## Question 1

Which of the following are illegal initializations?
(Choose two options).

- A. int [ ] i = new int [2] { 5, 10};
- B. int i [5] = { 1, 2, 3, 4, 5};
- C. int [ ] i [ ] = { { }, new int [ ] { } };
- D. int i [ ][ ] = { {1,2}, new int [2] };
- E. int i [ ] = { 1, 2, 3, 4 } ;

Explanation:
The correct answers are options A and B.

Options A and B are correct because they are illegal initialization examples.

Option A because when creating anonymous arrays, it is illegal to specify the size of the array. For example, this is wrong; "new int[2]{1,2}", it should be corrected as "new int[]{1,2}".
Option B is because when declaring an array it is illegal to specify the size of the array as "int [2]i".

Options C, D, and E are as they are legal.

## Question 2

What will be the output of this code fragment?

```
10. int[] e[] = {{1, 2, 4}, {10, 11, 12, 13}};
11. int[] a = {1};
12. int[] b = {1, 2, 9};
13. int[] c = {1, 2, 6, 4, 5};
14. e = new int[][]{a, b, c};
15. System.out.println(e[1][2]);
```

- A. 4
- B. 0
- C. 9
- D. 6

Explanation:
The correct answer is option C.

Here at line 10, we have first created a two-dimensional array. Then we have created three one-dimensional arrays. Then we have assigned the anonymous two-dimensional array which contains three one-dimensional arrays to reference “e”. So, now the “e” refers to the two-dimensional array which contains, three one-dimensional arrays, a, b and c.

Option C is correct as the “e[1]” refers to “b” and “b[2]” refers to 9.

Other options are as explained above.

## Question 3

What will be the output of this code fragment?

```
1. int arrSize = 100;
2. String[][] sampleArray = new String[arrSize][];
3. sampleArray[0][0]="1";
4. System.out.println(sampleArray[0][0]);
```

- A. Compilation error at line 2.
- B. Null Pointer exception at line 3.
- C. 0
- D. Compilation error at line 3.

Explanation:
The correct answer is option B.

Option B is correct because sample Array is a two-dimensional array wherein the first dimension specifies the no. of arrays that can be held and the second dimension specifies the elements of the array. In this case, the first dimension details have been provided via arr Size but the second dimension has not been provided yet and we are trying to access the 0th element of the first array.

Hence, the exception occurs. If it is changes to String[][] sample Array = new String[arrSize][10]; then the code will compile & run fine.

Options A, C, and D are because of option B being correct.

## Question 4

Which of the following is valid for this given code snippet?

```
1. float f[][][] = new float[3][][];
2. float f0 = 1.0f;
3. float[][] fltarray = new float[1][1];
```

- A. f[0] = f0;
- B. f[0] = fltarray;
- C. f[0] = fltarray[0];
- D. f[0] = fltarray[0][0];

Explanation:
The correct answer is option B.

Option B is correct because f[0] would represent a two-dimensional array reference, so it can hold the fltarray as it is a two-dimensional array.

Other Options are as they try to assign elements which are not a two-dimensional array, for example in option A it f0 is just a variable.

Options A, C, and D are because of option B being correct.

## Question 5

Which of the following are the advantages an ArrayList have over arrays?

(Choose two options)

- A. Can store sequential data.
- B. It can grow dynamically
- C. It provides more powerful insertion and search mechanisms than arrays.
- D. Duplicate data is avoided.
- E. It has a key value pairs.

Explanation:
The correct answers are options B and C.

Options B and C are correct because ArrayList can grow dynamically & It provides more powerful insertion and search mechanisms than arrays.

Option A is because ArrayList can store sequential as well as non-sequential data.

Option D is  because duplicate data can be added.

Option E is  because key value pairs are stored inside a Map.

## Question 6

Which of the following is the correct way of declaring an ArrayList?

- A. List myList = ArrayList();
- B. List myList = ArrayList().newInstance();
- C. List myList = new ArrayList();
- D. List myList = ArrayList;

Explanation:

The correct answer is option C.

Option C is correct because List myList = new ArrayList() is a legal declaration of an ArrayList.

Option A is  as there the keyword new is not used, Options B and D are  as they are an invalid syntax.

## Question 7

Which of the following is the correct way of adding data inside an ArrayList?

List test = new ArrayList();

- A. test.add("string");
- B. test.put("string");
- C. test.set("string");
- D. test.add[0];

Explanation:
The correct answer is option A.

Option A is correct as test.add("string") is the legal declaration of adding data inside a list.

Options C, B, and D are  because they are illegal declarations of adding data inside a list.

## Question 8

Which of the following is the correct way of checking the size of an ArrayList?

List test = new ArrayList();

- A. test.length();
- B. test.size();
- C. test.size;
- D. test.length;

Explanation:
The correct answer is option B.

Option B is correct as test.size(); is the legal declaration of finding the number of elements inside a list.

Options A, C, and D are  because they are illegal declarations of finding the number of elements inside a list.

## Question 9

What will be the output of this code fragment?

1. int[] arr1;

2. System.out.println("arr1: " + arr1);

- A. arr: 0
- B. null
- C. compilation error
- D. arr:

Explanation:
The correct answer is option C.

Option C is correct because object references declared as members are initialized to null but object references declared in methods (local variables) are not initialized to default values. So, trying to use a variable without initializing causes a compile-time error. Hence, other options are  .

## Question 10

What will be the output of this program code?

```
1. int arrSize = 100;
2. String[] myArray = new String[arrSize];
3. myArray[0]="1";
4. System.out.println(myArray[0]);
```

- A. compilation error at line 2.
- B. 1
- C. 0
- D. compilation error at line 3.

Explanation:
The correct answer is option B.

Option B is correct because myArray[0]="1"; sets the value of the first element as 1 and eventually myArray[0] will display the same value.

Options A and D are  because there is no compilation error.

Option C is  because of option B being correct.

## Question 11

Which of the following is an illegal initialization?

- A. String[ ] oneDimArray = { "abc","def","xyz" };
- B. String s [ ] = ( "abcd", "defg", "hijk");
- C. int[ ] arr = new int[ ] {1,2,3};
- D. Integer[ ] oneDimArray1 = { 1,2,3 };

Explanation:
The correct answer is option B.

Option B is correct because curly braces { } can only be used in array declaration statements.

Options A, C, and D are  because they are legal array declarations.

## Question 12

What will be the output of this program code?

```
class App {
	public static void main(String[] args) {
		try {
			try {
			} finally {
				try {
					throw new IllegalStateException("ISE");
				} finally {
					throw new IllegalArgumentException("IAE");
				}
			}
		} finally {
			throw new RuntimeException("RE");
		}
	}
}
```

- A. Compilation fails
- B. Compilation succeeds but an IllegalStateException is thrown at runtime
- C. Compilation succeeds but an IllegalArgumentException is thrown at runtime
- D. Compilation succeeds but a RuntimeException is thrown at runtime
- E. The code compiles and runs without displaying any output

The correct answer is option D.

It is legal to throw exceptions from within the try, catch and finally block. However, the exception thrown from the outermost finally block will suppress all other exceptions. Hence, a RuntimeException will be thrown. In the absence of this throws statement, the IllegalArgumentException would have been thrown and if that too was missing, then the IllegalStateException would have been thrown.

## Question 13

What will be the output of this program code?

```
class App {
	public static void main(String[] args) {
		try {
			try {
			} finally {
				try {
					throw new IllegalStateException("ISE");
```

```
                    } finally {
                        throw new IllegalArgumentException("IAE");
                    }
                    throw new RuntimeException("RE");
                }
            } finally {
                System.out.println(“Inside Finally”);
            }
        }
}
```

- A. Compilation fails
- B. Compilation succeeds but an IllegalStateException is thrown at runtime
- C. Compilation succeeds but an IllegalArgumentException is thrown at runtime
- D. Compilation succeeds but a RuntimeException is thrown at runtime
- E. The code compiles and prints Inside Finally and then throws RuntimeException

The correct answer is option A.

Though it is legal to throw exceptions from within the try catch block, the line which throws the RuntimeException is unreachable. Hence, the compiler will throw an error. Removing that line will print Inside Finally followed by an IllegalArgumentException being thrown.

## Question 14

What will be the output of this program code?

```
57.  public class Fruit {
58.          public void eatMe() throws Exception {
59.                  System.out.print("Eat a fruit");
60.          }
61.          public static void main (String [] args) {
62.                  Fruit f = new Mango();
63.                  f.eatMe();
64.          }
65.  }
66.  class Mango extends Fruit {
67.          public void eatMe() {
68.                  System.out.println("Eating a mango");
69.          }
```

70. }

- A. Compilation fails due to an Error at line 3
- B. Compilation fails due to an Error at line 5
- C. Compilation fails due to an Error at line 7
- D. Compilation fails due to an Error at line 11
- E. Eat a fruit
- F. Eating a mango
- G. Compilation succeeds but an error is thrown at runtime

The correct answer is option C.

It is perfectly valid to override a method and throw no exceptions from the overriding method. However, since the main method is using a parent class reference to point to a child class object, the compiler will check for exceptions based on the method signature of the parent class. Since the eatMe method of Fruit throws Exception which is a checked exception, the compiler will force you to handle it. Hence, a compile-time error will be thrown at Line 7.

## Question 15

What will be the output of this program code?

```
78.  public class Fruit {
79.          public void eatMe() throws Exception {
80.                  System.out.print("Eat a fruit");
81.          }
82.          public static void main (String [] args) {
83.                  Mango m = new Mango();
84.                  m.eatMe();
85.          }
86.  }
87.  class Mango extends Fruit {
88.          public void eatMe() {
89.                  System.out.println("Eating a mango");
90.          }
91.  }
```

- A. Compilation fails due to an Error at line 3
- B. Compilation fails due to an Error at line 5
- C. Compilation fails due to an Error at line 7
- D. Compilation fails due to an Error at line 11
- E. Eat a fruit
- F. Eating a mango
- G. Compilation succeeds but an error is thrown at runtime

The correct answer is option F.

An overriding method can choose to not throw any exceptions that the parent class throws in the overridden method. If a reference of this same class is used while creating an object, the compiler will not force it to handle any exception as the method signature does not throw any exception. So, the code will compile successfully and the appropriate output will be printed.

## Question 16

What will be the output of this program code?

```
99. class Exception1 extends Exception { }
100.        class Exception2 extends Exception1 { }
101.        public class Fruit {
102.              public void eatMe() throws Exception2 {
103.                    System.out.print("Eat a fruit");
104.              }
105.              public static void main (String [] args) throws Exception2 {
106.                    Fruit f = new Mango();
107.                    f.eatMe();
108.              }
109.        }
110.        class Mango extends Fruit {
111.              public void eatMe() throws Exception1{
112.                    System.out.println("Eating a mango");
113.              }
114.        }
```

- A. Compilation fails due to an Error at line 5
- B. Compilation fails due to an Error at line 7
- C. Compilation fails due to an Error at line 9
- D. Compilation fails due to an Error at line 13
- E. Eat a fruit
- F. Eating a mango
- G. Compilation succeeds but an error is thrown at runtime

The correct answer is option D.

An overriding method can not be chosen to throw any exceptions that the overridden method had thrown. However, an overriding method cannot throw new or broader checked exceptions. Thus, the overriding method can throw the same exception or its, but not superclasses.

## Question 17

What will be the output of this program code?

```
122.    class Exception1 extends Exception {}
123.    public class Fruit {
124.        public void eatMe() throws Exception1 {
125.            System.out.print("Eat a fruit");
126.        }
127.        public static void main (String [] args) {
128.            Mango m = new Mango();
129.            m.eatMe();
130.        }
131.    }
132.    class Mango extends Fruit {
133.        public void eatMe() throws IllegalStateException {
134.            System.out.println("Eating a mango");
135.        }
136.    }
```

- A. Compilation fails due to an Error at line 4
- B. Compilation fails due to an Error at line 6
- C. Compilation fails due to an Error at line 8
- D. Compilation fails due to an Error at line 12
- E. Eat a fruit
- F. Eating a mango
- G. Compilation succeeds but an error is thrown at runtime

The correct answer is option F.

An overriding method can throw new runtime exceptions only. It can choose not to throw any exception that the overridden method throws. It cannot throw new or broader checked exceptions but there is no such restriction on unchecked exceptions. The throws class in the eatMe method of Fruit is not required but putting it there won't lead to any compilation error or runtime error. Moreover, since IllegalStateException is an unchecked exception (runtime exception), the main method will not be forced to handle it when it calls the eatMe method of Mango. However, if the main method was to call the eatMe method of Fruit, then it would have been forced to handle Exception1.

## Question 18

Which of the following option is correct about this statement?

**“Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.”**

- A. This is related to a checked exception.
- B. This is related to an error.
- C. This describes ParseException.
- D. This describes ArrayIndexOutOfBoundsException.
- E. None of the above.

Option E is the correct answer.

This statement describe the java.lang.IndexOutOfBoundsException and it is unchecked exception and the
java.lang.ArrayIndexOutOfBoundsException is its subclass.

## Question 19

Which of the following statements are true about assertions?

(Select 2 options)

- A. Assertions must not be used to validate arguments to a public method
- B. It is appropriate to use assertion for validating command line arguments
- C. Assertions can be used to check the cases that should not happen like the default case in a switch statement
- D. It’s a good practice to use assertions to cause side effects like calling a method if an assertion is true

The correct answers are options A and C.

Assertions should not be used to validate arguments to a public method but it can be used to validate arguments to a private method.

Option B is as it is not appropriate to use assertion for validating command line arguments, in this case, using the exception mechanism is suitable.

Similarly, assertions should not cause any side effects like a method call etc. because if assertions are turned off, then the method will never be called and the code might not function as required.

## Question 20

What will be the output of this program code?

```
import java.util.ArrayList;

public class Test {
        public static void main(String [ ] args) {
                ArrayList<String> strings = new ArrayList<String>();
                strings.add(“abc”);
                System.out.println(strings.get(1));
        }
}
```

- A. Compilation fails
- B. abc is printed
- C. An IndexOutOfBoundsException is thrown at runtime
- D. An ArrayIndexOutOfBoundsException is thrown at runtime
- E. Code compiles and runs with no output

The correct answer is option C.

The first element of a list is placed at index 0 and not 1. So, the correct way to access "abc" is strings.get(0). For an invalid index, an IndexOutOfBoundsException is thrown for a List.

ArrayIndexOutOfBoundsException is thrown when we try to access the elements of ArrayList with negative index and for LinkedList an IndexOutOfBoundsException is thrown for negative index.

## Question 21

Which of the following statement is true?

- A. Only checked exceptions can be used as an argument in a catch clause
- B. Only subclasses of java.lang.Error can be used as an argument in a catch clause
- C. Only subclasses of java.lang.Throwable or the Throwable class itself can be used as an argument in a catch clause
- D. Only runtime exceptions can be used as an argument in a catch clause

The correct answer is option C.

Any subclass of java.lang.Throwable can be used as an argument in a catch clause. Errors and Exceptions are direct subclasses of Throwable and hence, these subclasses, as well as their children, can be used in catch clause as an argument.

## Question 22

What will be the output of this program code?

```
1. public class Test {
2.       public static void m1() throws NewThrowable { }
3.       public static void main(String[] args) {
4.             m1();
5.       }
6. }
7. class NewThrowable extends Throwable {}
```

- A. Compilation fails at line 4 – Unhandled Exception type NewThrowable
- B. Compilation fails at line 7 – Cannot Extend Throwable
- C. Compilation succeeds and NewThrowable is thrown at runtime
- D. Compilation succeeds and the code runs without showing any output

The correct answer is option A.

Any class which is a subclass of Throwable but not a subclass of Error or RuntimeException is a checked exception. Hence, the code above is valid. However, it will compile correctly when NewThrowable thrown by method m1 is handled properly. The call to m1 on line 4 must be wrapped in a try and catch block or the main method must throw NewThrowable. Once these changes are made, the code will compile and run with no output.

## Question 23

Which of the following statements are true?
(Select three options)

- A. A throwable contains a snapshot of the execution stack of the thread that it was created in
- B. A throwable can contain a message string that gives more information about the error

- C. A throwable is created in a new Thread by the JVM that is separate from the thread in which the throwable is actually thrown
- D. A throwable can suppress other throwables from being propagated

The correct answer are options A, B, and D.

Option A is correct. One can get access to the execution stack through methods like printStackTrace or getStackTrace.

Option B is correct. This message can be retrieved through the method getMessage.

Option C is . The JVM does not create a new Thread for a throwable. The throwable is thrown in the same thread.

Option D is correct. Throwables can be suppressed in the case of try-with-resources blocks. The suppressed exception can be accessed through methods like addSuppressed and getSuppressed.

## Question 24

Which of the following statements are true?

(Select two options.)

- A. The finally block will execute even if the JVM exits while executing the corresponding try or catch block
- B. There can be only one finally block for single catch.
- C. The finally block can help prevent resource leaks if appropriate clean up code is written in finally block
- D. The finally block executes immediately after a try block finishes its execution

The correct answers are options B and C.

Option A is because the finally block will not be executed if the JVM exits during try or catch block execution.

Options B and C are correct.

Option D is because if a catch block is present after a try block and if an exception occurs, then the finally block will be executed after the corresponding catch block.

## Question 25

What will be the output of this program code?

```
public class Fruit {
        public void eat() {
                eatAgain();
        }
        public void eatAgain() {
                eat();
        }

       public static void main(String[] args) {
               new Fruit().eat();
       }
}
```

- A. The code will not compile.
- B. The code will compile but an IllegalAccessException will be thrown at runtime.
- C. The code will compile and run with no output
- D. A StackOverflowError will be thrown at runtime

The correct answer is option D.

An infinite recursive method call will not result in an exception; it will result in an error. Since the methods will be kept pushing over the stack without any elements coming out of the stack, a StackOverflowError will be thrown at runtime. The code will compile successfully.

## Question 26

What will be the output of this program code?

```
public class Fruit {
        public Fruit() {
                this("Mango");
        }
        public Fruit(String name) {
                this();
        }

        public static void main(String[] args) {
                Fruit f1 = new Fruit();
```

```
        }
}
```

- A. The code will not compile.
- B. The code will compile but an IllegalAccessException will be thrown at runtime.
- C. The code will compile and run with no output
- D. A StackOverflowError will be thrown at runtime

The correct answer is option A.

In the case of methods, an infinite recursive call will throw a StackOverflowError, but in the case of constructors, the compiler can determine that the object can never be instantiated if there is a recursive call amongst constructors. Hence, a compilation error will be encountered. It can be resolved, by removing this() from one of the constructors and replacing it with a super constructor call.

## Question 27

What will be the output of this program code?

```
public class Fruit {
    static {
        int x = 10/0;
    }

    public static void main(String[ ] args) {
        Fruit f1 = new Fruit();
    }
}
```

- A. The code will not compile.
- B. The code will compile but an ArithmeticException will be thrown at runtime.
- C. The code will compile but an ExceptionInInitializerError will be thrown at runtime.
- D. The code will compile and run with no output.

The correct answer is option C.

When a static initializer block does not complete, the class cannot be loaded and an ExceptionInInitializerError is thrown. The cause of this error will be ArithmeticException.

## Question 28

What will be the output of this program code?

```
public class Fruit {
    Fruit() {
        int x = 10/0;
    }

    public static void main(String[] args) {
        Fruit f1 = new Fruit();
    }
}
```

- A. The code will not compile.
- B. The code will compile but an ArithmeticException will be thrown at runtime.
- C. The code will compile but an ExceptionInInitializerError will be thrown at runtime.
- D. The code will compile and run with no output.

The correct answer is option B.

When a static initializer block does not complete, the class cannot be loaded and an ExceptionInInitializerError is thrown. However, when a constructor throws an exception, it is propagated to the stack. So, this code will throw an ArithmeticException.

## Question 29

What will be the output of this program code?

```
public class Alpha {
    public static String greet() {
        return “Hello World”;
    }

    public static void main (String [] args) {
        System.out.println(greet());
    }
}
```

- A. The program will compile and run successfully and will print Hello World
- B. The program will not compile
- C. The program will compile but will throw an error at run-time
- D. The program will compile and run with no output

The correct answer is option A.

The main method is static and can access other static members of the class including static fields and methods. Hence, the program will compile and run fine to print the output Hello World. Thus, option A is the correct answer.

## Question 30

Which of the following statements are true?
(Select 3 options).

- A. A constructor may have same name as the class.
- B. Constructors can be overloaded.
- C. Instance members are accessible only after the super-constructor runs.
- D. A class must have at least one constructor.
- E. Constructors can not be used for modifying static variables.
- F. Constructors may have a return type.

Explanation:
Options B, C, and D are the correct answers.

Constructors never inherited so it can't be overridden but constructors can be overloaded, so option B is correct.

Option C is correct as the instance members are initialized after the super constructor completes.

Option D is correct as every class including abstract classes must have at least one constructor but interfaces haven't.

Option A is as the constructor name must be same as the class name.

Option E is since it is legal to modify static content using constructors but it is not s good practice.

Option F is as a constructor hasn't got a return type.

## Question 31

What will be the output of this code fragment?

```
import static java.lang.System.*; //line 1

public class Pro {

        public static void main(String[] args){
        out.print("A"); //line 5
        }

}
```

- A. A
- B. Compilation fails due to an error at line 1.
- C. Compilation fails due to an error at line 5.
- D. Compilation fails due to multiple errors.

Option A is correct.

Option A is correct as the “A” will produce as the output.

Options B and C are as we can use a static keyword for importing only the static member of a class; in this case, we have import java.lang.Systems class all static members. So, we don’t have to use “System.out” for printing.

Option D is as there is no compile-time error while compiling this code.

## Question 32

Which of the following can be instantiated?

- A. Abstract classes
- B. Interfaces
- C. Concrete classes
- D. Primitive types

The option C is the correct answer.

Concrete classes are the classes that can be instantiated and they cannot contain any

abstract methods. They also provide implementations for any abstract methods in their superclass or interfaces implemented by them. Abstract classes are class marked as abstract and cannot be instantiated, so choice A is . They may or may not contain abstract methods.

Choice B is because interfaces contain only abstract methods and cannot be instantiated.

Only classes can be instantiated but primitive types cannot be instantiated. So, choice D is also .

## Question 33

A Person may use zero to many Cars. A Car belongs to a single Person.

Which of the following code fragment correctly shows this relationship?

- A. class Person { Car c[ ]; } class Car { Person p; }
- B. class Person { Car c; } class Car { Person[ ] p; }
- C. class Person { Vector c; } class Car { Person[ ] p; }
- D. class Person { Car c; } class Car { Person p; }

The correct answer is option A.

A Person may use

zero to many Cars, so the Person class will contain an object reference
an instance variable, which refers to a Car object array.

Hence, options B, C, and D are automatically .

## Question 34

A private member of class Alpha can be accessed by class Beta if Beta extends Alpha.

- A. True
- B. False

The private access scope in Java is the most restrictive access scope and private members can only be accessed within the same class. No other class including the sub class can access private members from outside the class. Only members of the same class including inner classes can access private members.

## Question 35

Which will be the first line to cause an error in this program code?

```
1. class Train
2. {
3.         public static void main(String arg[ ])
4.         {   boolean flag=false;
5.             while(true)
6.             {
7.                 flag=true;
8.             }
9.             while(!flag)
10.            {
11.            }
12.            do {
13.            } while(true);
14.            do
15.            {
16.               ;
17.            }
18.            while(false);
19.    }
20. }
```

- A. Line no. 9
- B. Line no. 14
- C. Line no. 16
- D. Line no. 5
- E. Line no.18

Explanation:
The correct answer is option A.

Option A is correct. At line 5, the while condition is set to true always. Hence, it is an infinite loop, so compiler sees that code after the while loop will never reach, so compiler gives an error due to unreachable statements.

Option D is due to compiler error shows at line number 9.

Option B is because it is allowed to provide false as a direct value to the while condition.

Option C is because there is no issue there.

## Question 36

What will be the output of this program code?

```
219.        public class test {
220.              public static void main(String args[]) {
221.                    int i = 1;
222.                    do {
223.                          i-- ;
224.                    } while ( i > 2);
225.                    System.out.println(i);
226.              }
227.        }
```

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

Explanation:

The correct answer is option A.

Option A is correct because since i is never greater than 2, the while loop does not execute but since its do while the iteration is done once Hence the value of i goes on to become 0.

Options B, C, and D are  because of option A being correct.

## Question 37

What will be the output of this program code?

```
11. public class Example {
12.       public static void main(String[] args) throws Exception {
13.
 14.             for (int a = 30;) {
                       15. System.out.print("value:" + a--);
16.             }
17.       }
18. }
```

- A. Compilation error
- B. 30 29,.....until 1
- C. 1 2 3.....until 30
- D. No output

Explanation:
The correct answer is option A.

Option A is correct because for loop needs to have semicolons separating the initialization, conditional check, and operation. But any of these or all these can be left blank but the semicolons should be there.
For example followings are legal ; for(int i=30;;), for(;;)."

Options B and C are  because the code does not compile, hence no output is produced.

Option D is  because the ' no output ' option even though correct is less specific as compared to option A.

## Question 38

What will be the output of this code fragment?

```
15. public class Sample {
16.         public static void main(String[] argv) {
17.                 for (int i;i<7;) {
18.                     i++;
19.                 }
20.                 System.out.println("i is " + i);
21.         }
22. }
```

- A. An exception is thrown
- B. Compilation error
- C. 8
- D. 0

Explanation:
The correct answer is option B.

Option B is correct because i variable is never given a value i.e. The local variable i may not have been initialized before being used, hence the compiler error.

Option A is  because the compilation phase fails.

Options C and D are  because the code does not run successfully, hence no output is produced.

## Question 39

What will be the output of this program code?
(Select 2 options).

```
240.    public class Sample {
241.        public static void main(String [] args) {
242.            int bang = 1;
243.            do while ( bang < 1 )
244.               System.out.print(" bang is " + bang);
245.            while ( bang >= 1 ) ;
246.        }
247.    }
```

- A. bang is 1
- B. bang is 1 bang is 1
- C. Compilation error
- D. No output is produced.
- E. It is an infinite loop.

Explanation:
The correct answers are options D and E.

Options D and E are correct because There are two different looping constructs in this problem. The first is a do-while loop and the second is a while loop, nested inside the do-while.
The body of the do-while is only a single statement—brackets are not needed.

You are assured that, while expression will be evaluated at least once, followed by an evaluation of the do-while expression. Although the inner while loop never evaluates to true since bang will never be less than 1 ( Hence, no output is produced ) but the outer do while loop's condition always evaluates to true since bang is equal to 1 (Hence, an infinite loop).

Options A, B, and C are based on the program logic described above.

## Question 40

What will be the output of this code fragment?

```
253.    public static void main(String[ ] args) {
254.        boolean flag = false;
255.        int x = 0;
256.        for (;flag=!flag;) {
257.            x++;
258.        }
259.        System.out.println(x);
260.    }
```

- A. 0

- B. 1
- C. 2
- D. 3
- E. Compilation error
- F. The loop is infinite and will cause the program to break

Explanation:
The correct answer is option B.

Option B is correct because in the condition we are not only comparing the boolean values but also changing the value of the flag to true, in the first iteration.

Options A, C, and D are because of the option B being correct.

Option E is because compilation does not fail & the code runs successfully.

Option F is because the loop terminates successfully.

## Question 41

What will be the output of this code fragment?

```
11. public class Example {
12.         public static void main(String[] args) throws Exception {
13.                 int a = 30;
14.                 while (a) {
15.                         System.out.print("value:" + a--);
16.                 }
17.         }
18. }
```

- A. Compilation error
- B. 30 29,.....until 1
- C. 1 2 3.....uniltil 30
- D. No output

Explanation:
The correct answer is option A.

Option A is correct because 'While' statement should contain a 'Boolean' value, 'condition' or 'expression', whose outcome must be of type Boolean. Hence, the compilation error.

Options B and C are because the code does not compile, hence no output is produced.

Option D is  because the ' no output ' option even though correct is less specific as compared to option A.

## Question 42

What will be the output of this code fragment?

```
15. public class Sample {
16.         public static void main(String[] argv) {
17.                 while (int i < 7) {
18.                         i++;
19.                 }
20.                 System.out.println("i is " + i);
21.         }
22. }
```

- A. An exception is thrown
- B. Compilation error
- C. 8
- D. 0

Explanation:
The correct answer is option B.

Option B is correct because i variable is never given a value before being used, hence the compilation error.

Option A is  because the compilation phase fails.

Options C and D are  because the code does not run successfully, hence no output is produced.

## Question 43

Which of the following will produce output 12 when inserted at line 14?

```
11. class Sample4 {
12.         public static void main(String [] args) {
13.                 int x = 0;
14.                 // insert code here
15.                 while (x++ < y) { }
16.                 System.out.println(x);
```

```
17.         }
18. }
```

- A. int y = x;
- B. int y = 10;
- C. int y = 11;
- D. int y = 12;
- E. int y = 13;
- F. None of the above will allow compilation to succeed.

Explanation:

The correct answer is option C.

Option C is correct because x reaches the value of 11, at which point the while test fails. x is then incremented (after the comparison test!), and the println() method runs.

Options A, B, D, E, and F are based on the above.

## Question 44

What will be the output of this code fragment?

```
public class Pro {
        public static void main(String [] ar) {
                for( int x = 10; x-- > 5; System.out.print(x) );
        }
}
```

- A. 101112131415
- B. 987654
- C. 98765
- D. 1098765
- E. Compilation fails.

Option C is the correct answer.

Option C is correct since the output is 98765 when for loop begins the value of the variable x is 10, then the value is reduced by 1, and then for every iteration, this happens till x>5 after that loop stops.

Other options are as explained above.

## Question 45

What will be the output of this code fragment?

```
11. public static void main(String[ ] args) {
12.         Integer i = new Integer(2) + new Integer(2);
13.         switch(i) {
14.                 case 4: System.out.println(”foo”); break;
15.                 default: System.out.println(”other”); break;
16.         }
17. }
```

- A. foo
- B. other
- C. An exception is thrown at runtime.
- D. Compilation fails due to an error at line 12.
- E. Compilation fails due to an error at line 13.
- F. Compilation fails due to an error at line 15.

Explanation:
The correct answer is option A.

Option A is correct because case 4 is matched and eventually the break statement stops the flow of the switch case.

Option B is  because case 4 is matched. When a match has occurred & break is executed, then the default will not be executed.

option C is  because there is no exception at runtime, the program runs successfully.

Options D, E, and F are  because there is no compiler error, the program runs successfully.

## Question 46

What will be the output of this code fragment?

```
11. int i =2,j =1;
12. do {
```

```
13. if(i++> --j) {

        14. continue;
        15. }

    16. j-- ;
    17. } while (i <9);

18. System.out.println("i = " +i+ "and j = "+j);
```

- A. i = 9 and j = 9
- B. i = 9 and j = 6
- C. i = 9 and j = -6
- D. i = 6 and j = 6

Explanation:
The correct answer is option C.

Option C is correct because
1st iteration i=3 j=0 if condition true causing the flow to the next iteration
2nd iteration i=4 j=-1 if condition true causing the flow to the next iteration
3rd iteration i=5 j=-2 if condition true causing the flow to the next iteration
4th iteration i=6 j=-3 if condition true causing the flow to the next iteration
5th iteration i=7 j=-4 if condition true causing the flow to the next iteration
6th iteration i=8 j=-5 if condition true causing the flow to the next iteration
7th iteration i=9 j=-6 if condition true causing the flow to the next iteration

Now, while loop exists since i < 9 Hence, i=9 & j=-6

Options A, B, and D are because of the option C being correct.

## Question 47

What will be the output of this code fragment?

```
11. int i = 2, j =1;
12. do {
13.     if ( i++ > --j ) {
14.         break;
```

```
15.      }
16. } while (i <9);
17. System.out.println(“i = “ +i+ “and j = “+j);
```

- A. i = 0 and j = 3
- B. i = 3 and j = 0
- C. i = 9 and j=3
- D. i = 0 and j=6

Explanation:
The correct answer is option B.

Option B is correct because
1st iteration i=3 j=0 if condition true because of which break is executed & the looping exits.

Options A, C, and D are  because of the option B being correct.

## Question 48

You want to print 'Yes' if a character primitive contains 0 or 1 and 'No' if it contains 2 or more. Which of the following code fragment will correctly achieve this?

- A. if(c=='0' || c=='1')  System.out.println("Yes");  else if(c>='2')  System.out.println("No");
- B. if(c=='0' && c=='1')  System.out.println("Yes");  else if(c>'2')  System.out.println("No");
- C. if(c='0' || c='1')  System.out.println("Yes");  else if(c>='2')  System.out.println("No");
- D. if(c=="0" || c=="1")  System.out.println("Yes");  else if(c>="2")  System.out.println("No");

Explanation:

The option A is the correct answer.

We need to print 'Yes' if a character primitive contains 0 or 1 and 'No' if it contains 2 or more. For this, we compare the value of the character to '0' and '1' using the 'if' statement and the '==' operator.

Choice B is  because it uses the '&&' operator instead of the '||' operator, the value cannot be equal to both '0' and '1'.

Choice C is  because it uses the '=' operator instead of the '==' operator.

Choice D is  because it uses double quotes instead of single quotes for representing character literals.

## Question 49

Which of the following statement is true?

```
1. class Mammal {
2.       public int chromosomes;
3.       protected Mammal(int chromosomes) {
4.             this.chromosomes = chromosomes;
5.       }
6. }
7. public class Animal extends Mammal {
8.       private Animal(int chromosomes) {
9.             super(chromosomes);
10.      }
11.      public static void main(String[ ] args) {
12.            Animal ext = new Animal(420);
13.            System.out.print(ext.chromosomes );
14.      }
15. }
```

- A. 420 is the output.
- B. An exception is thrown at runtime.
- C. All constructors must be declared public.
- D. Constructors CANNOT use the private modifier.
- E. Constructors CANNOT use the protected modifier.

The correct answer is option A.

Option A is correct because this is another example of constructor chaining where the flow is as follows line.12 >line.8 > line.9 > line.3 > line.4 > line.9 > line.12 > line.13

Option B is  because there is no such code that would invoke the runtime exception.

## Question 50

Which of the followings will allow Star to compile?
(Select two options).

```
11. class Sky {
12.         private int s;
13.         protected Sky(int s) { this.s = s; }
14. }
.....
21. class Star extends Sky {
22.         public Star(int s) { super(s); }
23.         public Star() { this.s= 9; }
24. }
```

- A. Change line 12 to: public int s; ☐
- B. Remove line 12. ☑
- C. Change line 23 to: public Star() { this(9); } ☐
- D. Change line 23 to: public Star() { super(9); } ☑
- E. Change line 23 to: public Star() { super(s); } ☐

The correct answers are options C and D.

Options C and D are correct because line 23 conflicts the rule of the constructor of the super class being called before the constructor of the child class. Currently, Line 23 does not in any way calls the super constructor.

Hence, option C in a way calls the another child class constructor which has the call to the super class constructor and option D directly calls the super constructor.

Option A is even if we make line the variable s public, there is another mistake at line 23 due to the wrong constructor.

Option B is as removing the variable s will introduce more compile-time errors.

Option E is because it is an improper call to the superclass constructor and is syntactically wrong.

## Question 51

Consider the following code-

```
1. public class Maths {
2.         public int value;
3.         public void compute() { value += 7; }
```

```
4. }
```

And-

```
11. public class Statistics extends Maths {
12.         public void compute() { value -= 3; }
13.         public void compute(int multiplier) {
14.                 compute();
15.                 super.compute();
16.                 value *=multiplier;
17.         }
18.         public static void main(String[] args) {
19.                 Statistics calci = new Statistics();
20.                 calci.compute(2);
21.                 System.out.println("Value is: "+ calci.value);
22.         }
23. }
```

What will be the output?

- A. Value is: 8
- B. Compilation fails.
- C. Value is: 12
- D. Value is: -12
- E. The code runs with no output.
- F. An exception is thrown at runtime.

The correct answer is option A.

Option A is correct because line.20 > line.13 > line.14 > line.12 here Statistics compute() will be called value=value-3=0-3=-3 > line.14 > line.15 > line.3 here Maths compute() will be called value=value+7=-3+7=4 > line.15 >line.16 here value=multiplier * value=2 * 4= 8.
Hence, 8 is displayed.

Options B, E, and F are  because the code executes successfully with the output 8 displayed.

Options C and D are  because 8 is output rather than 12 or -12.

## Question 52

What will be the output of this code fragment?

```
320.    public class CastIt {
321.        public static void main(String[] args) {
322.            boolean t = true;
323.            byte b = 4;
324.            b = (byte)(t?1:0);
325.            int s=b;
326.            System.out.println("Value of s after conversion : " + s);
327.        }
328.    }
```

- A. Value of s after conversion : 4
- B. Value of s after conversion : 1
- C. Exception occurs at line : 6
- D. Value of s after conversion : 0

The correct answer is option B.

Option B is correct answer because we cannot convert boolean to byte but we can convert 1 or 0 to byte i.e. We explicitly typecast it to a byte; the further byte is implicitly typecasted to int printing the value 1.

Options A and D are because the original value of b which was 4 is changed due to the logical expression at line.5 hence further the changed value 1 is passed to s.
Option C is because there is no scope for any kind of exception getting created since the cast is proper.

## Question 53

Which of the following is not an eligible cast?

- A. short sVal = 45;
- B. short sVal = 234251434324324;
- C. short sVal = 38; int lVal = sVal;
- D. int lVal=16; short sVal = (short) lVal;
- E. Object temp = "avalue"; String str = (String)temp;

The correct answer is option B.

Option B is correct because this is not allowed, the right-hand side value i.e.

234251434324324 is too big for a short which is on the left-hand side.

Option A is  because it is syntactical right since it will allow if the right-hand side value i.e. 45 is small enough to fit in short.
Option C is  because it is syntactical right since assigning a short to an integer is an implicit cast and this the compiler can handle because ints are bigger than shorts.

Option D is  because it is syntactical right since assigning an integer to short requires an explicit cast because short is smaller than int although they would lead to a loss of precision.
Option E is  because it is syntactical right since casting is a method of assigning a value of one type to a variable of a more specific type, here is an example:

String extends Object, making "String" the more specific type (subclass). The (String) in parenthesis is how you perform the cast from Object to String. It basically assures the compiler that you, as the programmer, know that the actual value extends the expected type.

## Question 54

Which of the following is wrong in terms of casting?

- A. Type promotion casting i.e. implicit casting is an automatic type conversion from a “lesser” base type to a “greater” one or from any type to String via the toString() method.
- B. In explicit casting, it is possible to force the conversion where in a type cast, it may mean loss of precision, but the compiler will perform the conversion and will not generate an error.
- C. When casting an object to a non-parent type, a runtime check will be performed and if the cast fails, an exception will be raised.
- D. It is also possible to cast objects from one type to another. When casting an object to a parent type, the cast will always succeed and will never generate an error.
- E. There are chances of upcasting failing since it is not clear to the compiler to which datatype the cast will occur to.

The correct answer is option E.

Option E is correct because upcasting i.e. Upward casting never fails because a larger data holder can always hold the data of the small data holder.

Options A, B, C, and D are  because they are right as per the specifications of jdk.

## Question 55

Which of the following statement is wrong in terms of casting?

- A. The cast operator (type) is used to convert numeric values from one numeric type to another or to change an object reference to a compatible type
- B. It is mandatory to use explicit casting since the conversions will not be allowed by the compiler.
- C. A reference of any object can be cast to a reference of type Object
- D. A reference to an object can be cast into a reference of type ClassName if the actual class of the object, when it was created, is a subclass of ClassName
- E. A reference to an object can be cast into a reference of type InterfaceName if the class of the object implements Interface, if the object is a subinterface of InterfaceName or if the object is an array type and InterfaceName is the Cloneable interface

The correct answer is option B.

Option B is correct because It is not mandatory to use explicit casting since there are possibilities of an implicit cast can be applicable.

Options A, C, D, and E are as they are right as per the specifications of jdk.

## Question 56

What will be the output of this code snippet?

```
1. Parent p = new Parent(1.0, 2.0); //Parent( double x, double y)
2. Parent q = p;
3. System.out.print(p.x+" ");
4. q.x = 13.0;
5. System.out.print(p.x);
```

- A. 1.0 2.0
- B. 2.0 1.0
- C. 1.0 13.0
- D. 13.0 13.0
- E. 13.0 1.0

The correct answer is option C.

Option C is correct as after the code Parent p = new Parent(1.0, 2.0); runs, the variable q holds a copy of the reference held in the variable p. There is still only one copy of the Parent object in the VM, but there are now two copies of the reference to that object. Since the variables p and q hold references to the same object, either variable can be used to make changes to the object, and those changes are visible through the other variable as well.

Options A, B, D, and E are  because of the explanation stated above.

## Question 57

What will be the output of this code snippet?

```
1. char[] compass = { 'n','s','e','w' };
2. char[] pointer = compass;
3. pointer[3] = '@';
4. System.out.println(compass);
```

- A. nsew
- B. nse@
- C. nsew@
- D. ns@w

The correct answer is option B.

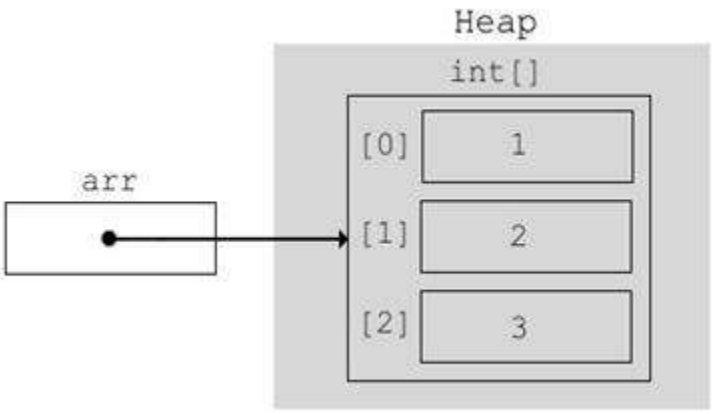
Option B is correct because from line 2 compass holds an array reference and pointer holds the same reference hence both compass and pointer are references which point to the same object { 'n','s','e','w' }.
So, when the 3rd index of the pointer is changed due to the use of a reference to change an element, the same is reflected in compass since both are pointing to the same object.

Options A, C, and D are  because of the stated explanation.

## Question 58

Which of the followings are correct on the basis of this image?

(Select four options)




- A. int[] arr = new int[]{1,2,3};
- B. arr is the reference name
- C. int[] is the object type
- D. individual contents of the int[] are primitives & the elements in the array store their actual value.
- E. Heap is the object type.
- F. arr is the object type.

The correct answers are options A, B, C, and D.

Option A, B, C and D are correct because from the image if int[] arr = new int[]{1,2,3}; hence, arr is the reference name and int[] is the object type wherein individual contents of the int[] are primitives and the elements in the array store their actual value.

Options E and F are because Heap is not an object type and arr is not an object type. Heap is a storage area.

## Question 59

Consider this code:

```
10. abstract class X {

        11. abstract void a1();
        12. void a2() { }

13. }
14. class Y extends X {
```

15. void a1() { }
16. void a2() { }

17. }
18. class Z extends Y { void c1() { } }

And this:

X x = new Y();
Y y = new Z();
X z = new Z();

Which of the followings are valid examples of polymorphic proper referential method calls?
(Choose four options)

- A. x.a2();
- B. z.a2();
- C. z.c1();
- D. z.a1();
- E. y.c1();
- F. x.a1();

The correct answers are options A, B, D, and F.

Options A, B, D, and F are correct because -
x.a2();
here x is a type of X and a2() is inside Y which is a subtype of X and both Y and X have a2() where a2() is overridden in Y.

z.a2();
here z is a type of X and a2() is inside Y which is a subtype of X and both Y and X have a2() where a2() is overridden in Y.
z.a1(); here z
is a type of X and a1() is inside Y which is a subtype of X and both Y and X have a2() where a2() is overridden in Y.
x.a1(); here x is a
type of X and a1() is inside Y which is a subtype of X and both Y and X have a2() where a2() is overridden in Y.

Option C is
because in z.c1(); where z is a type of X and c1() is inside Z which is a subtype of Y further Y is a subtype of X but c1() only exists in Z

neither in X nor Y hence making polymorphism less ideal.

Option E is  because in c1() is only exists in the class Z and we can call it by only using a Z type reference
and trying to call it by using any other reference causes a compile-time error.

## Question 60

Which of the followings will make use of polymorphism when inserted individually at line 15?
(Choose three options).

```
10.  interface X { public int getValue(); }
11.  class V implements X {
12.          public int getValue() { return 1; }
13.  }
14.  class D extends V {
15.         // insert code here
16. }
```

- A. public void add(D c) { c.getValue(); }
- B. public void add(V b) { b.getValue(); }
- C. public void add(X a) { a.getValue(); }
- D. public void add(X a, V b) { a.getValue(); }
- E. public void add(D c1, D c2) { c1.getValue(); }

The correct answer are options B, C, and D.

Options B, C, and D are correct because java has provided support of polymorphism in terms of Inheritance, hence add() used are having the superclass parameters thus utilizing polymorphism.

Options A and E are  because by passing the same class instance does not exhibit polymorphism.

## Question 61

Which of the followings are correct in terms of polymorphism?

(Select two options).

- A. It is always adviced to make use of common interface instead of concrete implementation while writing code.

- B. Inheritance, method overloading and method overriding provide immense flexibilty to coding.
- C. It is necessary to make the class members private rather than public.
- D. Exception handling involves a try & catch block.

The correct answers are options A and B.

Options A and B are correct because Polymorphism is an Oops concept which advice use of common interface instead of concrete implementation while writing code and Java have an excellent support of polymorphism in terms of Inheritance, method overloading and method overriding.

Option C is because hiding data from external access relates to encapsulation.
Option D is because exception handling does not really elaborate Polymorphism.

## Question 62

Which of the following are valid on the basis of this given code snippet?
(Choose three options).

```
interface Writable { }
interface Erasable { }
```

- A. public class Pen implements Erasable,Writable { /*...*/ }
- B. public interface Pen extends Writable { /*...*/ }
- C. public interface Pen implements Writable { /*...*/ }
- D. public class Pen implements Writable { /*...*/ }
- E. public class Pen extends Writable { /*...*/ }

The correct answers are options A, B, and D.

Options A, B, and D are correct because a class can implement an interface and interface can extend another interface.

Option C is because an interface does not implement another interface.
Option E is because a class cannot extend an interface.

## Question 63

Which of the following statements are true?
(Choose two options).

```
interface I {
   //codes
}
abstract class Ab {
   //codes
}
class Sub extends Ab implements I {
   //codes
}
class Ex4 {
        public static void main(String [] args) {
                I i = new Sub(); //line 15
                Ab ab = new Sub();
               boolean t1 = i instanceof I ,t2 = ab instanceof Sub;

              System.out.print(t1 +" " + t2);
        }
}
```

- A. Compilation fails due to multiple errors.
- B. Compilation fails due to an error at line 15 as I is an interface.
- C. The output will be true false
- D. The output will be true true
- E. Compilation succeeds.

Options D and E are the correct answers.

Option D is correct as reference type can be an interface or abstract class but the object must not. So, this code compiles successfully because we have used interface I and abstract class ab as the reference type and not as the object.

Option E is correct as the class Sub extends an abstract class Ab, and implements an interface I. It can be treated as Ab or I so both instanceof result in true. So, output will be true true.

Options A and B are  as code compiles successfully.

## Question 64

Which of the following statement is true?

```
class Animal {
      //codes
}

class Mammal extends Animal {
       //codes
}

class Dog extends Mammal {
       //codes
}

class Ex3
{

                public static void main(String [ ] args)
                {

                                Animal a = new Mammal(); //line 15
                                Mammal b = new Dog();
                                Dog c = new Dog();

                                boolean t1 = a instanceof Dog ,t2 = b
                                instanceof Animal , t3 = c instanceof Animal;
                                //line 21

                                System.out.print(t1 +" " + t2 + " " + t3);

                }

}
```

- A. Compilation fails due to an error at line 15.
- B. Compilation fails due to an error at line 21.
- C. The output will be true false false
- D. The output will be false true true
- E. Compilation fails due to multiple errors

Option D is the correct answer.

In object-oriented programming, the concept of IS-A is based on class inheritance or interface implementation. The “IS-A” relationship shows that “this object is a type of that thing or not”.

Dog and Mammal objects can be treated as Animal objects as they both have Animal class as a super class but Mammal can not be threaded as Dog type because Mammal is superclass of the Dog class.

Using instanceOf we can check for IS-A relationship as follows:

393. an instanceOf Dog : this means; is object a is a type of Dog? False, as Mammal is a superclass of the Dog class.
394. b instanceOf Animal : this means; is object b is a type of Animal? true, as the object type of b is Mammal.
395. c instanceOf Animal : this means; is object c is type of Dog? true, as the object type of c is Dog.

So, the output will be false true true. So, the correct option is D.

There are no reasons for compilation to fail, so options A, B, and E are .

## Question 65

Which of the following statements are true?
(Choose two options.)

- A. Overriding method can have different argument list that of the overridden method.
- B. Overriding methods return type can be subtype that of overridden method.
- C. Overriding method may change the order of argument list that of overridden method
- D. Trying to override static method will cause a compile time error.
- E. If a method can’t be inherited, we can’t override it.

Options B and E are the correct answers.

Option B is correct as when overriding, the covariant return types are allowed.

Option E is correct as if a superclass method isn’t visible to subclass, we can’t override it even we declare a new method with same method signature, it will be just another method in sub class.

Options A and C are  as the overriding method must exactly match the argument list of overridden method including the order of arguments.

Option D is  as there won't be a compile error when trying to override a static method.

## Question 66

Which of the following are valid?
(Choose three options).

1. interface Pet { }
2. class Cat implements Pet { }
3. class Lamborghini extends Cat { }

- A. Pet a = new Cat();
- B. Cat d = new Lamborghini();
- C. Cat f = new Pet();
- D. Lamborghini c = new Cat();
- E. Pet e = new Lamborghini();
- F. Pet b = new Pet();

The correct answers are options A, B, and E.

Options A, B and E are correct because Lamborghini is a type of Cat and Cat is a subtype of Pet.

Option C is  because here Cat is a subtype of Pet. Hence, implicit casting is not possible.
Option D is  because here Lamborghini is a subtype of Cat. Hence, implicit casting is not possible.
Option F is  because Pet is an interface & interface can't be instantiated.

## Question 67

Which of the following results in compilation failure when inserted at line 5?

1. abstract class A {}
2. class B {}
3. interface X {}
4. interface D {}
5. // insert code here

- A. class E implements X {}
- B. class E extends B implements D {}

- C. class E extends A {}
- D. interface E extends X, D {}
- E. class E extends A, B {}
- F. class E implements X, D {}

The correct answer is option E.

In java, multiple inheritance is not supported so a class can't extend more than one class. So, trying to extend two class "A" and "B" is illegal, so the option E is correct.

Interfaces can extend one or more interfaces and they can't implement anything. So, option D is .

Option C is as a class can extend an abstract class.

A class can implement one or more than one interfaces, so the options A, B, and F are .

## Question 68

You are asked to create code that defines a Drink, and includes method implementation code for some drink behaviors. Drink subtype will be required to provide implementations of some, but not all, of the methods defined in Drink.

Which of the following approach correctly implements these goals?

- A. Create an abstract Drink class that defines only abstract methods.
- B. Create a Drink interface that all drink subtypes must implement.
- C. Create an abstract Drink class that defines both abstract and concrete methods.
- D. Create a concrete Drink class that defines both abstract and concrete methods.

The correct answer is option C.

Option C is correct because to the given explanation -

In order to answer the question, it is very necessary to understand the question. Let us take an example there will be a makeBubbles() implementation which is common to all the subtypes of Drink hence we can have a concrete method makeBubbles() inside Drink class and have all the subtypes use it by extending Drink class. Now tasteFlavour() is a method which a necessary part of Drink but its implementation varies with its subtypes, for example, orange juice is a Drink but has an orange flavor & apple juice is also a Drink but has an apple flavor.

Hence we can have an abstract method tasteFlavour() inside Drink class so that every subtype that claims to be Drink should have the concrete implementation of tasteFlavour.

Options A, B, and D are  because of the above explanation.

## Question 69

Which of the following statements are true?
(Choose two options).

- A. An interface can implement another interface.
- B. A class can implement more than one interface.
- C. Many classes can implement the same interface.
- D. Every class must implement at least one interface.

The correct answers are options B and C.

Options B and C are correct because one class can implement more than one interface and at the same time many classes can implement the same interface.

Option A is  because an interface extends another interface.
Option D is  because there is no such rule every class must implement at least one interface.

## Question 70

Which of the following is a legal assignment?

- A. double d=1.4;
- B. double d=1.3D;
- C. double d=1.2d;
- D. All of the above

The option D is the correct answer.

All three assignments are legal. A series of digits with a decimal point can be automatically considered as a double literal. Hence, the assignment in choice A is allowed. A double literal can also take a suffix of 'd' or 'D' however, this is optional. So, options B and C are also legal.

## Question 71

Which of the following assignments are legal?
(Choose two options).

- A. String s = null;
- B. String s = "null";
- C. String s = 'null';
- D. String s = new String(null);

The options A and B are the correct answers.

A String is an Object and it is legal to assign a null to any object reference in Java. So, option A is correct. Anything within double quotes is a String literal, so "null" is a String literal and it is legal to assign it to a String object reference.

Choice C is because only one character literal can be specified within single quotes.

Choice D is because passing a null to the String class constructor causes a compilation error.

## Question 72

Which of the following produces the output 30 when inserted at line 8?

```
429.    class Sample {
430.        public static void main(String args[]) {
431.            int temp1 = 10, temp2 = 20, result;
432.            result = compute(temp1, temp2);
433.           System.out.println(result);
434.        }
435.
436.        // insert code here
437.    }
```

- A. static int compute(int x, int y) { return x + y; }
- B. public int compute(int x, int y) { return x + y; }
- C. public int compute(int x, int y) { return; }
- D. static int compute(int x, int y) { return; }
- E. static void compute(x, y) { return (x + y); }
- F. static int compute(int x, y) { return x, y; };

The correct answer is option A.

Option A is correct because it is directly called without the class instance (hence defined as Static) and returns a valid integer type data.

Options B and C are  since the computing method should be static as it has called without using an reference.

In option C, return statement is also invalid.

Option D is  as the return statement is  .

Options E and F are  as there arguments list are  .

## Question 73

What will be the output of this code fragment?

```
12. String s = "abcdefgabc";
13. char c = s.charAt(2);
14.
15. if (c == 'c')
16.    s = s.replace('c', 'X');
17. else if (c == 'b')
18.    s = s.replace('b', 'O');
19. else
20.    s = s.replace('c', 'O');
21.    System.out.println(s);
```

- A. aOcdefgabc
- B. Compilation fails.
- C. abOdefgabc
- D. abXdefgabc
- E. abOdefgabO
- F. aOcdefgaOc
- G. abXdefgabX

The correct answer is option G.

Option G is correct The
char c = s.charAt(2); will give char c='c';

Hence s = s.replace('c', 'X'); replaces c inside abcdefgabc with X resulting into abXdefgabX

Options A, B, C, D, E, and F are  because of the above explanation.

## Question 74

What will be the output of this code fragment?

```
18. String k = " abcdcba ";
19. int x = 2;
20. k = k.trim();
21. if (k.length() < 8) {
22.     x = k.indexOf('c', 3);
23. }
24. System.out.println("x = " + x);
```

- A. x = 5
- B. x = 3
- C. x = 6
- D. x = 4
- E. x = 2

The correct answer is option D.

Option D is correct as String class' the "trim()" will Return a copy of the string, with leading and trailing whitespace omitted. And the "indexOf(char, int)" will Return the index within this string of the first occurrence of the specified substring, starting at the specified index.

So here at line 20, calling "trim()" will remove the railing whitespaces, so the length of the String "K" will be 7. So the "if" will execute and assign the value 4 to x since the second occurrence of the 'c' is located in the forth index.

Other options are  based on above explanation.

## Question 75

Which of the followings contain the correct set of methods for getting and setting the value of a boolean property 'bright', according to JavaBeans naming standards? (Select two options).

- A. public void setBright(boolean bright)  public boolean getBright()
- B. public void setBright(boolean bright)  public boolean isBright()

- C. public void setbright(boolean bright)  public boolean getbright()
- D. public void setbright(boolean bright)  public boolean isbright()

Explanation:

Options A and B are the correct answers.

According to the JavaBeans naming standards, if the property name is 'x' and the type is Type, the accessor method is of the form:

Type getX()

and the mutator method is of the form:

void setX(Type newValue)

However, the boolean property also uses another convention:

boolean isX()

void setX(boolean newValue)

So choice C is , while choices A and B are correct.

The name of the property is capitalized in the accessor and mutator methods. So, choice D is .

## Question 76

What will be the output of this code fragment?

```
class PassTest
{

        int i=3;

}
public class Test
{

        public static void f1(PassTest p)
        {

                p.i=10;
```

```
    }

    public static void f2(PassTest p)
    {

        p=new PassTest();

    }

    public static void main(String args[])
    {

        PassTest passTest=new PassTest();
        f1(passTest);
        System.out.print(passTest.i + " ");
        f2(passTest);
        System.out.println(passTest.i);

    }

}
```

- A. Compilation error
- B. Prints 3 3
- C. Prints 10 10
- D. Prints 3 10
- E. Prints 10 3

The option C is the correct answer.

In Java, all method arguments are passed by value. When object references are passed by value, the same object is referred by the original reference variable, and a copy of the object itself is received by the method. Hence, the modifications made to the object using the copy reflect within the calling method. In the given code, the method 'f1' changes the instance variable 'i' of the PassTest object to 10. This alters the object, and hence the value of 10 is printed by the first print call.

The second method assigns the reference variable copy to a new object. This does not affect the reference variable in the calling method. It refers to the same old object with i=10. So the second print() also displays 10 itself.

Hence, options A, B, D, and E are automatically .

## Question 77

What will be the output of this code fragment?

```
1. class CallUs {
2.       static String setCall(String str) {
3.             return str + "How are you doing?";
4.       }
5.
6.       public static void main(String[ ] args) {
7.             String str = "Hi! ";
8.             str = setCall(str);
9.             System.out.println("str : " + str);
10.      }
11. }
```

- A. str : Hi! How are you doing?
- B. Compilation fails due to an error at line 7.
- C. Compilation fails due to an error at line 8.
- D. Compilation fails due to an error at line 2.
- E. Compilation fails due to an error at line 3.
- F. str : Hi!
- G. str : How are you doing?

The correct answer is option A.

Option A is correct because setCall(str) is a static method and can be called directly.
Options B, C, D and option E are because compilation does not fail.
Options F and G are because of the incomplete data in str.

## Question 78

Which of the following will allow another class to get the value of the salary variable, but NOT change the value of the salary variable, when inserted at line 13?

```
11. public class Clerk {
12.       public String name;
13.             // insert code here
14.
15.       public String getName() {
16.             return name;
```

```
17.         }
18.
19.         public int getSalary() {
20.                 return salary;
21.         }
22. }
```

- A. readOnly int salary;
- B. secure int salary;
- C. private int salary;
- D. public int salary;
- E. hidden int salary;

The correct answer is option C.

Option C is correct because by making salary private, outside code won't have direct access to make any changes to it & getter method should be sufficient to acquire the value.

Options A, B, and E are because readOnly or secure or hidden are not an access specifier.

Option D is because by making public outside code will be able to make changes directly.

## Question 79

Which of the following is valid and demonstrates encapsulation when inserted at line 2?

```
1. public class Boat {
2.         // insert code here
3.         public void setGas(int v) {
4.                 gas = v;
5.         }
6. }
```

- A. protected int gas;
- B. private int gas;
- C. public int gas;
- D. struct int gas;

The correct answer is option B.

Option B is correct because principles of encapsulation dictate that the instance variable should be made private so that the level of access & control is limited at the class level rather than allowing it be modified externally. If required to be modified, we can make public methods that will serve as getters and setters.

Options A and C are due to the above explanation.
Option D is due to improper syntax.

## Question 80

What will be the output of this code fragment?

```
481.  public class Yahya {
482.      int size;
483.      public static void main(String[] args) {
484.          Yahya f = new Yahya();
485.          f.setSize(5);
486.          Yahya g = f.go(f);
487.          System.out.print(f.size + " : " + g.size);
488.      }
489.      void setSize(int s) {
490.          size = s;
491.      }
492.      public Yahya go(Yahya g) {
493.          g.setSize(2);
494.          return g;
495.      }
496.  }
```

- A. Compilation fails.
- B. 2 : 5
- C. 5 : 5
- D. 2 : 2

The correct answer is option D.

Option D is correct because this is an example of pass by reference which highlights the effect upon object references and primitive values when they are passed into methods that change the values and displays 2:2 since the value is changed inside the go method is made on the reference i.e. the actual object.

Options B and C are because of the reasons stated above and being logically correct.

Option A is  because compilation does not fail.