

B. Access to fields are done through public methods

C. Read-only fields must have get methods but not set methods

D. Write-only fields must have get methods but not set methods

46 Question (8.1. Differentiate among checked exceptions, unchecked exceptions, and Errors) ?

Which of the following statements are true with regard to exception handling in Java?

A. A well-written program should try to catch both checked exceptions and unchecked exceptions

B. Unchecked exceptions are runtime exceptions

C. Errors are caused by external factors to the program

D. Errors should be caught

47 Question (6.6. Determine the effect upon object references and primitive values when they are passed into methods that change the values) ?

Consider this code:

```
package com.example.geniemanager;
import java.util.ArrayList;
import java.util.List;
class Genie {
    public String grantWishes(int requestCode, List<String> wishes) {
        for (int i = wishes.size() - 1; i >= 0; i--) {
            String wish = wishes.get(wishes.size() - 1);
            // do something to grant wish
            wishes.remove(wish);
        }
        requestCode++;
        return "OK";
    }
}
public class GenieManager {
    public static void main(String[] args) {
        Genie genie = new Genie();
        List<String> wishes = new ArrayList<>();
        wishes.add("I want to pass all exams without studying");
        wishes.add("I want a flashy electric car");
        byte requestCode = 100;
        genie.grantWishes(requestCode, wishes);
        System.out.println(requestCode);
        for (String wish : wishes) {
            System.out.println(wish);
        }
    }
}
```

What are the variable values right after the for loop in **main** is executed?

A. The value of requestCode is 100

B. The List contains two elements

C. The List is empty

D. The value of requestCode is 101

48 Question (7.5. Use abstract classes and interfaces) ?

Which of the following are correct implementations of the interface **WithWings**?

```
public abstract interface WithWings {
    void fly(int speed);
}

class DragonFly implements WithWings { }

abstract class Bird implements WithWings {
}

abstract class Bat implements WithWings {
    public void fly(int speed) {}
}

abstract class Moth implements WithWings {
    public abstract void fly(int speed);
}

final class Bee implements WithWings {
    protected void fly(int speed) {}
}
```

A. Dragonfly

B. Bird

C. Bat

D. Moth

E. Bee

49 Question (7.2. Develop code that demonstrates the use of polymorphism; including overriding and object type versus reference type)

Consider the following code:

```
 1. class Combatant {
 2.    public void fight() {}
 3. }
 4. class Gladiator extends Combatant {
 5.     public void changeEmployer() {}
 6. }
 7.
 8. public class Test1000 {
 9.     public static void main(String[] args) {
10.         Combatant c1 = new Combatant();
11.         Combatant c2 = new Gladiator();
12.         Gladiator c3 = new Gladiator();
13.         ...
14.     }
15. }
```

What can be inserted into line 13 without generating a compilation error?

A. c1.changeEmployer();

B. c2.changeEmployer();

C. (Gladiator)c2.changeEmployer()

D. ((Gladiator)c2).changeEmployer();

50 Question (9.4. Declare and use an ArrayList of a given type)

What is the output of the following code?

```
public class Numerator {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            numbers.add(i + numbers.size());
        }
        for (int i = 0; i < 5; i++) {
            if (numbers.contains(i))
                numbers.remove(new Integer(i));
        }
        System.out.println(numbers);
    }
}
```

A. [1, 2, 3, 4, 5]

B. [1, 3, 5, 7, 9]

C. [9, 7]

D. [6, 8]

51 Question (9.5. Write a simple Lambda expression that consumes a Lambda Predicate expression)

Given

```
Predicate<Void> xyz = (x) -> 3 == 1;
```

Which statement(s) can be used to invoke the Predicate?

A. xyz.test(null);

B. xyz.test("");

C. xyz.test();

D. xyz.test(xyz);

52 Question (9.1. Manipulate data using the StringBuilder class and its methods)

Given

```
public class StringUtil {
    public static String process(String s) {
        StringBuilder sb = new StringBuilder();
        for (int i = s.length() - 1; i >= 0; i--) {
            sb.append(i % 2 == 0? Character.toUpperCase(s.charAt(i)) :
                    s.charAt(s.length() - i));
        }
        return sb.toString();
    }
    public static void main(String[] args) {
        String s = "New Age";
        System.out.println(process(s));
    }
}
```

What is the output of the class?

A. NEW AGE

B. Ega weN

C. EEgnwen

D. EwAAWeN

53 Question (3.4. Use a switch statement) ?

Consider the following code:

```
public class DataAccess {
    public static void main(String[] args) {
        String s = "\\\\temp\\\\data";
        switch (s.substring(1, 2)) {
        case "\\":
            s = "slash" + s;
        case "\\\\":
            s += "\\\\temp";
            break;
        case "data":
            s += "data";
            break;
        default:
            s += "default";
        }
        System.out.println(s);
    }
}
```

What is printed on the console when the class is executed?

A. \\

B. \\\\

C. slash\\temp\\data\\temp

D. temp\data

54 Question (8.2. Create a try-catch block and determine how exceptions alter normal program flow) ?

What is the result of the following class if invoked without an argument?

```
 1. public class TryCatchFinally {
 2.     public static void main(String[] args) {
 3.         String s = null;
 4.         try {
 5.             s = args[0];
 6.         } catch (Exception e) {
 7.             try {
 8.                 s = "exception ";
 9.                 throw new Exception();
10.             } catch (Exception x) {
11.                 s = s + "exception ";
12.             } finally {
13.                 s = s + "finally ";
14.             }
15.         } finally {
16.             s += "finally ";
17.         }
18.         System.out.println(s);
```

```
19.     }
20. }
```

A. Compilation error

B. Uncaught exception

C. exception exception

D. exception exception finally finally

E. exception finally exception finally

55 Question (4.1. Declare, instantiate, initialize and use a onedimensional array) ?

What is the result of this code?

```
package animalkingdom;
public class Dog {
    private String name;
    public Dog(String name) {
        this.name = name;
    }
    @Override
    public String toString() {
        return name;
    }
    public static void main(String[] args) {
        Dog dog1 = new Dog("Wanda");
        Dog dog2 = new Dog("William Biteman");
        Dog[] dogs = { dog1, dog2 };
        dog1 = new Dog("Shark");
        dog2 = new Dog("Sherri");
        for (Dog dog : dogs) {
            System.out.print(dog + " ");
        }
    }
}
```

A. Compilation error

B. Runtime exception

C. Wanda William Biteman

D. Shark Sherri

56 Question (3.1. Use Java operators; including parentheses to override operator precedence) ?

What is the result of this code?

```
public class Prince {
    public static void main(String[] args) {
        System.out.print(" " + 66 + 2 + " ");
        System.out.print(66 + 2 + " ");
        System.out.print(" " + (66 + 2) + " ");
        System.out.print(66 + 2);
    }
}
```

A. 68 68 68 68

B. 662 662 68 68

C. 662 68 68 68

D. 662 662 662 662

57 Question (5.1. Create and use while loops) ?

What is the result of the following code?

```
public class WhileTest {
    public static void main(String[] args) {
        String s= "newworld";
        int i = s.length();
        while (!s.isEmpty()) {
            i++;
            s = s.substring(1);
```

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

A. NullPointerException

B. newworld

C. 18

D. 16

58 Question (5.3. Create and use do/while loops) ?

What is the result of the class?

```
class Roll {
    public static void main(String[] args) {
        String s= "java";
        do {
            System.out.print(s);
            s = s.substring(0, s.length() - 1);
        } while (s.length() > 1);
    }
}
```

A. javajavajava

B. javajavajav

C. javajavj

D. javajavja

59 Question (5.2. Create an use for loops including the enhanced for loop) ?

Given

```
import java.util.ArrayList;
import java.util.List;

public class Kitty {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Jay");
        names.add("James");
        names.addAll(names);
        names.addAll(names);
        names.remove(4);
        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

How many names are printed on the console?

A. 7

B. 4

C. 3

D. 8

60 Question (9.3. Create and manipulate calendar data using classes from java.time.LocalDateTime, java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period) ?

What is the output of this code?

```
LocalDate start = LocalDate.of(2015, 10, 13);
LocalDate finish = LocalDate.of(2015, 9, 15);
Period period = Period.between(start, finish);
System.out.println(period.getDays());
```

A. -29

B. 29

C. -28

D. 28

61 Question (7.4. Use super and this to access objects and constructors) ?

Given

```
class Tier1 {
    public void print() {
        System.out.print("Tier 1...");
    }
}
class Tier2 extends Tier1 {
     public void print() {
         super.print();
         System.out.print("Tier 2...");
     }
}

class Tier3 extends Tier2 {
    public void print(String name) {
        super.print();
        System.out.print("Tier 3...");
    }
}
public class Tiery {
    public static void main(String[] args) {
        Tier2 tier = new Tier3();
        tier.print();
        tier = (Tier3) tier;
        tier.print();
    }
}
```

What is printed on the console when the class is run?

A. Tier1...Tier2...Tier3...

B. Tier1...Tier2...Tier1...Tier2...

C. Tier1...Tier2...Tier1...Tier2...Tier3...

D. Tier1...Tier2...Tier3...Tier1...Tier2...Tier3...

62 Question (2.4. Explain an object's lifecycle (creation, "dereference by assignment" and garbage collection)) ?

Given

```
 1. public class Solution {
 2.     private void print() {
 3.         System.out.println(new Solution());
 4.         System.out.println(new Solution());
 5.     }
 6.     public static void main(String[] args) {
 7.         Solution solution = new Solution();
 8.         solution.print();
 9.
10.     }
11. }
```

At line 9, how many objects are eligible for garbage collection?

A. 0

B. 1

C. 2

D. 3

63 Question (7.1. Describe inheritance and its benefits) ?

Which of the following statements are true? (Choose all that apply)

A. A class can implement another class

B. An interface can extend another interface

C. A class can extend any class

D. A class can implement multiple interfaces

64 Question (5.5. Use break and continue) ?

What is the result of this code?

```
public class LoopTest {
    public static void main(String[] args) {
        int i = 0;
        while (true) {
            if (i == 1) continue;
            if (i == 2) break;
            System.out.print(i + " ");
            i++;
        }
        System.out.println("done");
    }
}
```

A. 1

B. 1 2

C. 1 done

D. Indefinite loop

65 Question (2.1. Declare and initialize variables (including casting of primitive data types) ?

Given

```
1. public class USB {
2.     public static void main(String[] args) {
3.         int e = 0;
4.         float f = 123.3f;
5.         float h = f + (int) 12.3;
6.         float j = f = h;
7.         float k = j + 100.00;
8.     }
9. }
```

Which line(s) of code contain invalid declaration?

A. Line 4

B. Line 5

C. Line 6

D. Line 7

66 Question (9.2. Creating and manipulating Strings) ?

What is the output of the following class?

```
package test;
public class Splitter {
    public static void main(String[] args) {
        String a = "aabraacadabra";
        String[] tokens = a.split("a");
        System.out.print(tokens.length);
        for (String token : tokens) {
            System.out.print(token + "|");
        }
    }
}
```

A. 6b|r|c|d|b|r|

B. 7||br||c|d|br|

C. 13a|a|b|r|a|a|c|a|d|a|b|r|a|

D. 8a|b|r|a|c|d|b|r|

67 Question (2.5. Develop code that uses wrapper classes such as Boolean, Double, and Integer) ?

What are the wrapper classes for char, short, int, long and boolean, respectively?

A. Char, Short, Int, Long and Bool

B. Character, Short, Int, Long and Boolean

C. Character, Short, Integer, Long and Bool

D. Character, Short, Integer, Long and Boolean

68 Question (6.4. Apply access modifiers) ?

Which access modifiers can be used for constructors?

A. public

B. protected

C. private

D. static

69 Question (6.6. Determine the effect upon object references and primitive values when they are passed into methods that change the values) ?

Which of the following statements are true?

A. Objects and arrays are passed by reference

B. Primitives are passed by value

C. Arrays of primitives are passed by value

D. Arrays of objects are passed by reference

70 Question (4.2. Declare, instantiate, initialize and use multidimensional array) ?

Consider the following code snippet:

```
public class Image {
    public static void main(String[] args) {
        int[][] pixels = {{0, 1, 0, 1}, {2, 2, 2}, {5, 6, 5}};
        int rows = pixels.length;
        int cols = pixels[0].length;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; i < cols; j++) {
                System.out.print(pixels[i][j]);
            }
        }
    }
}
```

What is the result of the code?

A. Compilation error

B. The code throws an exception

C. 0101222565

D. 0251260250

71 Question (6.1. Create methods with arguments and return values) ?

Which of the following shows the signature of a method that takes an **int** array and returns no value? (Choose all that apply)

A. public void method1(int... dots)

B. public null method2(int[] array)

C. private void method3(int... varargs)

D. method4(int[] array) as void

72 Question (8.1. Differentiate among checked exceptions, unchecked exceptions, and Errors) ?

Which of the following is not an exception category?

A. Checked exceptions

B. Bugs

C. Unchecked exceptions

D. Errors

73 Question (7.3. Determine when casting is necessary) ?

Consider the following code:

```
class Parent {}
class Child extends Parent {}
class GrandChild extends Child {}
```

Which of the following will generate a compile error? (Choose all that apply)

A. Parent p = new Child();

B. Child c = (Child) new Parent();

C. GrandChild g = (GrandChild) new Parent();

D. g = (Child) c;

74 Question (6.2. Apply the static methods and fields) ?

Which of the following statements are true?

A. A static method is shared by all instances

B. A static field is a good way to store a value that is unique throughout all instances.

C. A static method can be called from an instance method in the same class.

D. An instance method can be called from a static method defined in the same class.

75 Question (2.3. Know how to read or write to object fields) ?

Examine the following code.

```
public class Coder {
    private static short x;
    public long y;

    public static void main(String[] args) {
        Coder s = new Coder();
        x = (short) 7;
        y = 10;
        s.y = 12478;
        s.x = 100;
    }
}
```

Which of the following cause(s) a compile error?

A. x = (short) y;

B. y = 10;

C. s.y = 12478;

D. s.x = 100;

76 Question (6.5. Apply encapsulation principles to a class) ?

A Java class has a field (named **x** and of type **long**) that needs to be protected against direct access from other classes. Which of the following line(s) of code complies with the OOP encapsulation principles?

A. private long x;

B. public void setX(long value)

C. public void getX()

D. protected long x;

77 Question (9.4. Declare and use an ArrayList of a given type)

Which statement(s) creates an **ArrayList** of **File**s with an initial capacity of 36? (Choose all that apply)

A. ArrayList<File> names = new ArrayList<>();

B. List<File> names = new ArrayList<>(36);

C. List<File> names = new List<File>();

D. ArrayList<File> names = new ArrayList<File>(36);

Answers

1 **D**.

Program arguments are not evaluated. Thus, 20+30 is passed on to the **main** method as is.

2 **D**.

You can create a public static void **main** method to make a class executable. However, nothing prevents you from naming a non-static non-public method **main**.

3 **C**.

The first element of **args** is args[0]. The program therefore prints the second and third arguments. Quotes around an argument are removed before the argument is passed to the program.

4 **A, B, C**.

You can use members of the **java.lang** package without importing them, so you do not need to import **java.lang.System** even though doing so will not generate a compile error. On the other hand, members of other packages must be imported before use. You can import the whole package by using the asterisk as in option A. Alternatively, you can import individual types as in option B.

5 **C, D**.

You can use members of the **java.lang** package without importing them, but members of its subpackages, such as **java.lang.annotation**, must be imported before use. Therefore, you need to import **java.lang.annotation.Documented** and the two members of **java.util**. You can import the whole package by using the asterisk as in D. Alternatively, you can import individual types as in C.

6 **C**.

A and B are incorrect because arrays do not have a member called **size**.

C is correct because every array object has a field named **length**, which contains the number of elements in the array.

D is incorrect because an array object has no **length** method.

E and F are incorrect because array objects have no member called **capacity**.

7 **B**.

A **char** can be assigned to an **int** without casting. However, an **int** cannot be assigned to a **char**.

8 **D**.

A is incorrect because Java 8 interfaces may contain static methods.

B is incorrect because starting from Java 8, interfaces may contain implementations in default methods.

C is incorrect because by default all methods are public even when no access modifier is present.

D is correct because the interface compiles without issues.

9 **A, B, C**.

A is correct. It will print TRUE if the class is invoked with at least one argument that consists of numbers only.

B is correct. It will print FALSE if the class is invoked with at least one argument and the argument is not a whole number.

C is correct. It will throw an exception if the class is run without an argument.

D is incorrect as the code compiles without issues.

10 **C**.

The code inspects each character in the **StringBuilder** and upon encountering an 'e' calls this line:

```
buffer.replace(i - 1, I, "E");
```

This replaces the character to the left of every 'e' with E.

11 **B**.

The **average** method takes an array of **Integer**s. The **main** method passes it an array of **int**s. Since an array of **int**s is not compatible with an array of **Integer**s, the class will generate a compile error.

12 **A, C**.

The **run** and **brake** methods are overloaded methods, so A and C are correct.

B is incorrect because you can overload an instance and a static methods.

D is incorrect. The **toString** method overrides the method in java.lang.Object and is not an overload of it.

E is incorrect since the class compiles fine.

13 **A, B, C**.

A, B and C are correct as comments, methods and fields can appear in a Java class.

D is incorrect because a Java class cannot embed files.

E is incorrect because import statements cannot appear within a class.

14 **B, D**.

A is incorrect because Java is unlike the conventional programming paradigm. In fact, a Java program is compiled to bytecode and not an executable file. This bytecode can only run on a Java virtual machine.

B is correct, platform independence is one of the allures of Java that make Java very popular.

C is incorrect. To achieve platform independence, Java programs are compiled into bytecodes, which can run on a Java virtual machine. By making JVMs available in multiple platforms, the same bytecode can be run in different operating systems, including Windows, Linux, Unix and Mac OS X.

D is correct. Java programs are compiled into bytecodes.

15 **D**.

257 is too big for a byte and the value is truncated to 1 when narrowing conversion occurred.

16 **C, D**.

Applied on reference types, the == operator tests if two variables are pointing to the same object. In this case, **d** and **e** are referring to different objects. Therefore, d==e evaluates to false and A is incorrect.

B is also incorrect because **f** and **g** are assigned different objects that are created by boxing the primitive value 12.34. f==g therefore evaluates to false.

C is correct as the **equals** methods in **String** and primitive wrapper classes compares the value of an object with the value of the current object.

For the same reason, D is also correct.

17 **B, C**.

A is incorrect because the **==** operator on reference variables test whether the variables are pointing to the same object. In this case, they are clearly not because **a** and **b** are assigned different objects (even though both have the same value).

B is correct because comparing a **Long** and a **long** forces the former to be unboxed and the primitive value retrieved from the unboxing process to be compared to the **long**.

C is correct because two **long**s having the same value are equal.

D is incorrect because a==d evaluates to true. Therefore, a!=d evaluates to false.

18 **C**.

D is incorrect because 0 in the array declaration in the **main** method is boxed to an **Integer**. As such, the **average** method will not throw an exception.

C is correct. The value of **Integer.MIN_VALUE** is -2,147,483,648 and the value of Integer.MAX_VALUE is 2,147,483,647, so Integer.MIN_VALUE + Integer.MAX_VALUE = -1. Since there are four members in numbers, the average is -0.25 (1 / 4).

A and B are incorrect because the output is -0.25.

19 **A, B, C**.

A is correct because all classes are implicitly derived from **java.lang.Object**. As such, an instance of any class is implicitly an instance of **Object**.

B is correct because **Entertainment** implements **java.io.Serializable**. As such, an instance of **Entertainment** is a type of **Serializable**.

C is correct for the obvious reason that the object you get by instantiating a class is an instance of that class.

D is incorrect as **Object** is not a type of **Entertainment**.

E is incorrect because **Object** and **Serializable** have no parent-child relationship.

20 **B, C, D**.

A is incorrect because the **String** class's **equals** method compares two strings case-sensitively.

B, C and D are correct. You can either use **equalsIgnoreCase** to compare two strings case-insensitively or convert the strings to either upper case or lower case before comparing.

21 **D**.

The ternary construct will be easier to understand if written with parentheses:

```
args.length > 0 ? (args[0].equals("file")? "accept" : "ignore")
        : "invalid";
```

Since the class is invoked without arguments, **args.length** is 0 and the expression in the parentheses will be ignored and "invalid" printed.

22 **C**.

Since **a.length** = 6, the initial value of **i** is 3 and the loop iterates three times. The code prints 405010.

23 **C, D**.

A and B are incorrect because there is no guarantee if or when an object that is eligible for garbage collection will be garbage-collected. The garbage collector runs on a low-priority thread and an intelligent garbage collector will not start destroying objects unless the heap is close to full.

An object is eligible for garbage collection if it is no longer referenced by any variable. Assigning null to a reference variable removes the reference to an object. C is correct because if the class is executed with at least one argument, **s1** will be null at line 5 and the object it referenced will be eligible for garbage collection.

D is also correct. If the class was executed with at least two arguments, **s2** will be assigned a new object and the old object will be eligible for garbage collection.

24 **B**.

The Event class has two non-default constructors. As such, the compiler will not create a default constructor for it. As a result, attempting to create an **Event** by using this statement will generate a compilation error:

```
Event e3 = new Event();
```

25 **A, C**.

A is correct because the scope of **n** is only in the **for** block. Therefore, **n** cannot be used in region F.

B is incorrect because **n** is not yet declared in region B.

C is correct because **m** and **n** are visible in region D.

D is incorrect because **n** is not visible in region F.

26 **D**.

The **name** field in the **Event** class can be set directly because it is public. The **setDescription** method looks like it has been designed to write to the **description** field. However, there is a bug. The **this** keyword in **this.description** refers to the **description** field. As such, this line of code in **setDescription** does not change the value of the **description** field.

```
this.description = this.description
```

27 **D**.

This is an easy question if you know the difference between i++ and ++i:

The expression i++ returns the value of i and then increments i. By contrast, the expression ++i increments i and then returns its value. Therefore, (++i + i) is equal to (4 + 4) whereas (j++ + j) is equal to (3 + 4).

28 **C**.

Note that **++n** increments **n** and then return its value. In other words, in the code above, switch (++n) is equivalent to switch (101). Since there is no case for 101, the default case is executed.

29 **D**.

matrix[0][0][0] is 1 and **matrix[1][1][1]** is 50. Therefore, 50 will be printed on the console when the program is run.

30 **C**.

The question asks the length of **namesCities**. These loops add elements to the **List:**

```
for (String name : names) {
    mix.add(name);
    for (String city : cities) {
        mix.add(city);
    }
}
```

The outer loop iterates twice, adding two elements to **mix**. For each iteration, the inner **for** loop is called, adding two more elements each time. Therefore, the number of elements in **mix** is six after the outer loop terminates, and so is the length of **namesCities**.

31 **B**.

Note that the pattern is very unusual. mm represents minutes and MM the month. This pattern basically tells the date time to be displayed in this format: year minute date hour:month.

32 **A**.

The **ArrayList** can only store **Long**s and cannot be given an **int** (of value 459).

33 **Answer A, D**.

A is correct because inheritance is a feature of OOP.

B is incorrect. You use the keyword **extends** to create a subclass, not **extend**.

C is incorrect. Private methods in a parent class are not accessible from within a child class.

D is correct. Only protected and public fields in a parent class are guaranteed to be accessible from a child class. In addition, default fields in a parent class can also be accessed from a child class if the child class resides in the same package as the parent class.

34 **A, C**.

Narrowing conversions require explicitly casting.

35 **B, C, D**.

A is incorrect as it does not exist.

B, C and D are correct as **NullPointerException, IllegalArgumentException** and **NumberFormatException** are derived from **RuntimeException**.

E is incorrect because **OutOfMemoryError** is an error and not a runtime exception.

36 **A, B**.

C is incorrect because you should not try to catch serious errors such as **java.lang.StackOverflowError** and **java.lang.OutOfMemoryError**.

D is incorrect because all exception classes are derived from **java.lang.Throwable**.

37 **A**.

The **main** method of a class invoked without an argument will received an empty String array. Since the array is empty, trying to read an argument in it will throw an **ArrayIndexOutOfBoundsException**.

38 **A**.

The class will throw a **java.lang.StackOverflowError** because the **toString** method in **Super** calls itself recursively until a stack overflow occurs.

39 **A**.

The class starts by adding the elements in the string array to the **List**. It then uses a do-while loop to iterate over the **List**, removing the first element at each iteration until there is no more element in the **List**.

40 **C**.

If the class is invoked without an argument, **args.length** will be 0 and the first case of the **switch** statement will be executed. Because there is no break at the end of the code for the case, the case next to it will also be called.

41 **B**.

The second for loop will iterate until **c** is equal to 3 and then continue to the outer loop. As a result, the second loop will only iterate three times, printing 6 5 3. The outer loop iterates three times (for a = 0, a = 2 and a = 4), so 6 5 3 will be printed three times.

42 **A, B**.

A and B are correct.

C is incorrect because a is incremented twice.

D is incorrect because it uses variable i without declaring it. It will generate a compile error.

43 **C**.

You can access a static field as an instance field, however the field will still be shared by all instances.

44 **B**.

A is incorrect because a method's accessibility is determined by the method's access modifier, and not by the access modifier of its enclosing class. As long as you can instantiate the enclosing class, you can access its public methods.

B is correct because private methods can only be accessed from within the class itself.

C is incorrect because private methods can only be accessed from within the class itself, even though the methods are static.

D is incorrect because methods with the default access can only be accessed from other classes in the same package.

45 **A, B, C**.

A, B and C are correct.

D is incorrect because write-only fields must have set methods but not get methods.

46 **B, C**.

A is incorrect because programmers are supposed to catch checked exceptions only.

B and C are correct.

D is incorrect because errors are not supposed to be caught.

47 **A, C**.

A is correct because **requestCode** in the **main** method is not affected by the change made in **grantWishes**.

B is incorrect because the **List** is emptied and since **List** is a reference type, the changes made in a method affect the object passed. The **List** is empty after **grantWishes** is executed.

C is correct.

D is incorrect because primitives passed to a method are first copied. Changes are only made to the copies.

48 **B, C, D**.

A is incorrect because **DragonFly** implements **WithWings** but does not provide an implementation for its **fly** method.

B is correct. Even though the class **Bird** implements **WithWings** and does not provide an implementation for its **fly** method, the class itself is declared abstract. An abstract class does not have to provide implementations for the methods of the interfaces it implements.

C and D are also correct for the same reason B is correct.

E is incorrect because the **fly** implementation in **Bee** reduces the visibility of the method from public to protected. This is illegal in Java.

49 **D**.

A is incorrect because **c1** is pointing to an instance of **Combatant** and the **Combatant** class does not have a **changeEmployer** method.

B is incorrect because even though **c2** is pointing to an instance of **Gladiator, c2** is of type **Combatant**, which does not provide a **changeEmployer** method.

C is an attempt to downcast **c2** to **Gladiator**. However, its syntax is wrong. C is therefore incorrect.

D is correct, **c2** is downcast to **Gladiator**.

50 **D**.

In the first **for** loop, the values of **i** in the iterations are 0, 1, 2, 3 and 4, for which the values of **numbers.size()** are also 0, 1, 2, 3, 4, respectively. After the first loop, the content of **numbers** is [0, 2, 4, 6, 8]. The second loop will remove elements in numbers with the value of 0, 2 and 4, leaving 6 and 8.

51 **A**.

A **Predicate** that takes **Void** can only be invoked by passing null.

52 **D**.

The length of **s** is 7 and the value of **i** in the for loop starts with 6 and ends with 0. The sequence of characters returned by the **process** method is determined by this expression:

```
i % 2 == 0
```

For i = 6, the expression returns true, so the uppercase of the sixth character in **s** (E) is added to the **StringBuilder**.

For i = 5, i%2==0 returns false, so s.charAt(7-5), which is 'w', is added to the StringBuilder.

Since D is the only option that starts with Ew, D is correct.

53 **C**.

"\\\\temp\\\\data".substring(1, 2) returns a backslash, which has to be escaped with another backslash if it is to be written as a Java string. Therefore, the first case, "\\" will be executed, which appends the string "slash" to **s**. At this point, the value of **s** is "slash\\\\temp\\\\data". Since there is no break at the end of the first case, execution continues to the second case, appending "\\\\temp" to **s**. As a result, the value of **s** is now "slash\\\\temp\\\\data\\\\temp". Note that "\\\\" is printed simply as \\.

54 **D**.

Since the class is invoked without an argument, calling args[0] at line 5 will throw an **ArrayIndexOutOfBounds** exception, causing the catch block that starts at line 6 to be executed. As a result, at line 8, **s** is assigned "exception" and line 9 throws a new exception. Due to the new exception, the caught block that starts at line 10 is called, followed by the finally block at line 12. Finally, the finally block of the outer try is called.

55 **C**.

The dogs named Wanda and Wiliam Biteman are added to the array, then the reference variables for both dogs are assigned new dogs. However, this does not change the fact that the array contains the original dogs that were added to it. As a result, Wanda William Biteman is printed on the console.

56 **C**.

Adding a string with an integer results in a string. Therefore, " " + 6 returns "6". Therefore, the first **print** method prints " 662 ". The second print method evaluates 66 + 2 before concatenating the result with " ". As a result, it prints "68 ". The third and fourth calls to print evaluates the addition before printing the result.

57 **D**.

The code starts with i = 8, which is the length of the string **s. s.substring(1)** reduces the length by 1. Each time it happens, i is incremented. Because the initial length of s is 8, the final value for **i** is 16.

58 **D**.

The **substring** method in the code cuts the last character in the string. As a result, javajavja is printed on the console.

59 **A**.

The list is created empty and two names are added to it:

```
List<String> names = new ArrayList<>();
names.add("Jay");
names.add("James");
```

Next, the **addAll** method adds all members in the collection. In effect, this doubles the number of elements in **names** to four. It then calls **addAll** for the second time, doubling the number of elements to eight. Finally, the code removes the element at index 4. As a result, just before the for loop is executed, **names** contains 7 elements.

60 **C**.

The **between** method returns the difference between its two arguments, and there are 28 days between September 15 and October 13. Since **finish** is earlier than **start**, the result is negative.

61 **B**.

An instance of **Tier3** is assigned to a reference variable of type **Tier2**. Calling **print** on the object will invoke the **print** method in **Tier2**, not **Tier3**. Then, an attempt is made to downcast the variable to **Tier3**, but it has no effect since **tier** is already an instance of **Tier3** and, more importantly, it is not assigned to a variable of type **Tier3**. As a result, during the execution of the class, the **print** method defined in **Tier2** is called twice.

62 **D**.

At line 8 there are three **Solution** objects created, including two instances created by the **print** method. At line 9, none of the objects is referenced by any variable. As a result, all of them are eligible for garbage collection.

63 **B, D**.

A is incorrect because a class can implement interfaces, not another class.

B is correct because an interface can extend another interface.

C is incorrect because a class cannot extend a final class.

D is correct. A class can implement multiple interfaces.

64 **D**.

The **continue** statement in the **while** loop moves control to the beginning of the loop and causes an indefinite loop.

65 **D**.

Options A, B and C are valid. Option D is invalid as 100.00 is a **double** and j + 100.00 is also a **double**. Assigning a **double** to a **float** is invalid.

66 **B**.

The string is split into tokens using "a" as the delimiter. Two consecutive characters that are equal to the delimiter will result in an empty string. As such, there will be seven tokens since there are seven "a"s.

67 **D**.

The wrapper classes for char, short, int, long and boolean are **Character, Short, Integer, Long** and **Boolean**, respectively.

68 **A, B, C**.

Constructors can have the following access modifiers: public, protected, the default access modifier and private. Therefore, A, B and C are correct.

D is incorrect because **static** is not an access modifier.

69 **A, B, D**.

Objects are passed by reference and primitives are passed by value. Arrays (even arrays of primitives) are objects so arrays are also passed by reference.

70 **B**.

The code author assumed that all rows would have the same number of elements as the first row, which is not the case. The second and third rows only have three elements instead of four. As a result, an **ArrayIndexOutOfBoundsException** is thrown.

71 **A, C**.

A and C are correct. A varargs argument is basically an array.

B is incorrect because null is not a valid return value.

D is incorrect because of the syntax.

72 **B**.

Java exceptions fall into three categories: checked exceptions, unchecked exceptions and errors. Bugs are not exceptions.

73 **D**.

A will not cause a compile error because it is legal to assign an instance of a subclass to a reference variable that is of the type of the parent class.

B will not cause a compile error either, but will throw a **ClassCastException** because **Parent** cannot be downcast to **Child**.

C will not cause a compile error either, but will throw a **ClassCastException** because **Parent** cannot be downcast to **GrandChild**.

D will generate a compile error because Java does not permit the casting of a class to a subclass.

74 **A, C**.

Static methods and static fields are shared by all instances. A is therefore correct.

B is incorrect because a static field is shared by all instances.

C is also correct. A static method can be invoked from an instance method.

D is incorrect because an instance method needs an object. A static method cannot call an instance method.

75 **B**.

A is not a correct option because **x** is a static method so it can be set from a static method. And, even though it is private, it can be accessed from the same class, so it will not cause a compilation error.

B is correct. **y** is an instance field so it cannot be accessed from a static method without an instance of the class encapsulating **y**.

C and D will not cause a compile error because both **x** and **y** can be accessed through an instance even though **x** is static.

76 **A, B**.

To encapsulate a piece of data, a field needs to be made private and access to it is done through a get and a set methods. A and B are therefore correct.

C is incorrect because a get method should return a value;

D is incorrect because this would make **x** accessible from subclasses and from classes in the same package.

77 **B, D**.

By default, an **ArrayList** is created with an initial capacity of 10 elements. To reserve a space for 36 elements, pass 36 to the constructor. A is incorrect because the resulting **ArrayList** will have an initial capacity of 10.

B and D are correct.

C is incorrect because **List** is an interface and cannot be instantiated.