Question 1

What will be the output of this code fragment?

```
1. public class Pro {
2.         public static void main(String[] args) {
3.                 long [ ][ ] l2d;
4.                 long [ ] l1d = {1,2,3};
5.                 Object o = l1d;
6.                 l2d = new long[3][3];
7.                 l2d [0] [0] = (long[ ])o;
8.         }
9. }
```

- A. Compilation fails due to an error at line 5.
- B. Compilation fails due to an error at line 6.
- C. Compilation fails due to an error at line 7.
- D. Compilation succeeds and the code runs without any exception.
- E. Compilation succeeds and an exception is thrown at runtime.

Option C is the correct answer.

Option C is correct as arrays are objects, and that each array dimension is a separate type. So, for instance, l2d is of type “two-dimensional int array”, which is a different type than l1d. Line 7 attempts to assign a one-dimensional array into an int. So, compilation fails due to line 7 because they are incompatible types.

Options A and B are incorrect because lines 5 and 6 perform legal array manipulations

Question 2

Which of the following option is true about this code fragment?

```
1. import java.util.*;
2. class Animal { }
3. class Dog extends Animal { }
4.
5. public class Pro {
6.         public static void main(String[] args) {
7.                 List<Dog> c2 = new ArrayList<Dog>();
8.                 List<Animal> c3 = new ArrayList<Dog>();
9.                 ArrayList<Object> c4 = new ArrayList<Animal>();
10.         }
11. }
```

- A. Compilation succeeds.
- B. An exception is thrown at the runtime.
- C. Compilation fails due to an error at line 7.
- D. Compilation fails due to an error at line 9.
- E. Compilation fails due to multiple errors.

Option E is the correct answer.

Option E is correct as the code fails to compile due to error on line 8 and 9 because polymorphic assignments can't be applied to the generic type parameter.

Other options are incorrect as explained above.

Question 3

Which of the following will create a two-dimensional array of integers correctly?

- A. int array [ ][ ] = new int[10,10];
- B. int array [ ][ ] = new int[10];
- C. int array [ ][ ] = new int[10][ ];
- D. int [ ] array [ ] = new int[ ][10];
- E. int array[10][ ] = new int[ ][ ];

Option C is the correct answer.

Option C is correct since it uses the correct syntax for creating a two-dimensional array.

Option A is incorrect as when creating two-dimensional arrays we should use two "[ ]".

Options B and E are incorrect as it uses invalid syntax.

Option D is incorrect as we should specify the first dimension of the two-dimensional array when creating a two-dimensional array.

Question 4

Which of the following option is equal to this given code fragment?

```
int array [ ][ ][ ] = new int[1][2][3];
array [0][0][0] = 8;
array [0][0][2] = 12;
array [0][0][1] = 10;
array [0][1][0] = 40;
array [0][1][2] = 11;
array [0][1][1] = 21;
```

- A. int array1 [ ][ ]= {{8,10,12},{40,21,11}};
- B. int array [ ][ ][ ] = {{{8,10,12},{40,21,11}}};
- C. int array [ ][ ][ ] = {{8,10,12},{40,21,11}};
- D. int array [ ][ ][ ] = {{8,12,10},{40,11,21}};
- E. int array [ ][ ][ ] = {{{8,12,10},{40,11,21}}};

Option B is the correct answer.

Option B is correct as it creates an equal three-dimensional array as given in the code fragment.

Option A is incorrect since it is a two-dimensional array.

Option C and incorrect as they are invalid syntaxes because we have tried to assign a two-dimensional array to three-dimensional array declaration.

Option E is incorrect since the index positions of number 12, 10, 11 and 21 are mismatched with the given array.

Question 5

What will be the output of this program code?

```
import java.util.*;
public class Pro {
        public static void main(String[ ] args) {
                List<String> alist = new ArrayList<String>();
                alist.add("B"); alist.add("C"); alist.add("A");
                System.out.print(alist.indexOf("C"));
                Collections.reverse(alist);
                System.out.print(" " + alist.indexOf("D"));
                Collections.sort(alist);
                System.out.println(" " + alist.indexOf("A"));
        }
}
```

- A. 1 -1 0.
- B. 0 -1 1.
- C. 1 0 2.
- D. 2 followed by a runtime exception.
- E. Compilation fails.

Option A is the correct answer.

Option A is correct as the output is “1 -1 0”. At line 6 the indexOf method will print 1 as the index of the C is 1 because the List assigns index position according to the insertion order. At line 8, -1 will be printed since “D” is not included in the list. At line 10, 0 will be printed because we have sorted the alist using Collections class sort method.

Options B and C are incorrect as explained above.

Option D is incorrect as trying to get index position of a element which is not include in the list doesn’t cause an exception.

Option E is incorrect as the code compiles successfully.

Question 6

Which of the following is subjected to catch or specify requirement?

- A. Errors.
- B. All Exceptions except RuntimeException.
- C. RuntimeException .
- D. All Exceptions except Errors and RuntimeException.
- E. None of the above.

Option D is the correct answer.

Option D is correct as all Exceptions except Errors and RuntimeException are subjected to catch or specify requirement.

Options A and C are incorrect as both the Errors and RuntimeException are not subjected to catch or specify requirement.

Option B is incorrect as Errors are not subjected to catch or specify requirement.

Question 7

Given the following code :

```
1. class Pro {
2.
3.       public static void main(String args[]) {
4.            int x = 10, y = 0;
5.            try {
6.                int c = x/y;
7.            }
8.            catch (IllegalArgumentException | NullPointerException e) {
9.                System.out.print("Multi");
10.           }
11.           catch(Exception e) {
12.               System.out.print("Exc");
13.           }
14.      }
15. }
```

What is the output?

- A. No output.
- B. Multi.
- C. Exc.
- D. An uncaught exception will be thrown.
- E. Compilation fails.

Option C is the correct answer.

Option C is correct as at line 6, an ArithmeticException is thrown but it couldn’t be caught by the multiple exception catch box so the exception is passed to the catch box at line 11, so “Exc” will be printed.

Option A is incorrect as an ArithmeticException is thrown at the runtime.

Option B is incorrect as the ArithmeticException couldn’t be caught by the multiple exception catch box.

Option D is incorrect as there is no uncaught exception

Option E is incorrect as code compiles successfully.

Question 8

Which of the following option is true?

```
1. public class Pro {
2.        public static void main(String[] args) {
3.                new Pro().doIt();
4.        }
5.
6.        static void doIt() {
7.                throw new Exception();
8.        }
9. }
```

- A. Compilation succeeds if line 2 is changed to “public static void main(String[] args) throws Exception{“
- B. Compilation succeeds if line 3 is wrapped by using try/catch box and
- C. Compilation succeeds if line 7 is wrapped by using try/catch box.
- D. Compilation succeeds without additional codes.
- E. Compilation succeeds if line 3 is removed.

Option C is the correct answer.

Option C is correct as if we handle the Exception using try/catch there, and then there is no need of modifying or adding any code to the program for compilation to succeed.

Options A, B and E are incorrect. As the “doIt” method throws the exception it must also handle or declare it.

Option D is incorrect as code won’t compile without proper handling of the exception which is thrown in the “doIt” method.

Question 9

Which of the following will compile successfully when inserted at line 2?

```
41. public class Pro {
42.        // insert code here
43.                new Pro().doIt();
44.                new Pro().didIt();
45.        }
46.
47.        static void doIt()throws java.io.IOException {
48.                throw new java.io.IOException();
49.        }
50.
```

```
    static void didIt()throws ClassNotFoundException {
        throw new SecurityException();
    }
}
```

- A. public static void main(String[] args)throws java.io.IOException, ClassNotFoundException {
- B. public static void main(String[] args)throws java.io.IOException, SecurityException {
- C. public static void main(String[] args)throws java.io.IOException|SecurityException {
- D. public static void main(String[] args)throws ClassNotFoundException {
- E. None of the above.

Option A is the correct answer.

When a method is declared to throw a checked exception, then the compiler check whether the calling method has handled or declared the checked exception or not, if not then a compile-time error will be produced.

Option A is correct since the both IO Exception and Class Not Found Exception are checked exceptions, so they must be handled or declared in the main method. Using this declaration, we have declared the both checked exceptions to be thrown.

Option B is incorrect as the Class Not Found Exception is not handled or declared. and Security Exception is not a checked exception.

Option C is incorrect as the Class Not Found Exception is not handled or declared and also "|" is not valid there.

Option D is incorrect as the IO Exception is not handled or declared.

Question 10

Which of the following exceptions are possible with this code fragment?
(Choose two options).

```
public static void main(String[] args) {
    int i = Integer.parseInt(args[0]);
}
```

- A. ArithmeticException.
- B. NullPointerException.
- C. NumberFormatException.
- D. ArrayIndexOutOfBoundsException.
- E. IllegalArgumentException.

Options C and D are the correct answers.

Option C is correct as if we passed strings likes “one” and “sa” when passing strings to the main method. Number format exception is thrown when invalid conversion of a string to a numeric format.

Option D is correct as array Index Out Of Bounds Exception could be thrown if we didn’t pass any string argument to the main method.

Other exceptions are incorrect as explained above.

Question 11

What will be the output of this program code?

```
class Ex6 {
    public static void main(String args[ ]) {
        try {
            new Ex6().meth();
        } catch(ArithmeticException e) {
            System.out.print("Arithmetic");
        } finally {
            System.out.print("final 1");
        } catch(Exception e) {
            System.out.print("Exception");
        } finally {
            System.out.print("final 2");
        }
    }

    public void meth()throws ArithmeticException {
        for ( int x = 0; x < 5; x++ ) {
            int y = (int)5/x;
            System.out.print(x);
        }
    }
}
```

- A. Arithmetic final 1.
- B. Exception final 2.
- C. Arithmetic final 2.
- D. Compilation fails.
- E. Arithmetic.

Option D is the correct answer.

You can not have multiple finally clauses for one try clause. In this code, the first finally clause causes the end of the try clause. So, others catch clause is like a catch clause without a try clause, so it is illegal. So, compilation fails. So the answer is option C.

Options A, B, C, and E are incorrect as code fails to compile, so we cannot talk anything about run time.

Question 12

Which of the following modification allows this code to compile successfully?

```
1. class ExTest {
2.        Integer I;
3.        public static void main (String args[ ]) {
4.              String s;
5.              try {
6.                   s = new ExTest().I.toString();
7.              } finally {
8.
9.                   try{
10.                        int i = Integer.parseInt(args[0]);
11.                  } catch(NumberFormatException E) {
12.                       System.out.print("NumberFormat ");
13.                  } finally {
14.                       System.out.print("Fin2 ");
15.                  }
16.                  System.out.print("Fin1 ");
17.             }
18.       }
19. }
```

- A. Change the line 2 to “public static void main(String[] args)throw RuntimeException{“
- B. Change the line 2 to “public static void main(String[] args)throw Exception {“
- C. Change the line 5 to “static void print(String s)throw RuntimeException{“
- D. No modification is needed.

Option D is the correct answer.

Option D is correct because none of the code modifications will allow the code to compile but actually it doesn't need any modification.

All other modifications are illegal since when throwing an exception one should use the keyword “throw”.

For example,
"throw new Exception();"

The keyword “throws” must be added to the method body to declare that this method will throw an exception.

For example,
public static void main(String args[]) throws Exception {

Question 13

What will be the output of this program code?

```
1. class ExTest {
2.          Integer I;
3.         public static void main(String args[]) {
4.                 String s;
5.                 try {
6.                       s = new ExTest().I.toString();
7.                 } finally {
8.
9.                try {
10.                     int i = Integer.parseInt(args[0]);
11.               }catch(NumberFormatException E){
12.                    System.out.print("NumberFormat ");
13.               }finally{
14.                    System.out.print("Fin2 ");
15.               }
16.                   System.out.print("Fin1 ");
17.               }

18.         }

19. }
```

The given command line invocation is java ExTest 10

- A. NumberFormat Fin1 Fin2.
- B. NumberFormat Fin1.
- C. NumberFormat Fin2 .
- D. Fin1 Fin2 .
- E. Fin1 .
- F. Fin2 Fin1 followed by uncaught exception.

Option F is the correct answer.

At line 7, Null Pointer Exception is generated as we try to invoke to String method on the

Null reference. But there is no catch clause to catch it and it has only finally clause, so finally executes. In that the finally clause, there is another try clause. In that try clause, there will be no exception occurs so it goes to its' finally and prints Fin2. After that Fin1 prints as there is no exception generated inside try clause of first finally clause. As the Null Pointer Exception remains uncaught, it will be thrown at the end. So the answer is F.

At line 11, Number Format Exception won't be thrown as we have passed 10 as arg[0]. So, Number Format won't print. So A, B, and C are incorrect.

As uncaught exception is thrown, options D and E are incorrect.

Question 14

Which of the following will only print "whiz" as the output when inserted at line 12?

```
1. public class Pro {
2.
3.        public static void main(String[ ] args) {
4.                try {
5.                     print();
6.                } catch(Exception e){}
7.        }
8.
9.        static void print() {
10.               try
11.               {
12.                   //insert here
13.               } catch(ClassCastException e) {
14.               } finally
15.               {
16.                   System.out.print("whiz");
17.               }
18.               System.out.print("labs");
19.         }
20. }
```

- A. throw new Exception(); .
- B. Insert nothing.
- C. throw new ClassCastException();
- D. throw new NullPointerException();

Option D is the correct answer.

Option D is correct as it will produce the expected output. As the catch box for Class Cast Exception can't catch the Null Pointer Exception finally is executed and then the control is returned to the main method, so the output will be only "whiz".

Option A is incorrect as it will cause a compile-time error.

Option B is incorrect as without modifications “whizlabs” will be printed as the output.

Option C is incorrect since, with it, the output will be “whizlabs”.

Question 15

Which is of the following can appear before the package statement in a single code file?

- A. Comments and/or another package statement.
- B. Class statements and/or comments.
- C. Another package statement.
- D. Import statements and/or class statement.
- E. Comments.
- F. None of the above.

Option E is the correct Answer.

While creating a java source code file, one should follow following order.

package statement - comes first and only one such a statement is allowed.

import statements - comes after the package statement and multiple import statements are allowed.

class statements - comes after the import statements.

We can use comments anywhere in the source code files as they are ignored by the compiler in the compile-time.
As explained above option E is correct.

Options A, B, and D are incorrect as import and class statements should come after the package statement as explained above.

Option C is incorrect as there can be only one package statement for a source code file.

Question 16

Which of the following option is true about these given statements?

I. The scope of a local variable is wider than the scope of an instance variable.

II. The instance variables are only visible to the methods in which they are declared.

III. The variable “x” in the declaration “int x = 10” should be an instance variable.

- A. Only I.
- B. Only II.
- C. Only I and II.
- D. Only I and III.
- E. None of the above.

Option E is the correct Answer.

Option E is correct as all statements are incorrect.

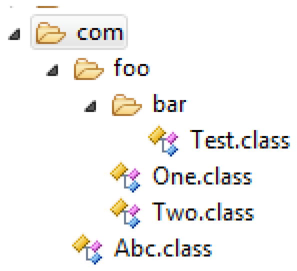
Statement I is incorrect as the instance variables have a wider scope than the local variables. The scope of a local variable is limited to the method in which it is declared and scope of an instance variable spread over the class.

Statement II is incorrect since the variables the instance variables are not declared inside methods. They are declared inside the class without using a static modifier.

Statement III is incorrect as we can’t say anything about the variable “x” as it could be a local variable or instance variable because it doesn’t use any modifier. If It has used a modifier except “final” then it should be an instance variable.

Question 17

Consider following directory structure.
(Assume that current directory is the same directory where “com” is located.)



Now consider the following import statement;
import com.foo.*;

Which of the following shows the set of the classes which will be imported by the above statement?

- A. Abc, One, Two, Test.
- B. One, Two, Test.
- C. One, Two.

- D. Abc, One, Two.
- E. None of the above.

Option C is the correct answer.

We can use “*” for importing all the classes of a package. So, the given imported statement will import both “One” and “Two” classes. Packages appear to be hierarchical, but they are not. For example, the Java API includes a java.awt package, a java.awt.color package, a java.awt.font package, and many others that begin with java.awt. However, the java.awt.color package, the java.awt.font package, and other java.awt.xxxx packages are not included in the java.awt package. So here, the classes of “bar” package are not imported. Therefore, option C is correct.

Other options are incorrect as explained above.

Question 18

What will be the output of this program code?

```
public class Pro {
        public static void main(int []i) {
                System.out.print("main1");
        }
        public static void main(String... c) {
                System.out.print("main2");
        }
        public static void main(String [][]c){
                System.out.print("main3");
        }
}
```

- A. main1.
- B. main2.
- C. main3.
- D. An Error is thrown at the runtime, stating that, Main method not found in class Pro.
- E. Compilation fails.

Option B is the correct answer.

The signature of the main method must take one form of the following two forms;
public static void main(String[] args) or public static void main(String… args)
And, it can also be a final.

The starting point of a program is the main method; simply means that the program starts to execute statements which are located inside the main method. So, here “main2” will be printed. Therefore, option B is correct.

Options A and C are incorrect as they are not the main method. They are just overloaded versions of the main method and it is legal.

Option D is incorrect as there will be no error thrown as the code has the main method.

Option E is incorrect as the code compiles successfully

Question 19

What will be the output of this program code?

```
public class Pro {

    static int x = 11;
    private int y = 33;

    public static void main(String args[ ]) {
        Pro pro = new Pro();
        pro.call(5);
        System.out.print(Pro.x);
        System.out.print(" " + pro.x);
        System.out.print(" " + pro.y);
    }

    public void call(int x) {
        x = 22;
        y = 44;
    }
}
```

- A. 22 22 33.
- B. 22 22 44.
- C. 11 11 33.
- D. 11 11 44.
- E. 11 22 44.
- F. Compilation fails.

Option D is the correct answer.

Option D is correct since inside the call method only the instance variable “y” and the method argument “x” is changed. Here the static variable is shadowed by the argument variable “x”. Shadowing occurs when two variables are located in two different scopes with the same name. In such case, the closet scope variable is chosen.
For example here, the call method changes the value of the argument “x” from 5 to 22 since the argument is in the local variable scope. So, the static variable remains unchanged. Therefore, the output will be 11 11 44.

Other options are incorrect as explained above.

Question 20

What will be the output of this code fragment?

```
public class Pro {
    static int x = 50;
    public final static void main(String[ ] a) {
        Integer []a = new Integer[2];
        a[1] = 10;
        for (Integer I:a)
        System.out.print(I);
    }
}
```

- A. null10
- B. 10
- C. A null pointer exception is thrown.
- D. Compilation fails.
- E. None of the above.

Option D is the correct answer.

Option D is correct as the code fails to compile. In the main method, we have declared the argument as “String[] a” and inside the main method we have tried to create an Integer type array with the name “a”. This will cause a compile time error since we can't declare two variables with the same name inside the same scope. Here, both method argument and the Integer array belong to the same local scope.

Other options are incorrect as the code fails to compile.

Question 21

What will be the output of this code segment?

```
import java.util.*;
public class Pro {
    public static void main(String[] args) {
        int [ ]a = {3,2,1,0};
        int y = a.length;
        int x = y - 1;

8.          while( y >= 0 ) {
9.               System.out.print( a [ --y ] );
10.         }
11.     }
12. }
```

- A. 0123
- B. 3210
- C. Never ending loop after printing 3210 .
- D. 0123 followed by an exception.
- E. Compilation fails.

Option D is the correct answer.

Length of array “a” is 4, so the value of the variable y is 4. Execution of while loop will print array element reverse as variable “ y ” is decremented by one at line 6 , So printing of elements start with 0 (a[3]) and runs till the value of variable “y” equal to -1. When it reaches to -1, an ArrayIndexOutOfBoundsException is thrown. So, option D is correct.

Options A, B, and C are incorrect as an exception is thrown at the runtime.

Option E is incorrect as the code fails to compile.

Question 22

What will be the output of this program code?

```
1.  public class Pro {
2.         static String out = "";
3.         public static void main(String[] args) {
4.
5.                 int x = 5, y = 8;
6.
7.                 if (x ++== 5)
8.                   out += "1";
9.                 if (x!=6) { }
10.               else if (x > 9) { out += "2"; }
11.               else if (y < 9) { out += "3"; }
12.               else if (x == 6) { out += "4"; }
13.               else { out += "5"; }
14.                 System.out.println(out);
15.        }
16. }
```

- A. 1
- B. 134

- C. 15
- D. 13
- E. Compilation fails.

Option D is the correct answer.

When using if-else if, we should remember that when one condition is true then below else ifs are not checked. But if the condition is false then program check below else if and finally executes “else” when no condition is true.
Option C is correct.

At line 7, if condition is true since x ++== 5 is true, So line 8 executes.

Then at line 9, there is another if and all the else if belong to that if.

At line 9, if condition is false since the value of x is 6 e, so line 10 else if condition will be checked.

At line 10, else if condition is false since x<9 false, so line 11 else if condition will be checked.

At line 11, else if condition is true since y<9 is true, so 3 will be added to the variable “out” and below else if will be skipped.

So, the output will be 13, therefore option D is correct.

Other Options are incorrect as explained above.

Question 23

What will be the output of this program code?

```
1. class Pro {
2.
3.        public static void main(String[] args){
4.
5.                int value1 = 1;
6.                int value2 = 2;
7.
8.                if (value1 == value2)
9.                     System.out.print("1");
10.               if (value1 != value2)
11.                    System.out.print("2");
12.               if (value1 > value2)
13.                   System.out.print("3");
14.               if (value1 < value2)
15.                 System.out.print(4);
16.               if (value1 => value2)
17.                 System.out.print("5");
18.
19.         }
20. }
```

- A. 245
- B. 24
- C. Compilation fails due to an error at line 13.
- D. Compilation fails due to multiple errors.
- E. None of the above.

Option E is the correct answer.

Option E is correct as code fails due to an error at line 16, as “=>” is not a valid operator. It should be corrected to “>=”. If it is corrected then the output would be 24.

Options A and B are incorrect as code fails to compile.

Option C is incorrect as the print method of the PrintStream class has overloaded version which can take an int as a parameter.

Question 24

If “x.equals (y)” return true, then which of the following is false?

- A. “y.hashCode()” must equal to “x.hashCode()”
- B. Both “x” and “y” objects may have same field status.
- C. “y.hashCode()” may not equal to “x.hashCode()”
- D. None of the above.

Option C is the correct answer.

If “x.equals(y)” return true then the hash codes of both objects must equal. So option A is true.

Option B is true as even the objects may have different values or may have same values for fields. In both cases “x.equals(y)” may still return true as it is decided by the “equals()” method. For example, if the class has three fields called id, name, age, then if we override the “equals()” method to compare the only value of “id”, then equality will be decided only based on the id.

Option C is false because the hashcodes must equal if “x.equals(y)” return true, so option C is correct

Question 25

What will be the output of this program code?

```
1. public class Pro {
2.
3.        public static void main(String args[]) {
4.
5.                int y = 5, j = 11;
6.
7.                if (false && j++==11)
8.                      System.out.print ( y );
9.                else if (true || --y == 4)
10.                     System.out.print ( y );
11.               else ( y == 5 ){ }
12.        }
13. }
```

- A. The output will be 6.
- B. The output will be 4.
- C. The output will be 5.
- D. No output.
- E. Compilation fails.

Option E is the correct answer.

The code fails due to error on line 11 because we cannot use a conditional clause with else. So, option E is correct.

If we modify line 11 to “else{ }” then the code will compile and the first if test “j++==11” will not be checked as we use &&. So, it’ll not print y at that time. But “else if’s first condition is true it won’t check “—j==4” and it’ll print the value of “y”. Since the value of “j” hasn’t changed the output will be 5. So, the answer would be C.

Question 26

What will be the output of this program code?

```
1. class Pro {

2.

3.          public static void main(String[] args) {

4.                  int x = 20;
5.          int y = 13;

6.          System.out.print("" + x + y + " ");

7.          System.out.print( x + y + " ");

8.          System.out.print(x + " " + y);

9.      }

10. }
```

- A. 33 33 33
- B. 2013 33 20 13
- C. 33 33 20 13
- D. 2013 2013 2013
- E. Compilation fails.

Option B is the correct answer.

Option B is correct as the output is “2013 33 20 13”.

At line 6, the result will be 2013 because before adding x and y, we have used the “+” operator for concatenation, in such case other “+” operators will also act as concatenation operator instead of adding.

At line 7, the result will be 33 since we haven’t used “+” as concatenation operator first. So, “+” will add two numbers and result in 22.

At line 8, we have placed “ “ for spacing so the result is two numbers separated by space.

Question 27

What will be the output of this program code?

```
1. class Ex1 {
2.
3.      public static void main(String args[]) {
4.
5.          String s = "OCAJP";
6.
7.          switch(s) {
8.                  case "OCAJP" : {System.out.print("A");}
9.                  case "ocajp" : {System.out.print("B");}
10.                 case default : System.out.print("default"); break;
11.         }
12.        }
13. }
```

- A. A
- B. AB default
- C. Compilation fails due to an error at line 7.
- D. Compilation fails due to an error at line 8.
- E. Compilation fails due to an error at line 10.

Option E is the correct answer.

Option E is correct as the code fails to compile due to an error on line 10. We can't use "case" when defining the default case so at line 10 "case default" will cause a compile-time error.

Options A and B are incorrect as the code fails to compile.

Option C is incorrect as using a string as a switch expression is valid from java SE 7.

Option D is incorrect as using a string as a case constant is valid from java SE 7.

Question 28

Which of the following statement is true?

- A. There is no difference between using "==" or the "equals()" method for testing equality between objects.
- B. Using "==" to test equality for primitives, will always provide accurate result.
- C. If "equals()" method returns true then "==" test will also result true.
- D. We can overload the "equals()" method to define how the two objects are going to be equal.
- E. Using the "equals()" method to test equality for primitives, will always provide accurate result.

Option B is the correct answer.

Option B is correct since with for primitives "==" always gives the correct result.

Option C is incorrect because while using "==" for testing equality between two references, it returns true only the two references refer to the same object. So we can't do meaningful testing with "==" when considering objects. But when using "equals()" method, it returns true only given condition inside "equals()" method are satisfied and it deso not need both references to refer the same object.

The "equals()" method compares two objects for equality and returns true if they are equal. To test whether two objects are equal in the sense of equivalency (containing the same information), you must override the equals() method. So, option A is incorrect as explained above.

Option D is incorrect since we must override the “equals()” method but not overload it.

Option E is incorrect since we can’t use the “equals()” method with primitives.

Question 29

What will be the output of this program code?

```
class Pro {

    public static void main(String args[]) {

        byte i = (byte)(Math.random() * 150);

        switch(i) {
            case 50 : {System.out.print("A");}break;
            case 100 : System.out.print("B");
            case 150 : System.out.print("D"); break;
            default : System.out.print("C");
        }
    }
}
```

Note: Consider “Math.random()*150)” generates random numbers between 0 - 150

- A. A
- B. AB
- C. BD
- D. CD
- E. Compilation fails.

Option E is the correct answer.

Option E is correct as this code fails to compile because we have used a byte as switch expression and in case, constants we have used the value 150 which exceeds the range of a byte.

Other options are incorrect as the code fails to compile.

Question 30

Which of the following will compile successfully?

- A. for(int j = 0,int k=5; j < k; k--) ;
- B. for(;;System.out.print("a")) ;
- C. for();
- D. for(int k = 10; k--; k>0) ;

Option B is the correct answer.

General format of the for loop is;
for(initialization; Boolean expression; update){ //Statements }

Option B is correct as there we use correct syntax. While creating for loops all three blocks are optional, simply means that we can skip initialization, boolean expression or update statements.

Option A is incorrect as defining multiple variables in a for loop uses the incorrect syntax.

Option C is incorrect as we should use ";" for separating three statements.

Option D is incorrect as we have used boolean expression and update statements wrong places.

Question 31

What will be the output of this program code?

```
class Pro{

    public static void main(String[] args){
        int x = 0;
        String [] animal = new String[3];

        do {animal[x] = "Cat"; x++;} while(false);
        do {animal[x] = "Dog";} while(x>animal[x++].length());
        do {animal[x] = "Rat";} while(x>3);

        for(String s:animal){
            System.out.print(s + " ");
        }

    }
}
```

- A. Cat Dog Rat
- B. Dog Rat

- C. An Exception is thrown at the runtime before producing any output.
- D. Compilation fails due to an error at line 7.
- E. Compilation fails due to an error at line 8.

Option A is the correct answer.

While using "do-while", the statements in "do" block will be executed at least one time. So, here all do-while loops add elements to the String array even the all while loop conditions are false. So, the output will contain all three Strings added by three do-while loops. Therefore, option A is correct.

Option B is incorrect as explained above.

Options D and E are incorrect since the code compiles successfully

Question 32

Which of the following will print all the elements of array "arr"?

int arr[ ][ ] = { {1, 3, 5}, {7,8} };

- A. for(int [ ]a : arr) for(int i: a) System.out.print (i + " ");
- B. for (int [ ]a : arr); for (int i: a) System.out.print(i + " "); for(;;System.out.print("a")) ;
- C. for (int a : arr) System.out.print (a + " ");
- D. None of the above.

Option A is the correct answer.

Option A is correct as it produces the expected output.

Option B is incorrect since the two for loops work individually so in second for loop, trying to use the variable a, causes compile-time error.

Option C is incorrect as the arr is a two-dimensional array, so "for(int a: arr)" is an invalid syntax.

Question 33

What will be the output of this code fragment?

```
1. public class Pro {
2.       public static void main(String[ ] args) {
3.
4.             for(int j = 0,k=5; j < k; k--) ;
5.          for(int j = 0; j++ < 3;) ;
6.          for(int i = 0; i < 5; i++, System.out.print(i + ".Hi ")) ;
7.
8.     }
9. }
```

- A. 1.Hi 2.Hi 3.Hi 4.Hi 5.Hi
- B. 0.Hi 1.Hi 2.Hi 3.Hi 4.Hi 5.Hi
- C. An exception is thrown at runtime.
- D. Compilation fails due to an error at line 6.
- E. Compilation fails due to multiple errors.

Option A is correct.

Option A is correct as the code produces the output as “1.Hi 2.Hi 3.Hi 4.Hi 5.Hi”

Option B is incorrect as the “1.Hi” will be printed because “i++” executes before print statement.

Option C is incorrect as skipping any part of for loop doesn’t cause any compile-time error, in fact, we can skip all three part if necessary.

Option D is incorrect because there is no issue in that for loop, many mistakenly call the third expression of for loop as “increment expression”, but we can put any virtually arbitrary code statements that you want to happen with each iteration of the loop.

Question 34

Which of the following will produce the output as 50 40 30 20 10 when inserted at line 6?

```
232.        public class Pro {
233.            public static void main(String args[]) {
234.                L1: for(int i = 5, j = 0; i>0;i--) {
235.                    L2: for(;j<5;j++) {
236.                        System.out.print(i + "" +j +" " );
237.                        //insert here
238.                    }
239.                }
240.            }
241.        }
```

- A. if (j == 0) continue;
- B. if (j == 0) continue L1;
- C. if (j == 0) continue L2;
- D. if (j == 0) break L1;

- E. None of the above.

Option B is the correct answer.

Option B is correct as it will produce the expected output.

Options A and C are incorrect as “continue” and “continue L2” are not effected on the output and they will produce the output as 50 51 52 53 54. Even we do not insert them to line 6, the output will be the same.

Option D is incorrect as it will break entire looping by breaking the L1 loop after printing 50 as the output.

Question 35

Which of the following will produce the output as 1 3 8 when inserted at line 6 ?

```
public class Pro {
        public static void main(String args[]) {
                int arr[ ][ ] = { {1, 3, 5},{7,8} };
                out : for ( int [ ]a : arr ) {
                     for ( int i: a ) {
                     // }
                }
        }
}
```

- A. if (i == 7) continue; System.out.print(i + " "); if (i == 3) break out; }
- B. if (i == 7) continue out; System.out.print(i + " "); if (i == 3) break; }
- C. if (i == 7) continue; System.out.print (i + " "); } .
- D. if (i == 7) continue; System.out.print(i + " "); if (i == 3) break; }
- E. None of the above.

Option D is the correct answer.

Option D is correct as it will produce the expected output.

Option C is incorrect as it will produce the output 1 3 5 8.

Options A and B are incorrect as they will produce the output 1 3.

Question 36

Which of the following option is true?

```
1. class Ab {
2.        static void meth1() {
```

```
3.                 System.out.print(" A B");
4.         }
5. }
6.
7. class Cd extends Ab {
8.         protected static void meth1() {
9.                 System.out.print(" C D");
10.        }
11. }
12.
13. class Ex4 {
14.        public static void main(String [ ] args) {
15.                Ab ab = new Cd();
16.                ab.meth1();
17.        }
18. }
```

- A. The output will be A B C D
- B. The output will be A B
- C. The output will be C D
- D. An exception is thrown.
- E. Compilation fails due to an error at line 8.

Option B is the correct answer.

Option B is correct since polymorphism doesn’t apply to a static method, so invoking an overridden method with super class reference will cause invoking the superclass method itself. So, the output is A B.

Options A and C are incorrect as explained above.

Overriding static methods is legal so there won’t be any compile time errors, so option E is incorrect.

Question 37

Which of the following option is true about given code?

```
class Animal{
        void makeSound(){
                System.out.print("Animal sound");
        }
}
interface Runnable{void run(); }

class Dog extends Animal implements Runnable{
```

```

          void makeSound()throws RuntimeException{
                System.out.print("bark");
          }

          public void run(){
                   System.out.print(" Dog runs");
          }
     }
     class Pro {
           public static void main(String [] args){
                    Animal dog = new Dog();
                    dog.makeSound();
                    dog.run();
           }
     }
```

- A. bark Dog runs
- B. Animal sound Dog runs
- C. Compilation fails due to an error on line 10.
- D. Compilation fails due to an error at line 22.
- E. Compilation fails due to multiple errors.

Option D is the correct answer.

Option D is correct as the code fails to compile due to an error at line 22. The reference type of the “dog” object is Animal so we can’t invoke any method which is not defined in the Animal class on dog object. Therefore, the code fails as the run method is not defined in the Animal class.

Options A and B are incorrect as the code fails to compile.

Option C is incorrect as the overriding method can be declared to be thrown any unchecked exception.

Question 38

What will be the output of this program code?

```
     class Person{
           Person(){
                System.out.print("CP ");
           }
           static{ System.out.print("SP ");}
      }
      class Teacher extends Person{
              Teacher(){
                     System.out.print("CT ");
              }
           { System.out.print("IT "); }
      }
      class Pro {
               public static void main(String [] args) {
                      Person p1 = new Teacher();
               }
      }
```

- A. SP IT CP CT
- B. CP CP IT SP
- C. SP CP IT CT
- D. SP CP IT CT SP
- E. Compilation fails.

Option C is the correct answer.

The static contents are executed when the class is loaded so before the constructors are executed, so the SP will be printed first. Then the constructor of Person will print CP. Then the Teacher class instance code block will print IT before Constructor prints CT. So, the output is SP CP IT CT. Therefore, option C is correct.

Static content will print only ones so option D is incorrect.

Other outputs are incorrect as explained above.

The Code compiles successfully, so option E is incorrect.

Question 39

Which of the following will compile successfully when inserted at line 17?

```
 class Sup {
        protected void print() {
              System.out.print("A "); }
        }

 class Sub extends Sup {
       public void print() {
             System.out.print("B ");
       }
       void print(String s) {
            System.out.print(s);
       }
```

```
}

public class Main {
        public static void main(String [] args)throws Exception {
                //insert here
        }
}
```

- A. new Sup().print("C");
- B. print();
- C. new Sub().print('C');
- D. ((Sub)new Sup()).print("C");
- E. None of the above.

Option D is the correct answer.

Option D is correct as it allows this code to compile but it will cause a Class Cast Exception because we have tried to downcast the Sup-type object to be Sub-type.

Option A is incorrect as there is no “print(String s)” method is defined in the Sup class, so inserting it at line 17 causes a compile-time error.

Option B is incorrect as the print method is not a static method.

Option C is incorrect since it fails the compilation as there is no overloaded version of the print method which can take char as the parameter.

Question 40

Consider this given code:

```
class Bird extends Animal{ }
class Animal{}
class Pro {
        public static void main(String []args) {
                Animal aa = new Bird();
                Bird bb = new Bird();
                Animal aaa = new Animal();
        }
}
```

And following three statements about the above code fragment.

I. Object type of the variable “aa” is Animal.
II. Reference type of the variable “bb” is Bird.

III. Object type of the variable “aaa” is Animal.

Which of the following statement is true?

- A. Only I.
- B. Only II.
- C. Only I and II.
- D. Only II and III.
- E. All the statements are true.

Option D is the correct answer.

Statement I is incorrect as the object type of the variable “aa” is “Bird” since we have used “new Bird()” for initializing the object.

Statement II is correct. The Reference type of the variable “bb” is Bird as we declared the variable “bb” with “Bird” reference.

Statement III is correct the object type of the variable “aaa” is “Animal” since we have used “new Animal()” for initializing the object.

So, option D is correct as only the statement II and III are correct.

Question 41

When a class is extended by another class then the subclass can -

- A. All methods of super class can be used directly as they are located in subclass.
- B. All fields of super class can be used directly as they are located in subclass.
- C. We can hide the static method of super class by defining a method with same method signature in the subclass.
- D. All of the above.
- E. None of the above.

Option C is the correct answer.

Option C is correct as we can hide static methods of superclass by defining a method with the same name of superclass in the subclass.

Options A and B are incorrect as subclass can’t only use the private methods and the private fields of the superclass directly.

- Question 42
-
- Incorrect

Consider this given program code?

```
363.        class A {
364.                A meth() {
365.                    return new A();
366.                }
367.        }
368.
369.        class B extends A {
370.                B meth() {
371.                    return new B();
372.                }
373.        }
374.
375.        class Check {
376.                 public static void main(String [] args) {
377.                        //codes
378.                 }
379.        }
```

And the two command line invocations:

```
I. javac -source 1.4 Check.java
II. javac -source 1.7 Check.java
```

Which of the following are correct about the above code?
(Choose two options).

- A. With I, Compilation fails with an error at line 8.
- B. With I, Compilation succeeds.
- C. With II, Compilation succeeds.
- D. With II, Compilation fails with an error at line 8.
- E. With I, an exception is thrown at run time.

Options A and C are the correct answers.

Overriding Methods can change the return type only within the bounds of covariant returns. Simply means that the overriding method can return a subtype of the return type of the superclass method. Before java 1.5 this code fails as java didn’t allow covariant return types in those versions.

So, with I, the code compilation will fail as explained above, therefore the option A is correct and B, E are incorrect.

Option C is correct as explained above.

Question 43

Which of the following statement is true?

- A. The up casting increases the capabilities of the object.
- B. The up casting can’t be occurred implicitly.
- C. Casting subclass object to super class object is known as down casting.
- D. With down casting, a ClassCastException is possible.
- E. None of the above.

Option D is the correct answer.

Option D is correct as with down casting, a Class Cast Exception is possible.

Option A is incorrect as the when we casting the object becomes more general so it will remove object’s unique actions. Therefore object capabilities are reduced.

Option B is incorrect as up casting can be occurred implicitly but down casting should be done explicitly.

Option C is incorrect as the casting subclass object to super class object is known as up casting.

Question 44

Which of following represents a correct abstract class?

- A. abstract class A{ public void meth() { } }
- B. abstract class A{ public void meth(); }
- C. class A{ public void meth(); }
- D. final abstract class A{ public abstract void meth(); }
- E. None of the above.

Option A is the correct answer.

Option A is correct. Abstract methods are optional for an abstract class, it is enough a class to mark with an abstract keyword to class to be abstract.

Option B is incorrect as the abstract methods should be marked using the abstract keyword.

Option C is incorrect as an abstract class should be marked with the abstract keyword and also abstract method should mark with abstract keyword.

Option D is incorrect as the abstract class can’t be final.

Question 45

What will be the output of this program code?

```
395.    abstract class Base {
396.        public void method() {
397.            System.out.println("method Base");
398.        }
399.    }
400.
401.    public class Pro extends Base {
402.
403.         public static void main(String argv[]) {
404.              Object a = new Base();
405.               a.method();
406.         }
407.         public void method() {
408.              System.out.println("method Pro");
409.        }
410.    }
```

- A. method Pro.
- B. method Base .
- C. An exception is thrown at the runtime.
- D. Compilation fails due to an error at line 11.
- E. Compilation fails due to multiple errors.

Option E is the correct answer.

Option E is correct as code fails to compile due to errors on line 10 and 11. The base is an abstract class so we can’t instantiate it so line 10 causes a compile-time error. There is no such a method called “method” in the Object class, so trying to invoke the method “method” causes another compile-time error.

Other options are incorrect as explained above.

Question 46

What will be the output of this program code?

```
class Animal {
      void eat(){ System.out.print("Animal eats"); }
}
class Bird extends Animal {
      void eat(){ System.out.print("Bird eats"); }
      void print(){ super.eat(); }
}
public class Pro {
      public static void main(String [] args) {
            Bird ab = new Bird();
            ab.print();
      }
}
```

- A. Animal eats
- B. Bird eats
- C. No output.
- D. An exception is thrown.
- E. Compilation fails.

Option A is the correct answer.

In Bird class the “print()” method will invoke the superclass version of the “eat()” method since we have used the keyword super there. S,o “Animal eats” will be printed as the output. Therefore, option A is correct.

Option B is incorrect as we have used the keyword “super” at line 6.

Option D is incorrect as there is no exception.

Option E is incorrect as code compiles successfully.

Question 47

What will be the output of this program code?

```
interface I {
      int x = 10;
      void print();
}

public class Pro implements I {
       public static void main(String argv[]) {
             Pro pro = new Pro();
             pro.print();
        }
        public void print() {
            System.out.println(x++);
        }
    }
```

- A. 10
- B. 11
- C. An exception is thrown at the runtime.
- D. Compilation fails due to an error at line 12.
- E. Compilation fails due to multiple errors.

Option D is the correct answer.

Option D is correct since the code fails to compile due to an error at line 12. Interface variables are implicitly public, static and final, so trying to change the value of the variable x causes a compile-time error.

Other options are incorrect as explained above.

Question 48

Which of following represents a correct interface?

- A. interface I{ protected int x = 10; }
- B. interface I{int x = 10; void print(){} }
- C. abstract interface I{ int x = 10; void print();}
- D. interface I{int x = 10; private void print();}
- E. None of the above.

Option C is the correct answer.

Option C is correct as interfaces a 100% abstract so they can be marked with the abstract keyword but it is optional.

Option A is incorrect as the Interface variables are implicitly public, static and final so the modifier protected is not allowed here.

Option B is incorrect as the interface methods are implicitly abstract. An abstract method can’t have a body.

Option D is incorrect as interface methods are implicitly public, so trying to create private method causes a compile-time error.

Question 49

Which statement is true?

I. The variables declared inside a method are called as member variables.

II. The instance variables are initialized to its default value.

III. The variable “x” in the declaration “int x = 10” should be an instance variable.

- A. Only I.
- B. Only II.
- C. Only III.
- D. Only I and II.
- E. Only I and III.

Option B is the correct answer.

Option B is correct as only the statement II is correct.

Statement II is correct as the instance variables and class variables are initialized to its default value.

Statement I is incorrect since the variables inside a method are called as local variables.

Statement III is incorrect as we can’t say anything about the variable “x” as it could be a local variable or instance variable because it doesn’t use any modifier. If It has used a modifier except “final” then it should be an instance variable.

Question 50

What will be the output of this program code?

```
public class Pro {
    static int x = 10;
    public static void main(String args[ ]) {
        Pro pr = new Pro();
        pr.x = 5;
        int y = x/pr.x;
        System.out.print("y =");
        System.out.print();
        System.out.print( y );
    }
}
```

- A. y = 1
- B. y = 2
- C. y = 0
- D. Compilation fails due to an error at line 6.
- E. Compilation fails due to an error at line 8.

Option E is the correct answer.

While invoking methods, we should pass suitable arguments. So, the code fails to compile as we have tried to invoke the "print()" method without passing any parameters but there is no such overloaded version of the print method of the PrintStream class. Therefore, option E is correct.

Options A, B, and C are incorrect as the code fails to compile.
If we removed the line 8, then the output would be y=1.

Question 51

What will be the output of this program code?

```
class Pro implements inter {
    public static void main(String args[]) {
        System.out.print(s);
        System.out.print(inter.value);
    }
}
interface inter {
    static int value = 15;
    String s = "Value is: ";
}
```

- A. Value is: 15
- B. Compilation fails due to an error at line 3.
- C. Compilation fails due to an error at line 4.
- D. Compilation fails due to multiple errors.
- E. An exception will be thrown at runtime.

Option A is the correct answer.

The interface variables are implicitly static so we can access them using interface name, and also we can access them directly from a static context. So, this code will compile fine and produce the output as "Value is- 15". Therefore, option A is correct.

Option B is incorrect as at line 3, trying to access a variable s without interface name or reference variable won't cause any problem because variable "s" is inherited from interface inter.

Option C is incorrect as we can also access interface variable using interface name.

Option D is incorrect since there are no compile-time errors.

Question 52

Which of the following set contains only primitive literals?

- A. 1, ‘c’, “a”
- B. 1, 1.5f, True
- C. ‘BF’, 10, “Sure”
- D. 1.2D, 1f, ‘c’
- E. None of the above.

Option D is the correct answer.

Option D is correct since all are primitive literals, they are double, float and char.

Option A is incorrect as “a” is a String literal.

Option B is incorrect as True is incorrect literal is should be true.

Option C is incorrect as ‘BF’ is illegal; char literal can only have one letter.

Question 53

Which methods from the String class modify the object on which they are called?

- A. charAt(int index)
- B. concat(String str)
- C. toLowerCase()
- D. split(String regex)
- E. None of the above.

Option E is the correct answer.

Option E is correct since no method in String class effect on the string object which they are invoked on. For example, consider the following code fragment;

String s = “abc”;

s.concat(“def”); // after executing this the value of the variable s, still remain as “abc”.

What happens is the concat method returns the string “abcdef” without changing the value of the variable s.

Question 54

Which of the following method can be found in both StringBuilder and String classes?

- A. concat(String str)
- B. isEmpty()
- C. charAt(int index)
- D. toUpperCase()
- E. None of the above.

Option C is the correct answer.

Option C is correct as, among the given methods, only the charAt(int index) method can be found in both classes.

Options A, B, and D are incorrect as these methods are defined in the String class.

Question 55

What will be the output of this program code?

```
class Pro {
      public static void main(String [] args) {
           StringBuilder s =new StringBuilder("Java");
           s.concat(" SE 6");
          System.out.print(s.length() + " ");
          System.out.print(s.capacity());
      }
}
```

- A. 9 9
- B. 9 16
- C. 9 20
- D. 20 20
- E. Compilation fails.

Option E is the correct answer.

Option E is correct as code fails to compile due to an error on line 4. StringBuilder class doesn't have a method called "concat()". The "concat() method is a method belong to the String class. We should have used the "append()" method there.

If we change line 4 to "s.append("SE 6")", then the option C would be correct because the length of the String “Java SE 7” is nine, 9 will be printed when line 5 executed. And we have used StringBuilder class’ “public StringBuilder(String str)” constructor, when we construct a string builder initialized to the contents of the specified string. The initial capacity of the string builder is 16 plus the length of the string argument. So, here we have initially passed “Java” to constructor and length of it; is 4. Therefore, the initial capacity is 20. So 20 will be printed when line 6 executed.

Question 56

What will be the output of this program code?

```
1. public class Pro {
2.
3.        public static void main(String args[ ]) {
4.                char c[] = new char[]{'a','b','c'};
5.                String cd = "abcdef".substring(4);
6.                String s1 = new String(c);
7.                s1 += cd;
8.                System.out.print(s1);
9.        }
10. }
```

- A. abcef
- B. abcdef
- C. An exception will be thrown.
- D. Compilation fails due to an error at line 5.
- E. Compilation fails due to an error at line 6.

Option A is the correct answer.

Option A is correct as the variable “cd” is equal to “ef”, so “s1+cd” returns “abcef”.

Option D is incorrect as we can use a String in that manner. At line 5, calling substring method on the “abcdef” will return a string with value “ef” because the “4” is the index position of e and we haven’t specified the last index, so every letter after the index position 4 will be returned.

Option E is incorrect as we can create a string from invoking the string class constructor. There is a constructor which takes a char array, so line 6 is completely legal.

Question 57

What will be the output of this program code?

```
527.          public class Pro {
      static int x = 50;
     public final static void main(String[] args) {
          int x, y = 100;
         System.out.print(x);
      }
}
```

- A. 50
- B. 100
- C. 0
- D. An Error is thrown at the runtime, stating that, Main method not found in class Pro.
- E. Compilation fails.

Option E is the correct answer.

Option E is correct as the code fails to compile. There are three types of variables class, instance and local. Local variables (variables which are declared inside a method) must be initialized before using them. But at line 4, the statement “int x, y = 100”, only initialized the variable y. Therefore at line 5, trying to use the variable “x” before initialize, causes a compile-time error.

Other options are incorrect as the code fails to compile. However if we could initialize the variable “x” then the code would compile and output will be the value of the variable “x”. This concept is called as variable shadowing.

Question 58

What will be the output of this program code?

```
public class Pro {

     int x = y;
    static int y = 10;

     public static void main(String args[]) {
           System.out.print(new Pro().x + " ");
           System.out.print ( y );
      }

     static{ y = 15; }

}
```

- A. 10 15
- B. 10 10
- C. 15 15
- D. Compilation fails due to error at line 3.

- E. Compilation fails due to multiple errors.

Option C is the correct answer.

When class is loaded the static fields are initialized and also the static blocks are executed. So, when the class Pro is loaded, y is initialized to 10 but then the static field at line 11 changes the value of y to 15. So, the last value of the variable y is 15. Not like static variables, the instance variables initialized when objects are created. So, here the value of x is also 15. Therefore, the output is 15 15. So, option C is correct.

Options A and B are incorrect as explained above.

Option D is incorrect as there is no issue at line 3 because the variable is y is initialized before x get initialized.

Question 59

Which of the following statements are true?

(Choose two options.)

*GC – Garbage Collector

- A. Java applications never run out of memory as GC manages the memory.
- B. An object is eligible for GC when there is no live thread can reach it.
- C. The purpose of GC is to delete objects that there is no use at the moment.
- D. When you request GC to run, it will start to run immediately.
- E. Object Class has a finalize() method

Options B and E are the correct answers.

Option B is correct since an object is eligible for GC when there is no reference to an object in a currently live thread.

Option E is correct since there is a method called “finalize” in Object class, it is a special method much like the main method in java. finalize() is called before Garbage collector reclaims the Object, its last chance for any object to perform a clean up activity.

Like other any program, java applications can run out of memory. So, option A is incorrect.

Option C is incorrect as the purpose of the GC is to remove the objects which have no reference in a currently live thread.

Option D is incorrect as the JVM decide when to run GC whether we request or not.

Question 60

Which of the following set contains only primitive data types?

- A. Flaot, Byte, char, float
- B. int, Boolean, String
- C. Integer, char, double
- D. boolean, char, bit
- E. None of the above.

Option E is the correct answer.

There are only eight primitive data types boolean, char, byte, short, int, long, float, double. So, option E is correct.

Float, Byte, and Integer are not primitives. S,o options A, B, and C are incorrect.

Option D is incorrect as there is no data type called bit.

Question 61

Which of the following option is true about these statements?

I. StringBuilder sb =new StringBuilder("Java");

II. StringBuilder sb0 =new StringBuilder("Java");

III. String s =new String ("Java");

- A. s == sb
- B. sb == sb0
- C. sb.equals(sb0)
- D. s.equals(sb)
- E. sb.toString().equals(sb0.toString())
- F. None of the above.

Option E is the correct answer.

Option E is correct as calling “to String()” method on String builder objects returns strings accordingly. So, the comparison is equal to following;
“Java”equals(“Java”);

Option A is incorrect as it causes a compile time error because String and String Builder are incomparable types.

Option B is incorrect as “==” only check the both reference refer to the same object, so here sb and sb0 refer to two different objects, therefore this will return false.

Options C and D are incorrect as both result false because the StringBuilder class doesn’t override the “equals()” method.

Question 62

What will be the output of this program code?
(Choose two options)

```
class Ex1 {

    public static void main(String args[ ]) {
        int _5 = 10;
        int x = _5;
        System.out.print(_5/x);
    }

}
```

- A. 1
- B. 2
- C. We can’t use underscore at the beginning or end of a number literal.
- D. Compilation fails due to an error at line 4.
- E. Compilation fails due to an error at line 5.

Options A and C are the correct answers.

Option A is correct as the output is 1.

Option C is correct as it is illegal to use underscore at the beginning or end of a number literal.

At line 4, we have created an int variable with the name “_5” and at line 5 we have assigned the value of the variable “_5” to x. So, “_5” is not a literal with an underscore. If there is no variable declared as “_5” then line 5 would cause a compile-time error.

Question 63

Which of the following is correct about this given method signature?

void change(int y)

- A. This method returns integer
- B. This method is in the package access level
- C. This is an abstract method
- D. This method signature is invalid
- E. None of the above.

Option B is the correct answer.

The general method signature is;

[access modifier] [other keywords like static, final] [return type] [method name] [argument list].

Following is an example of such a method;

protected final int get It(int index)

The option B is correct as the given method has not specified the access modifier, so it is in default access level. (Default access level is also named as package access level.)

Option A is incorrect as the return type is declared as void, it means the method returns nothing.

Option C is incorrect as the method is not an abstract method as it is not marked with the abstract keyword.

Option D is incorrect as the method signature is correct.

Question 64

What will be the output of this program code?

```
class Ab {
    private final void meth1() {
        System.out.print("class Ab");
    }
}


class Sub extends Ab {
    private void meth1() {
        System.out.print("class Sub");
    }
}

class Ex4 {
    public static void main(String [] args) {
```

```
607.                    Ab ab = new Sub();
608.                    ab.meth1();
609.               }
610.          }
```

- A. The output will be Class Ab
- B. The output will be Class Sub
- C. Compilation fails due to a error at line 9 since final methods can't be overridden.
- D. Compilation fails due to an error at line 17.
- E. An exception will be thrown at runtime.

Option D is the correct answer.

Option D is correct as code fails to compile due to error on line 17 because, at line 17, we have tried to access the private method "meth1" of the Ab class from the Ex4 class.

Option C is incorrect since both meth1() methods of the Ab class and the Subclass can't be seen from any other class, so it is impossible to override them. So, it does not cause any overridden error due to the final.

Options A, B, and E are incorrect as the code fails to compile.

Question 65

Choose the correct method signature this satisfy this given requirement?

The method should be in package access level. It should take an int array and two int variables. It should return an int array. Name of the method should be "swap". And it can't be overridden.

- A. int[] swap(int []c, int a,b)
- B. protected int[] swap(int []c, int a, int b)
- C. final int[] swap (int []c, int a, int b)
- D. public int[] swap(int []c, final int a, final int b)
- E. package int[] swap(int []c, int a, int b)

Option C is the correct answer.

According to the given description, the method should be final as it can't be overridden and it should be in default access level, so only the method signature "final int[] swap (int []c, int a, int b)" satisfies all the requirements. Therefore, option C is correct.

Options A is incorrect as the method should be final and also the argument list declaration is invalid.

Option E is incorrect since we have used a package as an access modifier which is invalid.

Options B and D are incorrect as the method should be in default access level.

Question 66

What will be the output of this program code?

```
621.        public class Ex1 {
622.                final int j = 35;
623.
624.                public static void main(String args[]) {
625.                        char c = 'A'; //ASII value of 'A' is 65 and 'c' is 100
626.                       System.out.print((char)calc(c));
627.                }
628.
629.                static int calc(int i) {
630.                        return (i+j);
631.                }
632.        }
```

- A. 100
- B. c
- C. C
- D. An exception is thrown at the runtime.
- E. Compilation fails due to an error at line 10.

Option E is the correct answer.

Option E is correct as trying to access instance variable “j” from static method “calc” causes a compile-time error. So, line 10 will cause a compile-time error.

Other options are incorrect since the code fails to compile.

Question 67

What will be the output of this program code?

```
1. class Animal {
2.         public void eat()throws NullPointerException {
3.                 System.out.print("Animal eats");
4.         }
5. }
6.
```

```
7. class Dog extends Animal {
8.        void eat(String s)throws Exception {
9.                System.out.print("Dog eats "+ s);
10.       }
11. }
12.
13. public class TestIt {
14.         public static void main (String [ ] args) throws Exception {
15.                 Animal a= new Dog();
16.                 Dog d = (Dog)a;
17.                 a.eat();
18.                 d.eat("Meat");
19.         }
20. }
```

- A. Animal eats Dog eats Meat
- B. Animal eats Animal eats
- C. Compilation fails due to an error at line 17.
- D. An exception will be thrown at runtime.
- E. Compilation fails due to an error at line 19.

Option A is the correct answer.

Option A is correct as when we overload a method we must change the argument list and we can throw different exceptions. We can have different access modifiers. So, the overloading the eat method at line 8 doesn't cause any problem. Though the variable has Animal reference type, it is dog object and there is no issue when we cast it to be a dog object. So, the output is Animal eats Dog eats Meat.

Other options are incorrect as explained above.

Question 68

What will be the output of this program code?

```
class Pro {

        Pro(int x){ count = x;}

        static int count;

       public static void main(String[] args) {
              Pro p1 = new Pro(10);
              Pro p2 = new Pro(20);
```

```
                Pro p3 = new Pro(30);
                count = 5;
                System.out.print(p1.count + p2.count+ count);
        }
    }
```

- A. 65
- B. 15
- C. 35
- D. An exception is thrown at the runtime.
- E. Compilation fails.

Option B is the correct answer.

Option B is correct as the variable count is a static variable, so it belongs to the class and every instance share same static value. Therefore, if we change the field value then all instance share that change. So, in this code, at line 12, the value of the variable count is 5. Therefore, the result will be 15.

Other options are incorrect as explained above.

Option E is incorrect as the code compiles successfully.

Question 69

Which of the following statement is correct?

- A. Overloading methods may change the argument list.
- B. Overloading methods may throw new or broader checked exception.
- C. Overloading methods must change the return type.
- D. Overloading method must use less restrictive access level.
- E. Final methods can’t be overloaded.

Option B is the correct answer.

Option B is correct as the overloading methods can throw new or broader checked or unchecked exceptions.

Option A is incorrect as the overloading methods must change the argument list.

Option C is incorrect as the overloading methods may change the return type but it is not a must.

Option D is incorrect as the overloading method can change any access level.

Option E is incorrect as overloading method can be overloaded but not override.

Question 70

Which of the following method signature will allow the compilation to succeed when inserted at line 8?

```
1. class Ex1 {
2.
3.          public static void main(String args[ ]) {
4.                   int x = 10;
5.                   System.out.print( change(x) );
6.          }
7.
8.          //insert here
9.                   int x = c+10;
10.                  return x;
11.         }
12.
13. }
```

- A. private static int change(int c){
- B. public int change(int c){
- C. public static void change(int c){
- D. public char change(int c){
- E. public char change(){

Option A is the correct answer.

According to the method call at line 5, we could assume following things about the method we are trying to invoke;

- A method name is “change”
- A method should be static since it doesn’t use any object reference.
- A method should return something because if a method returns nothing then a compile-time error occur.
- A method should be able to receive int variable.
- A method can have any access level sincAe they are in the same class.

Option A is correct since it is the only method which satisfies all above requirements.

Options B, D, and E are incorrect as those methods are not static.

Option C is incorrect since it doesn’t return anything.

Question 71

What will be the output of this code fragment?

```
677.        class A {
        A (int i) {
            this();
            System.out.print("B ");
        }

        A ( ) {
            this(5);
           System.out.print("A "); }
    }

    class Ex6 {
          public static void main(String args[]) {
                    A a = new A();
          }
    }
```

- A. A B
- B. B A
- C. B
- D. An exception will be thrown at runtime.
- E. Compilation fails.

Option E is the correct answer.

When object creates, it must invoke the Object class constructor finally. But in this code, there is no way to call Object class constructor as the two constructors of class A have this() call, so it will cause recursive constructor invocation. So, the option E is correct.

Other options are incorrect as explained above.

Question 72

Which of the given statements are true?

I. Primitive arguments are passed into methods by value.

II. Reference data type parameters are passed into methods by value.

III. Integer, Double and Boolean are example for primitive type arguments.

- A. Only I.
- B. Only III.
- C. Only I and II.
- D. Only II and III.
- E. None of the above.

Option C is the correct answer.

Option C is correct since only the statements I and II are correct.

The statement I is correct since the primitive arguments, such as an int or a double, are passed into methods by value. This means that any changes to the values of the parameters exist only within the scope of the method. When the method returns, the parameters are gone and any changes to them are lost.

Statement II is correct since the reference data type parameters, such as objects, are also passed into methods by value. This means that when the method returns, the passed-in reference still references the same object as before. However, the values of the object's fields can be changed in the method, if they have the proper access level.

Statement III is incorrect since Integer, Double and Boolean are wrappers, so they are not primitive arguments.

Question 73

What will be the output of this code fragment?

```
class Ex1 {
    public static void main(String args[]) {
         int [ ] x = {1,2,3,4,5};
        nxt (x);
        for(int num : x)
        System.out.print(num+" ");
    }
    static void nxt(Object o) {
        int [ ] y = ( int [ ] ) o;
        for( int i = 5, j = 0; i > 0; --i , j++ ) {
            y [ j ] = i;
        }
    }
}
```

- A. 1 2 3 4 5
- B. 1 2 3 4
- C. 5 4 3 2 1
- D. 5 4 3 2
- E. Compilation fails.

Option C is the correct answer.

Option C is correct. Arrays are also the objects; so when they pass to a method reference to the object is passed; therefore the argument variable" y" and local variable “x” refers to the same array object. In the method “nxt”, for loop reverses the order of the array. The final result will be 5 4 3 2 1.

Other Options are incorrect as explained above.

Option E is incorrect as the code compiles successfully.

Question 74

Consider this given code -

```
1. class En {
2.        public int i;
3.        public char c;
4.        public static void main(String [ ] args) {
5.                //codes
6.        }
7.       private int getI() {
8.                return i;
9.       }
10. }
```

Which of the following can be considered as true about the above code?

I. This code hasn’t correctly implemented the encapsulation principals.

II. Ths “getI” method at line7, has correctly implemented the encapsulation principals but not the variables.

III. Variables in this code have correctly implemented the encapsulation principals but not the “getI” method.

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

Option A is the correct answer.

Statement I is correct as this code completely disagrees with encapsulation principals.

Statement II and III are incorrect as both the methods and variables in this class doesn’t implement the encapsulation principals. Here all variables are declared as public which allows anyone to modify variables without using methods and also methods have declared to be private which should have declared as public.

Option A is correct since only the statement I is correct.

Question 75

Which of the following statement is correct?

- A. Even if we provide a constructor to a class, a default constructor will be created.
- B. The default constructor has no statement.
- C. The default constructor is public.
- D. The default constructors have same name as that of the class name but user-defined constructors may have different names.
- E. User defined constructor can have any access level including private.

Option E is the correct answer.

Option E is correct as a constructor can have any access level, private, protected, default, and public.

Option A is incorrect as the compiler provides a default constructor when we fail to specify a constructor but if we explicitly provide one then the default constructor will not present.

Option B is incorrect as default constructor has “super()” statement.

Option C is incorrect as the access level of the default constructor is will take same access level of the class declaration, if the class is public then default constructor will be public too.

Option D is incorrect as any constructor, no matter user defined or default, all must have the same name as class name.

Question 76

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

```
package packA;
public class Ab {
        public int x = 1;
        protected int y = 2;
        int z = 3;
}

package packB;
import packA.*;
class Pro {
          public static void main(String [] args) {
                    Ab ab = new Ab();
                   System.out.print(ab.x);
                   System.out.print(ab.y);
                   System.out.print(ab.z);
          }
}
```

- A. 123
- B. Compilation fails due to an error at line 13.
- C. Compilation fails due to an error at line 14.
- D. Compilation fails due to an error at line 15.
- E. Compilation succeeds.

Options C and D are the correct answers.

Option C is correct as the code fails to compile because the variable “y” has the protected access level, so it can’t be accessed from another package by using an object but it is legal to access it through inheritance. Therefore, line 14 causes a compile-time error.

Option D is correct as the code fails to compile because the variable “z” has default access level, so it can’t be accessed from another package, therefore line 15 causes a compile-time error.

Other options are incorrect as explained above.

Question 77

Which of the following will produce the output as “Kitty unknown” when inserted at line 2?

```
1. class Cat extends Animal {
2.          //insert here
3.        public static void main(String[] args) {
4.                  new Cat("Kitty");
5.                  new Cat();
6.        }
7. }
8.
9.  class Animal {
10.         Animal(String s) { System.out.print(s + " "); }
11.  }
```

- A. Cat(String s){super(s);} Cat(){this("unknown");}
- B. Cat(String s){this(s);} Cat(){super("unknown");}
- C. Cat(String s){super(s);}
- D. Cat(){super("unknown");} Cat(String s){this();}

Option A is the correct answer.

Option A is correct since it produces the expected output. By looking at lines 4 and 5, we can say that there should be two constructors for the Cat class; one takes String and other nothing. By looking at line 10, we can see that Animal class has only one constructor which can take a String as the parameter, so we can assume that there should be a super call which passes a string in the class Cat.

Option B is incorrect as "this(s)" fails since there is no other constructor which can take a string.

Option C is incorrect as there should be a constructor with empty parameter list, otherwise line 5 would cause a compile time error.

Option D is incorrect as it doesn't produce expected output.

Question 78

Which of the following represents the correct order of the most restrictive to least restrictive?

- A. private, protected, default, public
- B. public, protected, default, private
- C. protected, private, default, public
- D. private, default, protected, public
- E. private, public, default, protected

Option D is the correct answer.

The private access level is the most restrictive access level. It limits the accessibility to the class.

The default access level is the second most restrictive access level. It limits the accessibility to the package.

The protected access level is the second least restrictive access level. It provides the accessibility inter packages through inheritance.

The public access level is the least restrictive access level, it provides global accessibility.

So, the correct option is D.

Question 79

What will be the output of this code fragment?

```
class Animal {
    protected void makeSound() {
        System.out.print("Genaric Sound");
    }
}

class Dog extends Animal {
    void makeSound(String s) {
        System.out.print(s);
    }
}

class Pro {
    public static void main(String[] args) {
        Animal dog = new Dog();
        dog.makeSound("bark");
    }
}
```

- A. Genaric Sound
- B. bark
- C. Genaric Sound bark
- D. An exception will be thrown at runtime.
- E. Compilation fails.

Option E is the correct answer.

Option E is incorrect as the code fails to compile. The reference type determines which overloaded version is selected, it happens at compile time. At line 15 we have created a Dog object and it is assigned to an Animal reference. So, at the compile time, the compiler fails to find the “make Sound(String s)” method, so compilation fails.

Other options are incorrect as the code fails to compile.

Question 80

This given code is an example of -

```
class School {
    Student s;
    Teacher t;
```

```
        //lots of codes here
}
```

- A. Tightly coupling.
- B. High cohesion.
- C. Loose coupling.
- D. Low cohesion.
- E. None of the above.

Option A is the correct answer.

The coupling can be identified by how much change in another class would bring a change to this class. It should be as less as possible. The Student and Teacher classes in the code have their own classes but these two classes make the School class depend on them and they are not declared as private. We can’t say anything about cohesion here, as we are not given other codes of School.