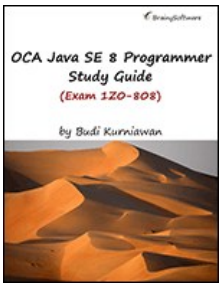

OCA Java SE 8 Programmer Study Guide (Exam 1Z0-808)

by Budi Kurniawan

Appendix A: Mock Exam 1

The following are answers to the questions in the first mock exam. Each question may have more than one correct answer. Each of the questions also shows which exam topic it relates to.

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

The keyword **extends** can be used

A. to create a subclass from another class

B. to create a subinterface from another interface

C. to make a class implement an interface

D. to make an interface implement another interface

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

Given the following class

```
class RocketScience {
    public String description;
    public String getDescription() {
       return description;
    }
}
```

Which of the following statements are correct?

A. The class will not compile because it has no constructor.

B. The class will compile.

C. All Java classes, including abstract classes, need at least one constructor.

D. The class will compile but will throw an exception when instantiated.

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

Given

```
abstract class Abstract {
      public Abstract() {
      }
}
```

Which of these statements is true?

A. The class will not compile because an abstract class must not have a constructor.

B. The class will compile but cannot be instantiated.

C. The class will compile and can be instantiated.

D. The class will not compile because **Abstract** is a keyword and cannot be used as a class name.

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

Which of the following statements about the default package is true.

A. You cannot import a class in the default package.

B. A class in the default package can only be used in other classes in the default package.

C. A class in the default package can be imported into classes in non-default packages.

D. A class in the default package can import classes in non-default packages.

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

Which ones are valid variable declarations?

A. long row_count;

B. long $;

C. long table$;

D. long _g5;

E. int $final_;

F. int static;

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

Here is the content of the **Computer.java** file.

```
1.  package test;
2.
3.  class A {
4.
5.  }
6.
7.  final class b {
8.
9.  }
```

Which of the following statements is (are) true?

A. The code will not compile because the file does not contain a **Computer** class.

B. A compile error at line 7 because a class name must start with a capital.

C. Compile errors at lines 3 and 7 because **Computer.java** may only contain a class with the same name as the file.

D. The code will compile.

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

What is the output of this code snippet:

```
StringBuilder sb = new StringBuilder("Best phone cameras!");
System.out.println(sb.replace(1, 5, "Worst"));
```

A. Worst phone cameras!

B. WorstBest phone cameras!

C. BWorstphone cameras!

D. Worstphone cameras!

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

Consider the following code snippet:

```
1. for (int i=10; i<20; i=i+2) {
2.     ...
3.          System.out.println(i);
4.     }
5. }
```

What can be inserted in line 2 to make the code print 10? (Choose all that apply)

A. if (i == 10) {

B. if (i % 5 == 0) {

C. if (i = 10) {

D. if (i % 2 == 5)

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

Given

```
package com.example;
public class Printer {
```

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

The program is invoked using this command.

```
java com.example.Printer Sonic "Drill" 'Toxic'
```

What is printed on the console?

A. SonicDrill

B. Sonic"Drill"

C. "Drill"'Toxic'

D. DrillToxic

10 Question (3.2. Use Java operators; including parenthesis to override operator precedence) ?

Consider the following code:

```
int a = 2 + 3 * 6;
int b = (2 + 3) * 6;
System.out.println(a + b);
```

What is printed on the standard out upon the execution of the code?

A. 60

B. 50

C. 40

D. 30

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

Consider the following code snippet that contains named regions R, S, T, U, V and X.

```
int m = 0;
// ------- R --------
while (m < 5) {
  // ------- S --------
    for (int n = 0; n < 3; n++) {
     // ------- T --------
       System.out.println(m);
       System.out.println(n);
     // ------- U --------
    }
  // ------- V --------
  m++;
}
// ------- X --------
```

Which of the following statements are true?

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

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

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

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

E. *m* can be used in regions S, T, U and V.

F. *m* and *n* are out of scope in Region X.

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

Given

```
package test;
public class Hello {
```

```
    protected static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Which of the following statements is (are) true?

A. The program does not compile.

B. The program compiles.

C. The program can be executed.

D. The program cannot be executed.

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

Consider this code snippet:

```
int i = 015;
if (i == 13L)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

What will be printed on the console when the code is executed?

A. Nothing will be printed.

B. Equal

C. Not equal

D. Equal Not equal

14 Question (6.4. Apply access modifiers) ?

Given the following class declaration

```
modifier class Pointer {}
```

Which modifiers can be used for class **Pointer?**

A. public

B. private

C. protected

D. final

E. static

F. abstract

15 Question (6.4. Apply access modifiers) ?

Given the following method declaration

```
modifier int calculate(int a, int b)
```

Which modifiers can be used for class **Nested?**

A. public

B. private

C. protected

D. final

E. static

F. abstract

G. strictfp

16 Question (6.4. Apply access modifiers)

Given the following interface declaration

```
modifier interface Device {}
```

Which modifiers can be used for interface **Device?**

A. public

B. private

C. protected

D. final

E. static

F. abstract

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

Given

```
package test;
public class Parent {
    public Parent() {
        System.out.print("Parent...");
    }
    public static void main(String[] args) {
        Child1 c1 = new Child1();
        Child2 c2 = new Child2();
    }
}

class Child1 extends Parent {
    public Child1() {
        super();
        System.out.print("Child1...");
    }
}

class Child2 extends Parent {
    public Child2() {
        System.out.print("Child2...");
    }
}
```

What will be printed on the console when the **Parent** class is executed?

A. Child1...Child2...

B. Parent...Child1...Child2...

C. Child1...Parent...Child2...

D. Parent...Child1...Parent...Child2...

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

Consider the following code

```
package test;
public class Parent {
    public Parent() {
        System.out.print("Parent...");
    }
    public static void main(String[] args) {
        Child1 c1 = new Child1("Joe");
        Child2 c2 = new Child2("Jane");
    }
}

class Child1 extends Parent {
    public Child1(String name) {
        super();
        System.out.print("Child1...");
    }
}
```

```
class Child2 extends Parent {
    public Child2(String name) {
        System.out.print("Child2...");
    }
}
```

Which of the following statements is (are) true?

A. The program will not compile because the **Parent** class does not have a constructor that takes a **String**.

B. The program will not compile because **Child1** and **Child2** do not have a constructor that matches the signature of the only constructor in **Parent**.

C. The program will compile and when executed it will print **Child1...Parent...Child2...**

D. The program will compile and when executed it will print **Parent...Child1...Parent...Child2...**

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

Given

```
package test;
public class Parent {
    public Parent(String name) {
        System.out.print("Parent...");
    }
      public static void main(String[] args) {
          Child1 c1 = new Child1("Joe");
          Child2 c2 = new Child2("Jane");
      }
}

class Child1 extends Parent {
    public Child1(String name) {
        super(name);
        System.out.print("Child1...");
    }
}

class Child2 extends Parent {
    public Child2(String name) {
        System.out.print("Child2...");
    }
}
```

Which of the following statements is (are) true?

A. The program will not compile because the **Parent** class does not have a no-argument constructor.

B. The program will not compile because the constructor in **Child2** will try to call the default constructor in **Parent** and such a constructor does not exist.

C. The program will compile and when executed it will print **Child1...Parent...Child2...**

D. The program will compile and when executed it will print **Parent...Child1...Parent...Child2...**

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

Given

```
1.  package test;
2.  ...
3.  public class MathUtil {
4.      double twoPis = PI * 2;
5.  }
```

To prevent line 4 from causing a compile error, which line must be inserted into line 2?

A. import static Math.PI;.

B. import static java.lang.Math;

C. import static java.lang.Math.PI;

D. import java.lang.Math.PI;

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

Consider the following code snippet:

```
1.  import static java.lang.Integer.MAX_VALUE;
2.  public class Storage {
3.      public static void main(String[] args) {
4.          int i = MAX_VALUE;
5.      }
6.  }
```

What do you have to insert into line 1 so you can use the static field **MAX_VALUE** in the **Integer** class?

A. import static java.lang.Integer.MAX_VALUE;.

B. import static java.lang.Integer;

C. import java.lang.Integer.MAX_VALUE;

D. import java.lang.Integer;

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

Which of the following are valid array declarations? (Choose all that apply)

A. int stats[] = {Integer.MIN_VALUE, Integer.MAX_VALUE };

B. double[] remainders[] = {1, 2, 3};

C. short[] smallNumbers = new short[5];

D. String keys[] = new ArrayList<String>();

23 Question (9.2. Creating and manipulating Strings) ?

Assuming **s** is a **String**, which of the following expressions returns the number of characters in **s?**

A. s.size

B. s.size()

C. s.length

D. s.length()

E. s.chars

F. s.chars()

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

Given the following class:

```
package demo;
public class Demo100 {
    private void doSomething() {
        System.out.println("Hello, World!");
    }

    public static void main(String[] args) {
        doSomething();
    }
}
```

What is the output of the program?

A. The program will print "Hello, World!"

B. The program will print an empty string

C. The program will fail to compile because the doSomething method is private

D. The program will fail to compile because you cannot access a non-static method from a static method

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

Consider the following code snippet:

```
1. package util;
```

```
2. public class NumberUtil {
3.     public int getTodaysNumber() {
4.         private int i = 9;
5.         return i;
6.     }
7. }
```

Which statement is true with regard to the class above?

A. The program will compile

B. The getTodaysNumber method returns 9

C. The getTodaysNumber method can be accessed from static methods

D. The program will not compile

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

Examine the following code:

```
package com.example.test;
public class Test101 {
    public static void main(String[] args) {
        int i = 10;
        System.out.println(i < 11);
    }
}
```

What is the output of this class?

A. 10<11

B. 1011

C. true

D. false

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

Consider this code snippet:

```
 1. public class MathUtil {
 2.     public static int add(int a, int b) {
 3.         a++;
 4.         b++;
 5.         return a + b;
 6.     }
 7.     public static void main(String[] args) {
 8.         int m = 10;
 9.         int n = 20;
10.         int o = add(m, n);
11.         //
12.     }
13. }
```

Which of the following statement(s) is/are true?

A. At line 11 the value of m is 11

B. At line 11 the value of n is 21

C. At line 11 the value of m is 10

D. At line 11 the value of n is 20

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

What can be said about this class?

```
public class Printer {
    private String name;
    public Printer() {
        this("Default");
    }
    public Printer(String name) {
        this.name = name;
    }
```

```
    public static void main(String[] args) {
        Printer printer = new Printer();
    }
}
```

A. The code creates two instances of **Printer**

B. The code creates one instance of **Printer**

C. The code calls the no-argument constructor and then the one-argument constructor

D. The name field will be assigned the value "Default"

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

What is the output of the following class?

```
public class Loop1 {
    public static void main(String[] args) {
        int x = 0;
        while (x < 5)
            while (x < 3)
                System.out.print(x++);
    }
}
```

A. 012 then stops

B. 123 then stops

C. 012 then loops indefinitely

D. 123 then loops indefinitely

30 Question (3.4. Use a switch statement) ?

Given the following code

```
 1. public class Printer {
 2.     public static void main(String[] args) {
 3.        int x1 = 1;
 4.        int x2 = 2;
 5.        int y = 4;
 6.        switch (y) {
 7.        case x1:
 8.            System.out.println("one");
 9.            break;
10.        case x2:
11.            System.out.println("two");
12.            break;
13.        default:
14.            System.out.println("unknown");
15.        }
16.     }
17. }
```

Which of the statement(s) is/are true?

A. The program prints "one"

B. The program prints "two"

C. The program prints "unknown"

D. Compile error

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

Given the following code:

```
interface Printable {}
class Document{}
class EBook extends Document implements Printable {}
```

Which of the following statements will not cause a compile error?

A. Printable printable1 = new Printable();

B. Printable printable2 = new Document();

C. Printable printable3 = new EBook();

D. Document document1 = new Document();

E. Document document2 = new EBook();

F. EBook ebook1 = new EBook();

G. EBook ebook2 = new Document();

32 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 test;
public class TasteOfMain {
    public static void main(String[] args) {
        System.out.println(args.length);
    }
}
```

The class is invoked with this command line command

```
java test.TasteOfMain \ \\\ \\
```

What is the value of **args.length?**

A. 1

B. 2

C. 3

D. 4

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

Given

```
 1. class Resident {
 2.     public void live() {
 3.     }
 4. }
 5.
 6. public class Printer {
 7.     public static void main(String[] args) {
 8.         Resident r1 = new Resident();
 9.         Resident r2 = new Resident();
10.         r1 = null;
11.         System.out.println(r1);
12.         r2 = r1;
13.         System.out.println(r2);
14.     }
15. }
```

Which of the following statements are true?

A. Two objects will be garbage collected at line 14.

B. One object will be garbage collected at line 12.

C. The object referenced by **r1** is eligible for garbage collection at line 11.

D. The object referenced by **r2** is eligible for garbage collection at line 13.

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

Given

```
package test;
public class StatementComposer {

    public String getStatement(int id, String name) {
        StringBuilder s = new StringBuilder();
        s.append("SELECT * FROM employees WHERE ");
```

```
            s.append("id = " + id);
            s.append(" OR ");
            if (name == null) {
                s.delete(s.length() - 4, s.length());
            } else {
                s.append("name = '" + name + "'");
            }
            return s.toString();
        }
        public static void main(String[] args) {
            StatementComposer sc = new StatementComposer();
            System.out.println(sc.getStatement(3, null));
        }
    }
```

What is the output of the class?

A. SELECT * FROM employees WHERE id = 3

B. SELECT * FROM employees WHERE id =

C. SELECT * FROM employees WHERE id = 3 OR name = 'null'

D. SELECT * FROM employees WHERE id = 3 OR name = null

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

What is the output of this program?

```
package test;
public class StringTest {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Rocks";
        String s3 = "JavaRocks";
        System.out.print(s3 == s1 + s2);
        System.out.print(".");
        System.out.print(s3.equals(s1 + s2));
    }
}
```

A. false.false

B. false.true

C. true.true

D. true.false

36 Question (2.7. Creating and manipulating Strings) ?

Consider the following method:

```
public String getSubstring(String s) {
    int index1 = s.indexOf(".");
    int index2 = -1;
    if (index1 != -1) {
        index2 = s.indexOf(".", index1 + 1);
    }
    if (index2 == -1) {
        return s.substring(index1);
    } else {
        return s.substring(index1, index2);
    }
}
```

What is the method's return value if the string "Hello.World" is passed to it?

A. Hello

B. World

C. .World

D. HelloWorld

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

What is the value of **y** after this code is executed?

```
int i = 10;
int y = 50;
if (i + y < 70) {
    if (i + y < 60) {
        y = 500;
    } else {
        y = 5000;
    }
}
```

A. 50

B. 500

C. 5000

D. 10

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

Which statement(s) creates an **ArrayList** of **String**s with an initial capacity of 20? (Choose all that apply)

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

B. ArrayList<String> names = new ArrayList<>(20);

C. ArrayList<String> names = new ArrayList<String>();

D. ArrayList<String> names = new ArrayList<String>(20);

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

Here is a multidimensional array:

```
int[][] tableCells = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11}};
```

What is the value of **tableCells[1][3]**?

A. 3

B. 9

C. 8

D. 11

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

Which of the following are valid declarations of multidimensional arrays? (Choose all that apply)

A. int[][] tableCells = new int[0][0];

B. String names[][] = {};

C. String[] employees[] = {{"Henry Wong", "Hendrick Takada"}};

D. long diameters[] = null;

E. long wheels[][] = null;

F. Object[][] cars = new cars[][];

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

Consider the following class:

```
 1. package test;
 2. public class MultiDimArray {
 3.
 4.     public static void main(String[] args) {
 5.         int[][] stickers = new int[4][];
 6.         for (int i = 0; i < stickers.length; i++) {
 7.             stickers[i] = new int[4];
 8.         }
 9.         System.out.println(stickers[0][0]);
10.     }
11.}
```

What happens if you try to compile and run this class?

A. Compile error at line 5

B. Compile error at line 7

C. It compiles and prints 0

D. It compiles and throws a runtime exception

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

Consider the following code.

```
package test;
public class Repeater {
    public static void main(String[] args) {
        if (args.length == 0) return;
        int input = 0;
        try {
            input = Integer.parseInt(args[0]);
            do {
                System.out.print(input + " ");
                input += 5;
            } while(input < 5);
        } catch (NumberFormatException e) {
            System.out.println("Not an integer");
        }
    }
}
```

What does the class print if it is invoked using this command line command?

```
java test.Repeater 50 20
```

A. Nothing

B. 50 55 60 65 70

C. 5

D. 50

43 Question (5.4. Compare loop constructs) ?

Consider the following while loop:

```
1. int x = 0;
2. while (true) {
3.     x++;
4.     if (x > 5) break;
5.     System.out.println(x);
6. }
```

Which of the following **for** statement can be used to replace line 2 without changing what the code does?

A. for (int x = 0; x < 5; x++) {

B. for (; x < 5;) {

C. for (; ;) {

D. for (int y = 0; y < 5; y++) {

44 Question (5.5. Use break and continue) ?

Consider this **for** loop:

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) continue;
    if (i % 5 == 0) break;
    System.out.print(i + " ");
}
```

What is printed on the console if the code snippet is executed?

A. 1 2 3 4 5 6 7 8 9

B. 1 3 5 7 9

C. 1 3 5

D. 1 3

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

Which of the following statements are true about encapsulation?

A. Instance variables are made private

B. For each instance variable, there is a get method that returns the instance variable. This method is public

C. For each instance variable that is writable, there is a set method that is public.

D. Class fields are made protected.

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

Consider the following code:

```
package test;
public class MathGenius {
    public void printDouble(int a, int b) {
        a *= 2;
        b *= 2;
        System.out.print(a + "__");
        System.out.print(b);
    }
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        new MathGenius().printDouble(x, y);
        System.out.print("__" + x + "__" + y);
    }
}
```

What is the output of the program?

A. 20__40__20__40

B. 20__40__10__20

C. 10__20__10__20

D. 10__20__20__40

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

Consider the following code:

```
package test;
class Point {
    int x, y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void setX(int x) {
        this.x = x;
    }
    public void setY(int y) {
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
    public String toString() {
        return x + "__" + y;
    }
}

public class PointDemo {
```

```
    public void enlargeAndPrintPoint(Point point) {
        point.setX(point.getX() * 2);
        point.setY(point.getY() * 2);
        System.out.print(point);
    }
    public static void main(String[] args) {
        Point point = new Point(100, 200);
        new PointDemo().enlargeAndPrintPoint(point);;
        System.out.print("__" + point);
    }
}
```

What is the output of the program?

A. 200__400__200__400

B. 200__400__100__200

C. 100__200__100__200

D. 100__200__200__400

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

Given

```
package temporary;
class Animal {
    public void walk() {
        System.out.print("Animal.walk()");
    }
}
class Giraffe extends Animal {
    public int getLegCount() {
        return 4;
    }
}
public class Main {
    public static void main(String[] args) {
        Animal animal = new Giraffe();
        // some code
        int legCount = animal.getLegCount();
    }
}
```

Which of the following statements are true?

A. This is polymorphism in action.

B. The code will compile

C. The code will not compile because the **getLegCount** method is not part of **Animal**.

D. The code will compile but will thrown a **NullPointerException**

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

Consider the following code:

```
 1. package test;
 2. class Request {
 3.     public void parse(String request) {
 4.         System.out.print("Request.parse()");
 5.     }
 6. }
 7. class HttpRequest extends Request {
 8.     public String getHttpMethod() {
 9.         return "post";
10.     }
11. }
12. public class Server {
13.     public static void main(String[] args) {
14.         Request request1 = new Request();
15.         HttpRequest httpRequest1 = (HttpRequest) request1;
16.         HttpRequest httpRequest2 = new HttpRequest();
17.         httpRequest2.parse("");
18.     }
19. }
```

Which of the following methods are untrue?

A. Compile error at line 15

B. Compile error at line 17

C. No compile error, but there will be a runtime error if the class is executed

D. No compile error and no runtime error if the class is execute

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

Which of the following statements are true?

A. A checked exception is internal to the program

B. Errors should not be caught

C. A runtime exception is an unchecked exception

D. A runtime exception is a checked exception

E. Applications can recover from checked exceptions

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

Consider the following code:

```
package test;
public class VideoGame {
    public static void main(String[] args) {
        int playerCount = 1;
        try {
            playerCount = Integer.parseInt(args[0]);
            if (playerCount > 1) {
                System.out.println("Multiple players found.");
            } else {
                System.out.println("One player");
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Please determine the number of players");
        } catch (NumberFormatException e) {
            System.out.println("Unrecognized number of players");
        }
    }
}
```

What is the output of the program if invoked using **"java test.VideoGame 1.0"?**

A. A runtime exception occurs

B. The program prints "Multiple players found"

C. The program prints "One player"

D. The program prints "Unrecognized number of players"

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

Which of the following are not runtime exceptions?

A. ClassCastException

B. NullPointerException

C. ArrayIndexOutOfBoundsException

D. NumberFormatException

E. OutOfMemoryError

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

Which of the following are caused by external errors?

A. ClassCastException

B. OutOfMemoryError

C. RuntimeException

D. StackOverflowError

E. Throwable

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

Let class **Circle** extend class **Shape**. Given the following code

```
Circle circle = new Circle();
Shape shape = new Shape();
```

Which of the following expressions evaluate to true?

A. circle instanceof Shape

B. circle instanceof Object

C. shape instanceof Circle

D. (Shape) circle instanceof Circle

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

Consider the following two classes:

```
package inheritance;
class Parent {
    public void print() { … }
    String describe() { … }
    protected String[] copyNames(String[] sources) { … }
    private boolean isResident() { … }
}

class Sub extends Parent {
    public void doIt() {
        …
    }
}
```

Which methods can be used in the **Sub** class?

A. print, describe, doIt

B. print, describe, copyNames, isResident

C. print, describe, copyNames

D. print, describe, copyNames, toString

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

Consider the following code:

```
package test;
public class Holder {
    float length = 100F;
    static {
        width = 23.00F;
    }
    static {
        width = 25.00F;
    }
    static float width = 12.30F;
    public static void main(String[] args) {
        Holder h1 = new Holder();
        h1.length++;
        h1.width++;
        Holder h2 = new Holder();
        System.out.print(h2.width + ", ");
        System.out.print(h2.length);
    }
}
```

What is the output of the program?

A. 25.0, 100.0

B. 12.3, 100.0

C. 26.0, 100.0

D. 26.0, 110.0

E. 13.3, 100.0

F. 13.3, 110.0

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

What is the value of **b?**

```
byte b = 2 ^ 6;
```

A. 64

B. 8

C. 4

D. 2

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

Given the following code:

```
package test;
import java.util.ArrayList;
public class Roller {
    public static void main(String[] args) {
        ArrayList<String> collector = new ArrayList<String>();
        collector.add("Study");
        collector.add(Integer.toOctalString(Integer.MAX_VALUE));

        ArrayList<String> names = new ArrayList<>();
        names.addAll(collector);
        names.add("Study");
        names.add(null);
        System.out.println(names.size());
    }
}
```

What is printed on the console?

A. 0

B. 2

C. 3

D. 4

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

Consider the following code:

```
 1. package test;
 2. abstract class Base {
 3.     public Base() {}
 4.     public Base(String s) {}
 5. }
 6. final class Impl extends Base {
 7. }
 8. public class Tester {
 9.     public static void main(String[] args) {
10.        Base base1 = new Base();
11.        Base base2 = new Impl();
12.        new Impl().toString();
13.        ((Base) new Impl()).toString();
14.    }
15. }
```

Does the code compile?

A. It compiles

B. No, compile error at line 10

C. No, compile error at line 12

D. No, compile error at line 13

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

Consider the following class:

```
class Engineer {
    public double calculate(double a, double b) { return 0; }
    private double calculate(double[] numbers) { return 0; }
    private void demandPayRise() {}
    public static void demandPayRise(BigDecimal byHowMuch) {}
    public long thinkInNumbers(int... numbers) { return 0; }
    public long thinkInNumbers(int a, int b) {return 1; }
}
```

Which of the following statements are true?

A. calculate are method overloads

B. demandPayRise are method overloads

C. thinkInNumbers are method overloads

D. calculate are not method overloads because they don't have the same access modifiers.

E. demandPayRise are not method overloads because one is static and the other is not.

F. thinkInNumbers are not method overloads because it is ambiguous which method will be called when two **int**s are passed in.

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

Given

```
1. public class ArrayUtil2 {
2.     public static void main(String[] args) {
3.         char[] characters = { '1', '2', '3', '4'};
4.         int[] numbers = characters;
5.         for (char c : characters) {
6.             System.out.print(c);
7.         }
8.     }
9. }
```

A. The code prints 1234

B. Compile error at line 4

C. Compile error at line 5

D. Compile error at lines 4 and 5

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

Given

```
1. public class ArrayUtil2 {
2.     public static void main(String[] args) {
3.         Character[] characters = { '1', '2', '3', '4'};
4.         Object[] numbers = characters;
5.         for (char c : characters) {
6.             System.out.print(c);
7.         }
8.     }
9. }
```

Which of the following statement(s) is/are true?

A. The code prints 1234

B. Compile error at line 4

C. Compile error at line 5

D. Compile error at lines 4 and 5

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

Which of the following are primitive wrappers?

A. Boolean

B. Char

C. Character

D. Int

E. Integer

F. Double

G. String

H. float

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

Which of the following statements are true?

A. The **try** statement can be used to handle errors in your program

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.Throwable**

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

Consider the following code:

```
public class VideoGame {
    public static void execute() {}
    public static void exit() {}
    public static void main(String[] args) {
        switch (args[0]) {
        default:
            exit();
            break;
        case "1":
        case "2":
        case "3":
            execute();
            break;
        }
    }
}
```

Which of the following statements are true?

A. The program may throw an exception

B. The **execute** method will be called if the value of the first argument to the program is "1", "2" or "3".

C. There is a compile error because the default case must be the last case in a switch statement

D. Nothing will be invoked if the value of the first argument to the program is "1" or "2"

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

Given the following code:

```
1. package science;
2. public class MathGenius {
3.     public int add(int… numbers) {
4.         int result = 0;
5.         for (int n : numbers) {
6.             result += n;
```

```
 7.          }
 8.          return result;
 9.      }
10.      public static void main(String[] args) {
11.          MathGenius mathGenius = new MathGenius();
12.          // some code
13.          System.out.println(result);
14.      }
15. }
```

Which of the following statements may replace the code in line 12 so the program will print **10**. (Choose all that apply).

A. int result = mathGenius.add({1, 2, 3, 4});

B. int result = mathGenius.add(new int[] {1, 2, 3, 4});

C. int result = mathGenius.add(1, 2, 3, 4);

D. int result = mathGenius.add(4, 3, 2, 1);

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

Consider the following code:

```
public class Employee {
    protected String firstName;
    public int age;
    private String lastName;
    boolean member;
}
---------------------------------
import test.sense.Employee;
public class Manager extends Employee {
    public void configure() {
        this.age = 12;
        this.firstName = "Aurora";
        this.lastName = "Valentino";
        this.member = false;
    }
}
```

Assuming **Employee** and **Manager** are not in the same package, which of the following statements in the **Manager** class will generate a compile error?

A. this.age = 12;

B. this.firstName = "Aurora";

C. this.lastName = "Valentino";

D. this.member = false;

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

Examine the following code.

```
class Person {
    protected String firstName;
    public int age;
    private String lastName;
    boolean member;
}

class HR {
    public static void main(String[] args) {
        Person person = new Person();
        person.age = 30;
        person.firstName = "Brad";
        person.lastName = "Sumitomo";
        person.member = false;
    }
}
```

Assuming **Person** and **HR** are classes in the same package, which statements will not generate a compile error?

A. person.age = 30;

B. person.firstName = "Brad";

C. person.lastName = "Sumitomo";

D. person.member = false;

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

Consider the signature of the following method:

```
public int handle(int a, int b)
```

Which of these method signatures are the overloads of the method above?

A. public static int handle(int a, int b)

B. public void handle(int[] args)

C. private String handle(int x, int y)

D. public Object handle(long a, long b)

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

Consider these two interfaces:

```
interface Printable {
    float getCost();
}

interface Uploadable {
    double getCost();
}
```

Is it possible for a class to implement both interfaces?

A. Yes, a class can always implement any set of interfaces

B. Yes, and you must provide two implementations of getCost()

C. Yes, and you must only provide one implementation of getCost()

D. No, it is not possible. The class will not compile

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

Which of the following options are part of what makes Java platform-independent? (Choose all that apply)

A. Java programs are compiled into bytecode

B. Small programs are compiled into native code

C. The JVM is available in many platforms

D. The JVM can run Java bytecode

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

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

A. double w = Double.MAX_VALUE;

B. Integer s = new Integer(123);

C. Long t = new Integer(9);

D. long u = new Integer(99);

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

What is the value of s after the following line of code is executed?

```
int s = new Double(23.8).intValue();
```

A. 23.8

B. 23

C. 24

D. 0

74 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 snippet:

```
String pattern = …;
DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern);
LocalDate date = LocalDate.parse("24 04 2019", formatter);
```

Which date pattern can be used to parse "24 04 2019" to April 24, 2019? (Choose all that apply)

A. dd mm yyyy

B. dd MM yyyy

C. d M yyy

D. dd MMM yyyy

75 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)

Which of the following lines of code can be used to create a date representing yesterday?

A. LocalDate yesterday = LocalDate.now().minusDays(1);

B. LocalDate yesterday2 = LocalDate.now().plusDays(-1);

C. LocalDate yesterday3 = LocalDate.now().plus(1, TemporalUnit.DAYS);

D. LocalDate yesterday4 = LocalDate.now().minus(1, ChronoUnit.DAYS);

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

Which statements are true with regard to **java.util.function.Predicate?**

A. **Predicate** is a predefined functional interface

B. **Predicate** can take primitive arguments

C. **Predicate** can take an array argument

D. **Predicate** can take one or two arguments

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

Which of the following are valid statements?

A. Predicate<Integer> abc = x -> x == 0;

B. Predicate<Void> s = (x) -> 2 == 1;

C. Predicate<> w = (x) -> 2 == 1;

D. Predicate<Long> t = x -> {return true};

Answers

1 **A, B**.

The keyword **extends** can be used to create a child class from another class or a subinterface from another interface. C is incorrect because **extends** cannot be used to make a class implement an interface. For this, you use the keyword **implements**. Also in Java an interface cannot implement another interface, so D is incorrect.

2 **B, C**.

All Java classes, even abstract ones, need at least one constructor. If no constructor is declared explicitly in a class, the compiler will create a default constructor, which means C is correct. Therefore, the class above will compile, which means B is correct.

3 **B**.

All Java classes, even abstract ones, need at least one constructor, so A is incorrect. However, you cannot instantiate an abstract class. Therefore, the class will compile but cannot be instantiated, so B is correct and C is incorrect. In addition, **abstract** is a Java keyword, but **Abstract** is not, which means D is incorrect.

4 **A, B, D**.

You cannot import a class in the default package. However, a class in the default package can use other classes in the default package without importing them, so statement B is also true. Finally, a class in the default package can import other classes in non-default packages.

5 **A, B, C, D, E**.

A and D are correct because the underscore character can be used in an identifier. B, C and E are also correct because $ can be used in a Java identifier, even though it is normally only used in machine-generated code. F is invalid because **static** is a reserved keyword and cannot be used as a variable name.

6 **D**.

A is incorrect because the rule that says a Java class must reside in a file with the same name as the class only applies to public classes. The code above has no compile error and will compile, so D is correct and B and C are incorrect.

7 **C**.

Here is the signature of the **StringBuilder** class's **replace** method:

```
public StringBuilder replace(int start, int end, String substring)
```

It works by first removing the characters between the two **int** arguments, including the character at the start position but not including the character at the end position. Note that the positions specified are zero-based, which means position 0 refers to the first character. The method then inserts the specified substring at the start position.

sb.replace(1, 5, "Worst") removes the second, third, fourth and fifth characters (resulting "Bphone cameras!") and inserts the word "Worst" at position 1 (resulting "BWorstphone cameras!").

8 **A, B**.

Note that the update statement in the **for** loop is **i=i+2**, which means the loop will have five iterations with i equal to 10, 12, 14, 16 and 18.

A is correct because the expression (i == 10) returns true only if i equals 10 and therefore the **println** method will only be called when i equals 10.

B is correct because the % operator is the remainder operator that calculates the remainder of a division operation. 10 % 5 = 0, whereas 12 % 5 = 2, 14 % 5 = 4, 16 % 2 = 1, 18 % 5 = 3.

C is incorrect because = is the assignment operator and this will cause a compile error.

D is incorrect because i % 2 will not return 5 for any integer.

9 **D**.

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

10 **B**.

The multiplication sign has precedence over addition, so 2 + 3 * 6 = 20 because the multiplication (3 * 6) is evaluated first and its result is added to 2. On the other hand, parentheses have precedence over multiplication, so (2 + 3) * 6 is 30. The result of a + b is therefore 20 + 30 = 50.

11 **A, B, C, E**.

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

B is correct because **m** and **n** are visible in region T.

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

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

E is correct because **m** is in scope in regions S, T, U and V.

F is incorrect because only **n** is out of scope in region X. **m** is still visible in region X.

12 **B, D**.

Nothing prevents you from naming a protected static void method **main**. However to make a class executable, the **main** method must be public and static, return no value, and accept one argument that is a String array.

13 **B**.

Remember that 015 is an octal number that is equivalent to 13. Comparing an **int** with a **long** having the same value evaluates to true.

14 **A, D, F**.

Only **public, final** and **abstract** are valid modifiers for top level class declarations.

15 **A, B, C, D, E, F, G**.

All these modifiers can be used on methods. However, you cannot use **final** and **abstract** on the same method.

16 **A, F**.

Only public and abstract are valid modifiers for top level interfaces. The use of abstract is valid but redundant as all interfaces are implicitly abstract.

17 **D**.

Instantiating a child class will also instantiate all its parents. **super()** is automatically called from a constructor if the first line of code in the constructor is not **super()**.

18 **D**.

By default, a constructor (regardless of its signature) in a derived class will call the no-argument constructor of its parent, unless there is another **super** statement that calls a non-default constructor of its parent.

19 **B**.

By default, a constructor (regardless of its signature) in a derived class will call the no-argument constructor of its parent, unless there is another **super** statement that calls a non-default constructor of its parent.

20 **C**.

To use a static member of a type in another type, import static that member prefixed by the fully-qualified name of the type containing the static member.

21 **A**.

To use a static member of a type in another type, import static that member prefixed by the fully-qualified name of the type containing the static member.

22 **A, C**.

A and C are valid array declarations.

B is incorrect because **remainders** is a multidimensional array and expects a multidimensional array but is assigned a one-dimensional array.

D is incorrect because an **ArrayList** is not an array.

23 **D**.

A and B are incorrect because there is no accessible member of the **String** class that is named **size**.

C is incorrect because **String** does not have a field named **length**.

D is correct. The **length** method returns the number of characters in the **String**.

E is incorrect because **String** does not have a field called **chars**.

F is incorrect because even though there is a **chars** method in **String**, it does not return the number of characters in a **String**.

24 **D**.

You can only invoke a non-static method by first creating an instance of the class containing the method. In order for the class to compile, you can either change **doSomething** to static like this:

```
private static void doSomething() {
    System.out.println("Hello, World!");
}
```

Or, you can create an instance of **Demo100** and call the method:

```
Demo100 demo = new Demo100();
demo.doSomething();
```

You can access a private method from the same class, so there is no need to change the method's access modifier.

25 **D**.

Access modifiers can only be applied to class-level variables or fields. You will therefore get a compile error at line 4.

26 **C**.

The boolean expression (i < 11) is evaluated and the result is passed to **System.out.println**. Since i = 10, i < 11 evaluates to true.

27 **C, D**.

Primitives are passed to methods as values. Changes to the method arguments do not affect the passed-in primitives.

28 **B, C, D**.

The program creates an instance of **Printer** by calling its no-argument constructor. The **this** keyword is used to call the other constructor, passing the string "Default". As a result, the **name** field will be given a value.

29 **C**.

The value of **x** is 0 when the first **while** loop is evaluated. Since 0 is less than 5, the interpreter enters the second loop. In the second loop, it prints the value of **x** and increments it. As a result, 012 is printed and the code exits the second loop with the value of x equal to 3. Since the interpreter cannot reenter the second loop and **x** does not increment further, the program will run indefinitely.

30 **D**.

Each case in a **switch** statement must be given a constant to evaluate. As the first and second cases are given a variable, there will be compile errors at lines 7 and 10.

31 **C, D, E, F**.

A is invalid because you cannot instantiate an interface.

B is illegal because there is no is-a relationship between **Printable** and **Document**. Therefore, assigning an instance of **Document** to a reference variable of type **Printable** is not allowed.

C is correct because **EBook** is an implementation of **Printable**. You can therefore assign an **EBook** instance of a variable of type **Printable**.

D is correct because this is just creating an instance of a class and assign the instance to a reference variable of the same type.

E is also correct because **EBook** is a subclass of **Document. EBook** and **Document** have an is-a relationship: An **EBook** is a **Document**.

F is definitely correct because you are creating an instance of a class and assign it to a reference variable of the same type.

G is not allowed because a **Document** is not necessarily an **EBook**.

32 **A**.

There is only one argument passed to the program for the following reason. The backslash character (\) can be used to escape another character. The first backslash character escapes the space and the second escapes the third escape. The fourth backslash escapes another space and the fifth escapes the sixth backslash. The argument received is four characters long and consists of two spaces and two backslashes:

\ \

33 **C, D**.

An object is eligible for garbage collection if it is no longer referenced by any variable. Assigning null to an reference variable removes the reference to an object. Therefore, C and D are correct.

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.

34 **A**.

The **StringBuilder** object in the **getStatement** method is first given the value "SELECT * FROM employees WHERE ". Then, "id = 3" is added to it and the length of the **StringBuilder** becomes 40. Because **name** is null, the **if** statement evaluates to true and the following line of code is executed:

```
s.delete(36, 40);
```

The **delete** method removes the characters in a substring of the **StringBuilder**. The substring begins at the specified start and extends to the character at index (end - 1) or to the end of the sequence if the length is shorter than (end - 1). This line of code effectively removes the last four characters in the **StringBuilder**.

35 **B**.

Comparing string variables with == tests if the variables point to the same object. Comparing string variables with equals tests if the string objects have the same value. Most of the time, when you compare strings, you are interested in knowing whether or not the string values are equal.

36 **C**.

The **indexOf** method returns the zero-based position (meaning the first character is at position 0) of the first occurrence of a substring in a string. In this case, **"Hello.World".indexOf(".")** returns 5. Because there is only one dot in "Hello.World", the value of **index2** will be -1 and **"Hello.World".substring(5)** returns ".World".

37 **C**.

The code starts by setting **x** to 10 and **y** to 50. The boolean expression to the first if statement evaluates to true because 10 + 50 < 70. The expression to the second if evaluates to false, so the else block is executed, in effect setting y to 5000.

38 **B, D**.

By default, an **ArrayList** is created with an initial capacity of 10 elements. To reserve a space for 20 elements, pass 20 to the constructor. A

and C are incorrect because the resulting **ArrayList** will have an initial capacity of 10. B and D are correct.

39 **C**.

tableCells[1] refers to the second row, which contains {5, 6, 7, 8}. **tableCells[1][3]** refers to the fourth member of the row, which is 8.

40 **A, B, C, E**.

A is a valid declaration of a multidimensional array even though the dimensions are 0.

B is also valid and is assigned an empty array.

C is legal. Square brackets after the type and square brackets after the variable name make it multidimensional.

D is a legal declaration of a one-dimensional array. It is therefore not a correct answer.

E is also valid even though it is assigned null.

F is invalid as cars in new cars[][] is not a type.

41 **C**.

A is incorrect as you can declare an array that has zero dimension.

B is incorrect because stickers was initialized and given proper values.

C is correct as by elements of an array of primitives are assigned the default value of the type. In this case, since it is an int array, each of its elements is assigned 0.

D is incorrect as it does not throw an exception.

42 **D**.

The first argument (50) is converted to an **int** and assigned to **input**. The **do** block will be executed at least once, so it will print "50 " and increment **input** by 5. Next, the value of **input** is compared to 5. Since 50 is not less than 5, the **do** block is not executed again.

43 **B, C, D**.

A is incorrect because x is redefined in the for statement and it will cause a compile error.

B is correct because the for block will loop until x is equal to 5, which will make the loop iterates five times.

C is correct because for (;;) is the same as while(true)

D is correct and will cause the loop iterates five times.

44 **D**.

A **continue** statement in a **for** loop causes the code below the **continue** statement to be skipped. A **break** statement in a **for** loop causes the loop to terminate.

The first **if** statement tests if **i** is evenly divisible by 2. If it is, the **continue** statement is called and this causes the rest of the **for** block to be skipped. Consequently, if **i** equals 0, 2, 4, 6 or 8, **continue** will be called.

The second **if** statement tests if **i** is evenly divisible by 5, which it is when i = 5. When this happens, the **break** statement is called and the loop terminates. The **print** method is only called when i equals 1 and i equals 3.

45 **A, B, C**.

A, B and C are correct. D is incorrect because protected fields are accessible from child classes and classes in the same package.

46 **B**.

Primitive variables are passed to a method by value, which means a copy of the primitive is created. As such, the values of **x** and **y** do not change.

47 **A**.

Point is a reference type, which means if it is passed in to a method, it is passed by reference. In other words, the **Point** object created in the **main** method and the one referenced in the **enlargeAndPrintPoint** method are the same object. Changing it in **enlargeAndPrintPoint** affects the object referenced in the **main** method.

48 **C**.

The code will not compile because the **Animal** class does not contain **getLegCount**. To get around this, cast **animal** to **Giraffe**, like so:

```
int legCount = ((Giraffe) animal).getLegCount();
```

49 **A, B, D**.

Note that the question wants you to point out which statements are false. A is false because there is no compile error at line 15. It is clear that request1 is of type **Request** and cannot be cast to **HttpRequest**. However, the compiler does not know that, so it does not generate a compile error.

B is false because there is no compile error at line 17. It is clear that **httpRequest2** is of type **HttpRequest** and **parse** is a public method defined in a parent class.

C is true because line 15 will throw a runtime error. You cannot cast a parent class to a subclass. Remember, an **Animal** is not necessarily a **Lion**, so you cannot cast an **Animal** to **Lion**. For the same reason, you cannot cast a **Request** to **HttpRequest**. However, you are not looking for true statements, so this option is not included in the answer.

D is false because there is a runtime error when the code is executed.

50 **A, B, C, E**.

A checked exception is an exceptional condition that a well-written program should anticipate and try to recover from. All exceptions are checked exceptions except **Error, RuntimeException**, and those derived from either of the two classes.

An error is an exceptional condition that is external to the program that it is unable to anticipate or recover from. Errors are represented by **java.lang.Error** and its subclasses.

A runtime exception is an exceptional condition that is internal to the program, and that the program usually cannot anticipate or recover from. A runtime exception is normally caused by a programming bugs. Runtime exceptions are unchecked exceptions.

51 **D**.

The argument "1.0" cannot be parsed to an int. As such, a **NumberFormatException** will be thrown.

52 **E**.

A, B, C and D are incorrect as **NullPointerException, ArrayIndexOutOfBoundsException** and **NumberFormatException** are derived from **RuntimeException**.

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

53 **B, D**.

A is incorrect because **ClassCastException** is a runtime exception and not an error.

B is correct because **OutOfMemoryError** is an error.

C is incorrect because **RuntimeException** is an exception.

D is correct because **StackOverflowError** is an error.

E is incorrect because **Throwable** is the parent class of all exceptions and errors and is not an error.

54 **A, B, D**.

A is true because a subclass has an is-a relationship with its parents.

B is true as all classes are directly or indirectly derived from **java.lang.Object**.

C is false because Shape is the parent class of Circle and a shape is not necessarily a circle.

D is true. Upcasting a **Circle** to its parent does not obscure the fact that it is still of type **Circle**.

55 **A, C, D**.

A subclass has access to the public, protected and default methods of its parents.

A is correct because **print** and **describe** have public and default access, respectively, and **doIt** is defined in **Sub**.

B is incorrect because **isResident** is private so it cannot be accessed from a child class.

C is correct a subclass can access the public and protected methods of its parents. In addition, **Parent** and **Sub** are in the same package, so **Sub** can access default methods in **Parent**.

D is correct because **Parent** implicitly extends **java.lang.Object**, As such, **Parent** inherits methods from **java.lang.Object** including its **toString** method. All methods that **Parent** inherit from its parent are also inherited to **Sub**.

56 **E**.

First off, static blocks and static variable initialization are called once when the class is loaded in the order they appear in the code. As such, the value of **width** is 12.30F as it is the last in the order. Second of all, static members are shared by all instances. Therefore, **h1.width++** increments the same static variable shared by all instances.

57 **C**.

The bitwise exclusive OR operator (^) works by comparing bits in the two operands. It returns true if the bits are different. Otherwise, it returns false.

Here is the bitwise exclusive OR operation of 2 and 6 in bit level:

```
2: 0000 0010
6: 0000 0110
------------
   0000 0100 (4)
```

58 **D**.

An **ArrayList** allows duplicates and null values. Therefore, the size of **names** after adding **collector**, a **String** and a null is 4.

59 **B**.

A is incorrect because the code does not compile for the reason outlined below.

B is correct because you cannot instantiate an abstract class. Line 10 is therefore causes a compile error.

C is incorrect because you can create an object and call its method without first assigning the object to a reference variable.

D is incorrect because **Impl** can be instantiated and upcast to its parent class.

60 **A, B, C**.

A, B and C are true statements.

D is false because overloaded methods may have different access modifiers.

E is false because overloaded methods may be static or otherwise.

F is false because in the case of two arguments, the method that take two arguments will be called.

61 **B**.

Just because a **char** can be assigned to an **int** does not mean a **char** array can be assigned to an **int** array. Because a **char** array and an **int** array are different types, there will be a compile error at line 4.

62 **A**.

java.lang.Object is the root of all Java classes. Assigning an array of any reference type to an **Object** array variable is allowed. However, you cannot assign an array of a primitive type to an **Object** array.

63 **A, C, E, F**.

The wrapper classes are **Byte, Short, Integer, Long, Float, Double, Character** and **Boolean**. A, C, E and F are therefore correct.

B is incorrect because there is no type **Char**. The wrapper class for **char** is **Character**.

D is incorrect because there is no type **Int**. The wrapper class for **int** is **Integer**.

G is incorrect because even though **String** is a reference type, it isn't a primitive wrapper class.

H is incorrect because float is a primitive, not a wrapper class.

64 **A, B, D**.

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

65 **A, B**.

A is true because if the program is executed without an argument, accessing the first argument to the program using the expression **args[0]** will throw an **ArrayIndexOutOfBoundsException**.

B is also true because there is no **break** statement after cases labeled "1" and "2". As such, in either case, control will fall through the case labeled "3" and the **execute** method will be called.

C is false because the default case may appear anywhere in the **switch** block.

D is false because if the value of the first argument to the program is "1" or "2", the case labeled "3" will be invoked.

66 **B, C**.

First note that the varargs used as the argument to the **add** method is equivalent to an **int** array. A is false because even though { 1, 2, 3, 4 } can be used to create an int array, it cannot by itself be used as a method argument. If you want to use this format, you could use these two statements to replace line 12:

```
int[] input = {1, 2, 3, 4};
int result = mathGenius.add(input);
```

B is correct. To create an array and use it as a method argument without first assigning the array to a variable, you can use this syntax:

```
new type[] { … }
```

C is correct because you are passing a series of **int**s, which is expected for a varargs.

D is correct because you are passing a series of **int**s, which is expected for a varargs.

67 **C, D**.

A is valid because **age** is a public field and can be accessed from anywhere.

B is also correct because **firstName** is a protected field and can therefore be accessed from classes in the same package as **Employee** as well as from classes that extend **Employee**.

C is invalid because **lastName** is private and cannot be accessed from outside its own class.

D is invalid because **member** has the default access and thus can be accessed by classes in the same package as **Employee**. Since **Employee** and **Manager** are not in the same package, **member** is not visible from **Manager**.

68 **A, B, D**.

Note that the question is which statements will *not* generate a compile error.

A is valid because **age** is a public field and can be accessed from anywhere.

B is also correct because **firstName** is a protected field and can therefore be accessed from classes in the same package as **Person** as well as from classes that extend **Person**.

C is invalid because **lastName** is private and cannot be accessed from outside the **Person** class.

D is correct because member has the default access and thus can be accessed by classes in the same package as **Person**.

69 **B, D**.

A is incorrect because simply making a method static does not create an overload. B is correct because it has a different set of arguments. C is incorrect because it has the same set of arguments. D is also correct because it has a different set of arguments.

70 **D**.

A class implementing both interfaces would have to provide the implementations of both **getCost** methods. However, the compiler would consider these implementations duplicate methods. If it were allowed and you made a call to **getCost** without assigning its return value to a variable, it would not be clear which **getCost** method should be invoked.

71 **A, C, D**.

Unlike the conventional programming paradigm, Java programs are compiled into bytecode. This bytecode cannot be executed directly but needs a Java Virtual Machine to run. By providing a JVM in a supported platform, the same bytecode can run in multiple platforms. Currently the JVM is available in Windows, Linux, Unix, Mac OS X, and others.

B is incorrect because even small programs are compiled into bytecode.

72 **A, B, D**.

A is valid because the **Double** object will be unboxed and its value assigned to **w**.

B is valid because assigning an **Integer** to a variable of the same type is valid.

C is invalid because **Long** and **Integer** are different classes and no unboxing will be attempted.

D is valid because the **Integer** will be unboxed and its int value assigned to **u**, which is a **long**.

73 **B**.

The **intValue** method of the **Double** class converts the internal double value to an **int**. No rounding will be performed, therefore the value of **s** will be 23.

74 **B, C**.

The letter "d" indicates the date and "m" the minute. As such, A is incorrect.

The letter M indicates the month. M and MM indicate the month in number, such as 01 or 1. MMM indicates the month in text, such as "Dec." Therefore, the pattern in D cannot be used to parse a month represented by a number.

B and C are both correct as they use d, M and y. "d" and "dd" are different. "d" allows one and two digit dates and "dd" only allows two digit dates. There is also a difference between M and MM. M accepts one or two digit months. MM only allows two-digit months.

For the year part, the following rules apply which can be found in the documentation for **java.text.SimpleDateFormat**.

For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "08/21/12" parses to August 21, 12 A.D.

75 **A, B, D**.

A is correct because **LocalDate.now()** returns a date representing today and **minusDays** decrements the date.

B is also correct because the **plusDays(-1)** decrements the date by one day.

C is incorrect because **TemporalUnitlDAYS** does not exist.

D is correct because the **minus** method can be used to decrement any component of the date by passing a number and a **TemporalUnit. ChronoUnit** implements **TemporalUnit** and therefore an instance of **ChronoUnit** can be passed to the **minus** method.

76 **A, C**.

A is correct because **Predicate** is one of the predefined functional interfaces in **java.util.function**.

B is incorrect because **Predicate** cannot take a primitive argument.

C is correct because **Predicate** can take any reference type, including an array.

D is incorrect because **Predicate** can only take one argument. You can use **BiPredicate** if you need a predicate that takes two arguments.

77 **A, B**.

A is correct as **Predicate** can take an **Integer**.

B is correct as **Void** can be used as an argument to **Predicate**.

C is incorrect because **Predicate** must take an argument.

D is incorrect because braces in a lambda expression must contain statements. "return true" is not a statement as it is missing a semicolon.