Top of Form

Question 1: **Correct**

Given code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. /\*INSERT\*/
6. arr[1] = 5;
7. arr[2] = 10;
8. System.out.println("[" + arr[1] + ", " + arr[2] + "]"); //Line n1
9. }
10. }

And below statements:

1. short arr [] = new short[2];

2. byte [] arr = new byte[10];

3. short [] arr; arr = new short[3];

4. short [2] arr;

5. short [3] arr;

6. int [] arr = new int[]{100, 100};

7. int [] arr = new int[]{0, 0, 0, 0};

8. short [] arr = {};

9. short [] arr = new short[2]{5, 10};

How many above statements can be used to replace /\*INSERT\*/, such that on execution, code will print [5, 10] on to the console?

* 

**Only one option**

* 

**Only three options**

**(Correct)**

* 

**Only four options**

* 

**None of the given options**

* 

**More than four options**

* 

**Only two options**

**Explanation**

Let's check all the statements one by one:

1. short arr [] = new short[2]; => ✗

You can declare one-dimensional array by using either "short arr []" or "short [] arr". 'arr' refers to a short array object of 2 elements. arr[2] will throw ArrayIndexOutOfBoundsException at runtime.

2. byte [] arr = new byte[10]; => ✓

'arr' refers to a byte array object of 10 elements, where 0 is assigned to each array element. But later on element at 1st and 2nd indexes have been re-initialized. Line n1 successfully prints [5, 10] on to the console.

3. short [] arr; arr = new short[3]; => ✓

You can create an array object in the same statement or next statement. 'arr' refers to a short array object of 3 elements, where 0 is assigned to each array element. Later on element at 1st and 2nd indexes have been re-initialized. Line n1 successfully prints [5, 10] on to the console.

4. short [2] arr; ✗

Array size cannot be specified at the time of declaration, so short [2] arr; causes compilation error.

5. short [3] arr; ✗

Array size cannot be specified at the time of declaration, so short [3] arr; causes compilation error.

6. int [] arr = new int[]{100, 100}; => ✗

'arr' refers to an int array object of size 2 and both array elements have value 100. arr[2] will throw ArrayIndexOutOfBoundsException at runtime.

7. int [] arr = new int[]{0, 0, 0, 0}; => ✓

'arr' refers to an int array object of size 4 and all array elements have value 0. Later on element at 1st and 2nd indexes have been re-initialized. Line n1 successfully prints [5, 10] on to the console.

8. short [] arr = {}; => ✗

'arr' refers to a short array object of 0 size. so arr[1] will throw ArrayIndexOutOfBoundsException at runtime.

9. short [] arr = new short[2]{5, 10}; => ✗

Array's size can't be specified, if you use {} to assign values to array elements.

Hence, out of the given 9 statements, only 3 will print [5, 10] on to the console.

Bottom of Form

Top of Form

Question 2: **Correct**

Which of the following statement is correct about below code?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. do {
6. System.out.println(100);
7. } while (false);
8. System.out.println("Bye");
9. }
10. }

* 

**Compiles successfully and prints "Bye"**

* 

**Unreachable code compilation error**

* 

**Compiles successfully and prints 100 in infinite loop**

* 

**100  
Bye**

**(Correct)**

**Explanation**

As do-while loop executes at least once, hence none of the code is unreachable in this case.

Java runtime prints 100 to the console, then it checks boolean expression, which is false.

Hence control goes out of do-while block. Java runtime executes 2nd System.out.println statement to print "Bye" on to the console.

Bottom of Form

Top of Form

Question 3: **Correct**

Below is the code of Test.java file:

1. package com.udayan.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String [] args) {
8. List<Integer> list = new ArrayList<Integer>();
9. list.add(new Integer(2));
10. list.add(new Integer(1));
11. list.add(new Integer(0));
13. list.remove(list.indexOf(0));
15. System.out.println(list);
16. }
17. }

What will be the result of compiling and executing Test class?

* 

**Compilation error**

* 

**[1, 0]**

* 

**[2, 1]**

**(Correct)**

* 

**An exception is thrown at runtime**

**Explanation**

remove method of List interface is overloaded: remove(int) and remove(Object).

indexOf method accepts argument of Object type, in this case list.indexOf(0) => 0 is auto-boxed to Integer object so no issues with indexOf code. list.indexOf(0) returns 2 (index at which 0 is stored in the list). So list.remove(list.indexOf(0)); is converted to list.remove(2);

remove(int) version is matched, it's a direct match so compiler doesn't do auto-boxing in this case. list.remove(2) removes the element at index 2, which is 0.

Hence in the output, you get [2, 1].

Bottom of Form

Top of Form

Question 4: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.Iterator;
6. import java.util.List;
8. public class Test {
9. public static void main(String[] args) {
10. List<String> dryFruits = new ArrayList<>();
11. dryFruits.add("Walnut");
12. dryFruits.add("Apricot");
13. dryFruits.add("Almond");
14. dryFruits.add("Date");
16. Iterator<String> iterator = dryFruits.iterator();
17. while(iterator.hasNext()) {
18. String dryFruit = iterator.next();
19. if(dryFruit.startsWith("A")) {
20. dryFruits.remove(dryFruit);
21. }
22. }
24. System.out.println(dryFruits);
25. }
26. }

What will be the result of compiling and executing Test class?

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**Compilation error**

* 

**[Walnut, Apricot, Almond, Date]**

* 

**[Walnut, Date]**

**Explanation**

ConcurrentModificationException exception may be thrown for following condition:

1. Collection is being iterated using Iterator/ListIterator or by using for-each loop.

And

2. Execution of Iterator.next(), Iterator.remove(), ListIterator.previous(), ListIterator.set(E) & ListIterator.add(E) methods. These methods may throw java.util.ConcurrentModificationException in case Collection had been modified by means other than the iterator itself, such as Collection.add(E) or Collection.remove(Object) or List.remove(int) etc.

For the given code, 'dryFruits' list is being iterated using the Iterator<String>.

hasNext() method of Iterator has following implementation:

1. public boolean hasNext() {
2. return cursor != size;
3. }

Where cursor is the index of next element to return and initially it is 0.

1st Iteration: cursor = 0, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns "Walnut".

2nd Iteration: cursor = 1, size = 4, hasNext() returns true. iterator.next() increments the cursor by 1 and returns "Apricot". As "Apricot" starts with "A", hence dryFruits.remove(dryFruit) removes "Apricot" from the list and hence reducing the list's size by 1, size becomes 3.

3rd Iteration: cursor = 2, size = 3, hasNext() returns true. iterator.next() method throws java.util.ConcurrentModificationException.

If you want to remove the items from ArrayList, while using Iterator or ListIterator, then use Iterator.remove() or ListIterator.remove() method and NOT List.remove(...) method. Using List.remove(...) method while iterating the list (using the Iterator/ListIterator or for-each) may throw java.util.ConcurrentModificationException.

Bottom of Form

Top of Form

Question 5: **Correct**

How many objects of Pen class are eligible for Garbage Collection at Line 4?

1. package com.udayan.oca;
3. class Pen {
5. }
7. public class TestPen {
8. public static void main(String[] args) {
9. new Pen(); //Line 1
10. Pen p = new Pen(); // Line 2
11. change(p); //Line 3
12. System.out.println("About to end."); //Line 4
13. }
15. public static void change(Pen pen) { //Line 5
16. pen = new Pen(); //Line 6
17. }
18. }

* 

**1**

* 

**3**

* 

**2**

**(Correct)**

* 

**0**

**Explanation**

Object created at Line 1 becomes eligible for Garbage collection after Line 1 only, as there are no references to it. So We have one object marked for GC.

Object created at Line 6 becomes unreachable after change(Pen) method pops out of the STACK, and this happens after Line 3.

So at Line 4, we have two Pen objects eligible for Garbage collection: Created at Line 1 and Created at Line 6.

Bottom of Form

Top of Form

Question 6: **Correct**

Which of the following correctly defines class Printer?

* 
  1. /\* Java Developer Comments. \*/
  2. package com.udayan.oca;
  3. public class Printer {
  5. }

**(Correct)**

* 
  1. public class Printer {
  3. }
  4. package com.udayan.oca;
* 
  1. public class Printer {
  2. package com.udayan.oca;
  3. }
* 
  1. import java.util.\*;
  2. package com.udayan.oca;
  3. public class Printer {
  5. }

**Explanation**

If package is used then it should be the first statement, but javadoc and developer comments are not considered as java statements so a class can have developer and javadoc comments before the package statement.

If import and package both are available, then correct order is package, import, class declaration.

Bottom of Form

Top of Form

Question 7: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. try {
6. main(args);
7. } catch (Exception ex) {
8. System.out.println("CATCH-");
9. }
10. System.out.println("OUT");
11. }
12. }

* 

**OUT**

* 

**CATCH-OUT**

* 

**None of the System.out.println statements are executed**

**(Correct)**

* 

**Compilation error**

**(Incorrect)**

**Explanation**

main(args) method is invoked recursively without specifying any exit condition, so this code ultimately throws java.lang.StackOverflowError. StackOverflowError is a subclass of Error type and not Exception type, hence it is not handled. Stack trace is printed to the console and program ends abruptly.

Java doesn't allow to catch specific checked exceptions if these are not thrown by the statements inside try block.

catch(java.io.FileNotFoundException ex) {} will cause compilation error in this case as main(args); will never throw FileNotFoundException. But Java allows to catch Exception type, hence catch (Exception ex) {} doesn't cause any compilation error.

Bottom of Form

Top of Form

Question 8: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. Double [] arr = new Double[2];
6. System.out.println(arr[0] + arr[1]);
7. }
8. }

* 

**NullPointerException is thrown at runtime**

**(Correct)**

* 

**0.0**

* 

**Compilation error**

* 

**ClassCastException is thrown at runtime**

**Explanation**

Array elements are initialized to their default values. arr is referring to an array of Double type, which is reference type and hence both the array elements are initialized to null.

To calculate arr[0] + arr[1], java runtime converts the expression to arr[0].doubleValue() + arr[1].doubleValue(). As arr[0] and arr[1] are null hence calling doubleValue() method throws NullPointerException.

Bottom of Form

Top of Form

Question 9: **Correct**

What will be the result of compiling and executing Test class?

1. //Test.java
2. package com.udayan.oca;
4. class Student {
5. String name;
6. int marks;
8. Student(String name, int marks) {
9. this.name = name;
10. this.marks = marks;
11. }
12. }
14. public class Test {
15. public static void main(String[] args) {
16. Student student = new Student("James", 25);
17. int marks = 25;
18. review(student, marks);
19. System.out.println(marks + "-" + student.marks);
20. }
22. private static void review(Student stud, int marks) {
23. marks = marks + 10;
24. stud.marks+=marks;
25. }
26. }

* 

**35-25**

* 

**25-25**

* 

**35-60**

* 

**25-60**

**(Correct)**

**Explanation**

In below statements: student<main> means student inside main method.

On execution of main method: student<main> --> {"James", 25}, marks<main> = 25.

On execution of review method: stud<review> --> {"James", 25} (same object referred by student<main>), marks<review> = 25 (this marks is different from the marks defined in main method). marks<review> = 35 and stud.marks = 60. So at the end of review method: stud<review> --> {"James", 60}, marks<review> = 35.

Control goes back to main method: student<main> --> {"James", 60}, marks<main> = 25. Changes done to reference variable are visible in main method but changes done to primitive variable are not reflected in main method.

Bottom of Form

Top of Form

Question 10: **Correct**

Consider codes below:

1. //A.java
2. package com.udayan.oca;
4. public class A {
5. public void print() {
6. System.out.println("A");
7. }
8. }
9. //B.java
10. package com.udayan.oca;
12. public class B extends A {
13. public void print() {
14. System.out.println("B");
15. }
16. }
17. //Test.java
18. package com.udayan.oca.test;
20. import com.udayan.oca.\*;
22. public class Test {
23. public static void main(String[] args) {
24. A obj1 = new A();
25. B obj2 = (B)obj1;
26. obj2.print();
27. }
28. }

What will be the result of compiling and executing Test class?

* 

**B**

* 

**ClassCastException is thrown at runtime**

**(Correct)**

* 

**Compilation error**

* 

**A**

**Explanation**

Class A and B are declared public and inside same package com.udayan.oca. Method print() of class A has correctly been overridden by B.

print() method is public so no issues in accessing it anywhere.

Let's check the code inside main method.

A obj1 = new A(); => obj1 refers to an instance of class A.

B obj2 = (B)obj1; => obj1 is of type A and it is assigned to obj2 (B type), hence explicit casting is necessary. obj1 refers to an instance of class A, so at runtime obj2 will also refer to an instance of class A. sub type can't refer to an object of super type so at runtime B obj2 = (B)obj1; will throw ClassCastException.

Bottom of Form

Top of Form

Question 11: **Incorrect**

For the given code snippet:

List<String> list = new /\*INSERT\*/();

Which of the following options, if used to replace /\*INSERT\*/, compiles successfully?

Select 2 options.

* 

**ArrayList<>**

**(Correct)**

* 

**List<String>**

**(Incorrect)**

* 

**ArrayList<String>**

**(Correct)**

* 

**List<>**

**(Incorrect)**

**Explanation**

List is an interface so its instance can't be created using new keyword. List<String> and List<> will cause compilation failure.

ArrayList implements List interface, so it can be it can be used to replace /\*INSERT\*/. List<String> list = new ArrayList<String>(); compiles successfully.

Starting with JDK 7, Java allows to not specify type while initializing the ArrayList. Type is inferred from the left side of the statement.

So List<String> list = new ArrayList<>(); is a valid syntax starting with JDK 7.

Bottom of Form

Top of Form

Question 12: **Incorrect**

Given code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String [] arr = {"I", "N", "S", "E", "R", "T"};
6. for(/\*INSERT\*/) {
7. if (n % 2 == 0) {
8. continue;
9. }
10. System.out.print(arr[n]); //Line n1
11. }
12. }
13. }

And below options:

1. int n = 0; n < arr.length; n += 1

2. int n = 0; n <= arr.length; n += 1

3. int n = 1; n < arr.length; n += 2

4. int n = 1; n <= arr.length; n += 2

How many above options can be used to replace /\*INSERT\*/, such that on execution, code will print NET on to the console?

* 

**Only one option**

**(Incorrect)**

* 

**Only three options**

* 

**All four options**

**(Correct)**

* 

**Only two options**

* 

**None of the other options**

**Explanation**

From the given array, if you print the elements at 1st, 3rd and 5th indexes, then you will get expected output.

Also note that, for values of n = 0, 2, 4, 6; Line n1 would not be executed, which means even if the value of n is 6, above code will not throw ArrayIndexOutOfBoundsException.

For 1st option [int n = 0; n < arr.length; n += 1], values of n used: 0, 1, 2, 3, 4, 5 and because of continue; statement, Line n1 will not execute for 0, 2 & 4 and it will execute only for 1, 3 & 5 and therefore NET will be printed.

For 2nd option [int n = 0; n <= arr.length; n += 1], values of n used: 0, 1, 2, 3, 4, 5, 6 and because of continue; statement, Line n1 will not execute for 0, 2, 4 & 6 and it will execute only for 1, 3 & 5 and therefore NET will be printed.

For 3rd option [int n = 1; n < arr.length; n += 2], values of n used: 1, 3, 5 and therefore NET will be printed.

For 4th option [int n = 1; n <= arr.length; n += 2], values of n used: 1, 3, 5 and therefore NET will be printed.

Hence, all the 4 options are valid.

Bottom of Form

Top of Form

Question 13: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. System.out.println("Output is: " + 10 != 5);
6. }
7. }

* 

**Output is: true**

* 

**Output is: 10 != 5**

* 

**Output is: false**

**(Incorrect)**

* 

**Compilation error**

**(Correct)**

**Explanation**

Binary plus (+) has got higher precedence than != operator. Let us group the expression.

"Output is: " + 10 != 5

= ("Output is: " + 10) != 5

[!= is binary operator, so we have to evaluate the left side first. + operator behaves as concatenation operator.]

= "Output is: 10" != 5

Left side of above expression is String, and right side is int. But String can't be compared to int, hence compilation error.

Bottom of Form

Top of Form

Question 14: **Correct**

Consider below code:

1. public class Test {
2. static {
3. System.out.println(1/0);
4. }
6. public static void main(String[] args) {
7. System.out.println("HELLO");
8. }
9. }

On execution, does Test class print "HELLO" on to the console?

* 

**No, HELLO is not printed on the console**

**(Correct)**

* 

**Yes, HELLO is printed on to the console**

**Explanation**

To invoke the special main method, JVM loads the class in the memory. At that time, static initializer block is invoked. 1/0 throws a RuntimeException and as a result static initializer block throws an instance of java.lang.ExceptionInInitializerError.

Bottom of Form

Top of Form

Question 15: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. import java.io.FileNotFoundException;
4. import java.io.IOException;
6. abstract class Super {
7. public abstract void m1() throws IOException;
8. }
10. class Sub extends Super {
11. @Override
12. public void m1() throws IOException {
13. throw new FileNotFoundException();
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. Super s = new Sub();
20. try {
21. s.m1();
22. } catch (FileNotFoundException e) {
23. System.out.print("M");
24. } finally {
25. System.out.print("N");
26. }
27. }
28. }

* 

**N**

* 

**Compilation error**

**(Correct)**

* 

**MN**

* 

**Program ends abruptly**

**Explanation**

Even though an instance of FileNotFoundException is thrown by method m1() at runtime, but method m1() declares to throw IOException.

Reference variable s is of Super type and hence for compiler, call to s.m1(); is to method m1() of Super, which throws IOException.

And as IOException is checked exception hence calling code should handle it.

As calling code doesn't handle IOException or its super type, so s.m1(); gives compilation error.

Bottom of Form

Top of Form

Question 16: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int x = 1;
6. while(checkAndIncrement(x)) {
7. System.out.println(x);
8. }
9. }
11. private static boolean checkAndIncrement(int x) {
12. if(x < 5) {
13. x++;
14. return true;
15. } else {
16. return false;
17. }
18. }
19. }

* 

**1  
2  
3  
4  
5**

* 

**2  
3  
4  
5**

* 

**1  
2  
3  
4**

**(Incorrect)**

* 

**Infinite loop**

**(Correct)**

**Explanation**

x of checkAndIncrement method contains the copy of variable x defined in main method. So, changes done to x in checkAndIncrement method are not reflected in the variable x of main. x of main remains 1 as code inside main is not changing its value.

Every time checkAndIncrement method is invoked with argument value 1, so true is returned always and hence while loop executes indefinitely.

Bottom of Form

Top of Form

Question 17: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
6. class Counter {
7. int count;
8. Counter(int count) {
9. this.count = count;
10. }
12. public String toString() {
13. return "Counter-" + count;
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. ArrayList<Counter> original = new ArrayList<>();
20. original.add(new Counter(10));
22. ArrayList<Counter> cloned = (ArrayList<Counter>) original.clone();
23. cloned.get(0).count = 5;
25. System.out.println(original);
26. }
27. }

What will be the result of compiling and executing Test class?

* 

**[Counter-10]**

* 

**An exception is thrown at runtime**

* 

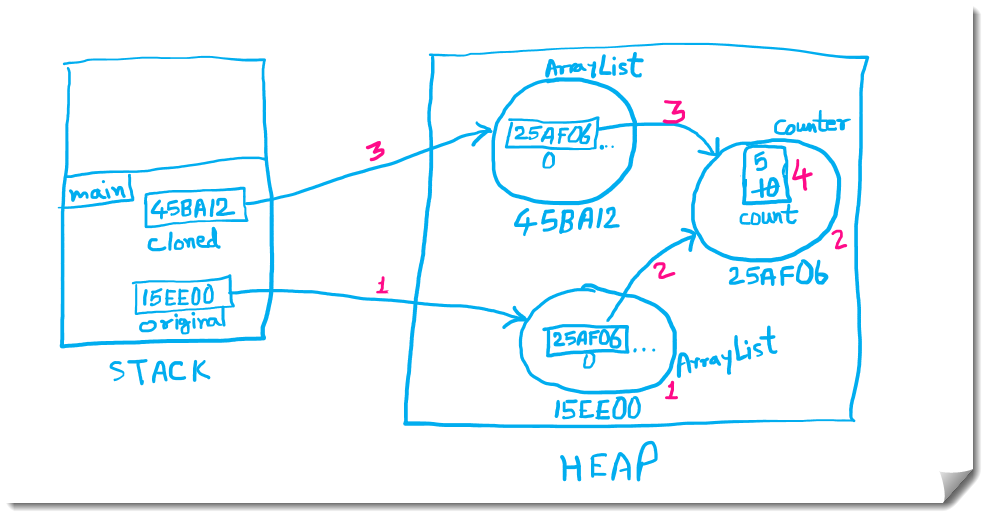
**[Counter-5]**

**(Correct)**

* 

**Compilation error**

**Explanation**



Let's see what is happening during execution:

main(String [] args) method goes on to the top of the STACK.

1. ArrayList<Counter> original = new ArrayList<>(); => It creates an ArrayList object [suppose at memory location 15EE00] and variable 'original' refers to it.

2. original.add(new Counter(10)); => It creates a Counter object [suppose at memory location 25AF06] and adds it as a first element of the ArrayList. This means element at 0th index of the ArrayList instance refers to Counter object at the memory location 25AF06.

3. ArrayList<Counter> cloned = (ArrayList<Counter>) original.clone(); => original.clone() creates a new array list object, [suppose at memory location 45BA12] and then it will copy the contents of the ArrayList object stored at [15EE00]. So, cloned contains memory address of the same Counter object.

In this case, original != cloned, but original.get(0) == cloned.get(0). This means both the array lists are created at different memory location but refer to same Counter object.

4. cloned.get(0).count = 5; => cloned.get(0) returns the Counter object stored at the memory location 25AF06 and .count = 5 means change the value of count variable of the Counter object (stored at memory location 25AF06) to 5.

5. System.out.println(original); Prints the element of ArrayList original, which is: {25AF06} and toString() method prints: [Counter-5] as Counter object referred by [25AF06] is [Counter object (5)].

Bottom of Form

Top of Form

Question 18: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. char var1;
5. double var2;
6. float var3;
8. public static void main(String[] args) {
9. Test obj = new Test();
10. System.out.println(">" + obj.var1);
11. System.out.println(">" + obj.var2);
12. System.out.println(">" + obj.var3);
13. }
14. }

* 
  1. >null
  2. >0.0
  3. >0.0f
* 
  1. >
  2. >0.0
  3. >0.0f
* 
  1. >
  2. >0.0
  3. >0.0

**(Correct)**

* 
  1. >null
  2. >0.0
  3. >0.0

**Explanation**

Primitive type instance variables are initialized to respective zeros (byte: 0, short: 0, int: 0, long: 0L, float: 0.0f, double: 0.0, boolean: false, char: \u0000).

When printed on the console; byte, short, int & long prints 0, float & double print 0.0, boolean prints false and char prints nothing or non-printable character (whitespace).

Reference type instance variables are initialized to null.

Bottom of Form

Top of Form

Question 19: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
6. public class Test {
7. public static void main(String [] args) {
8. LocalDate newYear = LocalDate.of(2018, 1, 1);
9. LocalDate christmas = LocalDate.of(2018, 12, 25);
10. boolean flag1 = newYear.isAfter(christmas);
11. boolean flag2 = newYear.isBefore(christmas);
12. System.out.println(flag1 + ":" + flag2);
13. }
14. }

What will be the result of compiling and executing Test class?

* 

**Compilation error**

* 

**true:false**

* 

**An exception is thrown at runtime**

* 

**false:true**

**(Correct)**

**Explanation**

isAfter and isBefore method can be interpreted as:

Does 1st Jan 2018 come after 25th Dec 2018? No, false.

Does 1st Jan 2018 come before 25th Dec 2018? Yes, true.

Bottom of Form

Top of Form

Question 20: **Correct**

Consider the code of Test.java file:

1. package com.udayan.oca;
3. class Student {
4. String name;
5. int age;
7. void Student() {
8. Student("James", 25);
9. }
11. void Student(String name, int age) {
12. this.name = name;
13. this.age = age;
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. Student s = new Student();
20. System.out.println(s.name + ":" + s.age);
21. }
22. }

What will be the result of compiling and executing Test class?

* 

**null:0**

**(Correct)**

* 

**James:25**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

Methods can have same name as the class. Student() and Student(String, int) are methods and not constructors of the class, note the void return type of these methods.

As no constructors are provided in the Student class, java compiler adds default no-arg constructor. That is why the statement Student s = new Student(); doesn't give any compilation error.

Default values are assigned to instance variables, hence null is assigned to name and 0 is assigned to age.

In the output, null:0 is displayed.

Bottom of Form

Top of Form

Question 21: **Incorrect**

Consider code of Test.java file:

1. package com.udayan.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<Character> list = new ArrayList<>();
9. list.add(0, 'V');
10. list.add('T');
11. list.add(1, 'E');
12. list.add(3, 'O');
14. if(list.contains('O')) {
15. list.remove('O');
16. }
18. for(char ch : list) {
19. System.out.print(ch);
20. }
21. }
22. }

What will be the result of compiling and executing Test class?

* 

**Runtime exception**

**(Correct)**

* 

**VETO**

**(Incorrect)**

* 

**Compilation error**

* 

**VET**

* 

**VTEO**

* 

**VTE**

**Explanation**

list.add(0, 'V'); => char 'V' is converted to Character object and stored as the first element in the list. list --> [V].

list.add('T'); => char 'T' is auto-boxed to Character object and stored at the end of the list. list --> [V,T].

list.add(1, 'E'); => char 'E' is auto-boxed to Character object and inserted at index 1 of the list, this shifts T to the right. list --> [V,E,T].

list.add(3, 'O'); => char 'O' is auto-boxed to Character object and added at index 3 of the list. list --> [V,E,T,O].

list.contains('O') => char 'O' is auto-boxed to Character object and as Character class overrides equals(String) method this expression returns true. Control goes inside if-block and executes: list.remove('O');.

remove method is overloaded: remove(int) and remove(Object). char can be easily assigned to int so compiler tags remove(int) method. list.remove(<ASCCI value of 'O'>); ASCCI value of 'A' is 65 (this everybody knows) so ASCII value of 'O' will be more than 65.

list.remove('O') throws runtime exception, as it tries to remove an item from the index greater than 65 but allowed index is 0 to 3 only.

Bottom of Form

Top of Form

Question 22: **Incorrect**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.Period;
6. public class Test {
7. public static void main(String [] args) {
8. Period period = Period.of(0, 0, 0);
9. System.out.println(period);
10. }
11. }

What will be the result of compiling and executing Test class?

* 

**p0y0m0d**

* 

**p0d**

* 

**P0Y0M0D**

**(Incorrect)**

* 

**P0D**

**(Correct)**

**Explanation**

Period.of(0, 0, 0); is equivalent to Period.ZERO. ZERO period is displayed as P0D, other than that, Period components (year, month, day) with 0 values are ignored.

toString()'s result starts with P, and for non-zero year, Y is appended; for non-zero month, M is appended; and for non-zero day, D is appended. P,Y,M and D are in upper case.

NOTE: Period.parse(CharSequence) method accepts the String parameter in "PnYnMnD" format, over here P,Y,M and D can be in any case.

Bottom of Form

Top of Form

Question 23: **Incorrect**

Consider code below:

1. package com.udayan.oca;
3. class PenDrive {
4. int capacity;
5. PenDrive(int capacity) {
6. this.capacity = capacity;
7. }
8. }
10. class OTG extends PenDrive {
11. String type;
12. String make;
13. OTG(int capacity, String type) {
14. /\*INSERT-1\*/
15. }
16. OTG(String make) {
17. /\*INSERT-2\*/
18. this.make = make;
19. }
20. }
22. public class Test {
23. public static void main(String[] args) {
24. OTG obj = new OTG(128, "TYPE-C");
25. System.out.println(obj.capacity + ":" + obj.type);
26. }
27. }

Currently above code causes compilation error.

Which of the options can successfully print 128:TYPE-C on to the console?

* 

**Replace /\*INSERT-1\*/ with:**

**super.capacity = capacity;**

**this.type = type;**

**Replace /\*INSERT-2\*/ with:**

**super(128);**

* 

**Replace /\*INSERT-1\*/ with:**

**this.type = type;**

**super(capacity);**

**Replace /\*INSERT-2\*/ with:**

**super(128);**

* 

**Replace /\*INSERT-1\*/ with:**

**super(capacity);**

**Replace /\*INSERT-2\*/ with:**

**super(128);**

**(Incorrect)**

* 

**None of the other options**

* 

**Replace /\*INSERT-1\*/ with:**

**super(capacity);**

**this.type = type;**

**Replace /\*INSERT-2\*/ with:**

**super(0);**

**(Correct)**

**Explanation**

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.

Compiler adds super(); as the first line in OTG's constructor: OTG(int capacity, String type) { super(); } but PenDrive class doesn't have a no-arg constructor and that is why OTG's constructor causes compilation error.

For the same reason, OTG(String make) constructor also causes compilation error.

To correct these compilation errors, parent class constructor should be invoked by using super(int); This would resolve compilation error.

Remember: Constructor call using this(...) or super(...) must be the first statement inside the constructor.

In the main(String[]) method, OTG(int, String) constructor is invoked, which means, we OTG(String) constructor will not be executed. So, to solve the complilation error in OTG(String) constructor, super(0); or super(128); both will work and these will not affect the expected output.

We have to make changes in OTG(int, String) constructor such that on execution, output is 128:TYPE-C.

super(capacity); will only assign value to capacity property, to assign value to type another statement is needed.

this.type = type; must be the 2nd statement.

So, /\*INSERT-1\*/ must be replaced with:

super(capacity);

this.type = type;

Bottom of Form

Top of Form

Question 24: **Correct**

Which of the following is a checked Exception?

* 

**RuntimeException**

* 

**ExceptionInInitializerError**

* 

**ClassCastException**

* 

**FileNotFoundException**

**(Correct)**

**Explanation**

ClassCastException extends RuntimeException (unchecked exception),

FileNotFoundException extends IOException, IOException extends Exception (checked exception),

ExceptionInInitializerError is from Error family and is thrown by an static initializer block,

RuntimeException and all its sub classes are unchecked exceptions.

Bottom of Form

Top of Form

Question 25: **Incorrect**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDateTime;
6. public class Test {
7. public static void main(String [] args) {
8. LocalDateTime obj = LocalDateTime.now();
9. System.out.println(obj.getSecond());
10. }
11. }

Which of the following statement is correct?

* 

**Code compiles successfully but throws Runtime exception**

* 

**It will print any int value between 0 and 59**

**(Correct)**

* 

**Code fails to compile**

* 

**It will print any int value between 1 and 60**

**(Incorrect)**

**Explanation**

LocalDateTime stores both date and time parts. LocalDateTime.now(); retrieves the current date and time from the system clock. obj.getSecond() can return any value between 0 and 59.

Bottom of Form

Top of Form

Question 26: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. import java.util.function.Predicate;
5. public class Test {
6. public static void main(String[] args) {
7. String [] arr = {"A", "ab", "bab", "Aa", "bb", "baba", "aba", "Abab"};
9. Predicate<String> p = s -> s.toUpperCase().substring(0,1).equals("A");
11. processStringArray(arr, p);
12. }
14. private static void processStringArray(String [] arr,
15. Predicate<String> predicate) {
16. for(String str : arr) {
17. if(predicate.test(str)) {
18. System.out.println(str);
19. }
20. }
21. }
22. }

* 

**A  
Aa  
Abab**

**(Incorrect)**

* 

**Compilation error**

* 

**Runtime exception**

* 

**ab  
aba**

* 

**A  
ab  
Aa  
aba  
Abab**

**(Correct)**

**Explanation**

Let us suppose test string is "aba".

Lambda expression s.toUpperCase().substring(0,1).equals("A"); means: "aba".toUpperCase().substring(0,1).equals("A"); => "ABA".substring(0,1).equals("A"); => "A".equals("A"); => true.

This lambda expression returns true for any string starting with a (in lower or upper case).  Based on the lambda expression, 5 array elements passes the Predicate's test and are printed on to the console.

Bottom of Form

Top of Form

Question 27: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. import java.time.LocalDate;
4. import java.time.Month;
5. import java.util.ArrayList;
6. import java.util.List;
8. public class Test {
9. public static void main(String[] args) {
10. List<LocalDate> dates = new ArrayList<>();
11. dates.add(LocalDate.parse("2018-07-11"));
12. dates.add(LocalDate.parse("1919-02-25"));
13. dates.add(LocalDate.of(2020, 4, 8));
14. dates.add(LocalDate.of(1980, Month.DECEMBER, 31));
16. dates.removeIf(x -> x.getYear() < 2000);
18. System.out.println(dates);
19. }
20. }

* 

**[1919-02-25, 1980-12-31]**

**(Incorrect)**

* 

**Runtime exception**

* 

**[2018-07-11, 2020-04-08]**

**(Correct)**

* 

**[2018-07-11, 1919-02-25, 2020-04-08, 1980-12-31]**

**Explanation**

LocalDate objects can be created by using static method parse and of.

removeIf(Predicate) method was added as a default method in Collection interface in JDK 8 and it removes all the elements of this collection that satisfy the given predicate.

Predicate's test method returns true for all the LocalDate objects with year less than 2000. So all the LocalDate objects with year value less than 2000 are removed from the list. Remaining LocalDate objects are printed in their insertion order.

Bottom of Form

Top of Form

Question 28: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.List;
7. class Student {
8. private String name;
9. private int age;
11. Student(String name, int age) {
12. this.name = name;
13. this.age = age;
14. }
16. public String toString() {
17. return "Student[" + name + ", " + age + "]";
18. }
19. }
21. public class Test {
22. public static void main(String[] args) {
23. List<Student> students = new ArrayList<>();
24. students.add(new Student("James", 25));
25. students.add(new Student("James", 27));
26. students.add(new Student("James", 25));
27. students.add(new Student("James", 25));
29. students.remove(new Student("James", 25));
31. for(Student stud : students) {
32. System.out.println(stud);
33. }
34. }
35. }

What will be the result of compiling and executing Test class?

* 

**Student[James, 27]**

* 

**Student[James, 27]  
Student[James, 25]  
Student[James, 25]**

* 

**Student[James, 25]  
Student[James, 27]  
Student[James, 25]**

* 

**Student[James, 25]  
Student[James, 27]  
Student[James, 25]  
Student[James, 25]**

**(Correct)**

**Explanation**

Before you answer this, you must know that there are 5 different Student object created in the memory (4 at the time of adding to the list and 1 at the time of removing from the list). This means these 5 Student objects will be stored at different memory addresses.

remove(Object) method removes the first occurrence of matching object and equals(Object) method decides whether 2 objects are equal or not. equals(Object) method defined in Object class uses == operator to check the equality and in this case as 5 Student objects are stored at different memory location, hence not equal.

Nothing is removed from the students list, all the 4 Student objects are printed in the insertion order.

Bottom of Form

Top of Form

Question 29: **Incorrect**

Consider codes below:

1. //A.java
2. package com.udayan.oca;
4. public class A {
5. public int i1 = 1;
6. protected int i2 = 2;
7. }
8. //B.java
9. package com.udayan.oca.test;
11. import com.udayan.oca.A;
13. public class B extends A {
14. public void print() {
15. A obj = new A();
16. System.out.println(obj.i1); //Line 8
17. System.out.println(obj.i2); //Line 9
18. System.out.println(this.i2); //Line 10
19. System.out.println(super.i2); //Line 11
20. }
22. public static void main(String [] args) {
23. new B().print();
24. }
25. }

One of the statements inside print() method is causing compilation failure. Which of the below solutions will help to resolve compilation error?

* 

**Comment the statement at Line 10**

**(Incorrect)**

* 

**Comment the statement at Line 8**

* 

**Comment the statement at Line 9**

**(Correct)**

* 

**Comment the statement at Line 11**

**Explanation**

class A is declared public and defined in com.udayan.oca package, there is no problem in accessing class A outside com.udayan.oca package.

class B is defined in com.udayan.oca.test package, to extend from class A either use import statement "import com.udayan.oca.A;" or fully qualified name of the class com.udayan.oca.A. No issues with this class definition as well.

Variable i1 is declared public in class A, so Line 8 doesn't cause any compilation error. Variable i2 is declared protected so it can only be accessed in subclass using inheritance but not using object reference variable. obj.i2 causes compilation failure.

class B inherits variable i2 from class A, so inside class B it can be accessed by using either this or super. Line 10 and Line 11 don't cause any compilation error.

Bottom of Form

Top of Form

Question 30: **Correct**

Which of these access modifiers can be used for a top level interface?

* 

**All of the other options**

* 

**private**

* 

**public**

**(Correct)**

* 

**protected**

**Explanation**

A top level interface can be declared with either public or default modifiers.

public interface is accessible across all packages but interface declared with default modifier and be accessed in the defining package only.

Bottom of Form

Top of Form

Question 31: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String [] args) {
5. int a = 100;
6. System.out.println(-a++);
7. }
8. }

* 

**-100**

**(Correct)**

* 

**99**

* 

**-101**

* 

**-99**

* 

**Compilation error**

**Explanation**

First add parenthesis (round brackets) to the given expression: -a++.

There are 2 operators involved. unary minus and Postfix operator. Let's start with expression and value of a.

-a++; [a = 100].

-(a++); [a = 100] Postfix operator has got higher precedence than unary operator.

-(100); [a = 101] Use the value of a (100) in the expression and after that increase the value of a to 101.

-100; [a = 101] -100 is printed on to the console.

Bottom of Form

Top of Form

Question 32: **Incorrect**

What will be the result of compiling and executing Test class?

1. //Test.java
2. package com.udayan.oca.test;
4. abstract class Animal {
5. private String name;
7. Animal(String name) {
8. this.name = name;
9. }
11. public String getName() {
12. return name;
13. }
14. }
16. class Dog extends Animal {
17. private String breed;
18. Dog(String breed) {
19. this.breed = breed;
20. }
22. Dog(String name, String breed) {
23. super(name);
24. this.breed = breed;
25. }
27. public String getBreed() {
28. return breed;
29. }
30. }
32. public class Test {
33. public static void main(String[] args) {
34. Dog dog1 = new Dog("Beagle");
35. Dog dog2 = new Dog("Bubbly", "Poodle");
36. System.out.println(dog1.getName() + ":" + dog1.getBreed() + ":" +
37. dog2.getName() + ":" + dog2.getBreed());
38. }
39. }

* 

**null:Beagle:Bubbly:Poodle**

**(Incorrect)**

* 

**Compilation error for Animal Class**

* 

**Compilation error for Dog(String, String) constructor**

* 

**:Beagle:Bubbly:Poodle**

* 

**Compilation error for Dog(String) constructor**

**(Correct)**

* 

**Compilation error for Animal(String) constructor**

**Explanation**

abstract class can have constructors and it also possible to have abstract class without any abstract method. So, there is no issue with Animal class.

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.

Inside Animal class Constructor, compiler adds super(); => Animal(String name) { super(); this.name = name; }, super() in this case invokes the no-arg constructor of Object class and hence no compilation error here.

Compiler changes Dog(String) constructor to: Dog(String breed) { super(); this.breed = breed; }. No-arg constructor is not available in Animal class and as another constructor is provided, java compiler doesn't add default constructor. Hence Dog(String) constructor gives compilation error.

There is no issue with Dog(String, String) constructor.

Bottom of Form

Top of Form

Question 33: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. class Message {
4. String msg = "Happy New Year!";
6. public void print() {
7. System.out.println(msg);
8. }
9. }
11. public class Test {
12. public static void change(Message m) { //Line n5
13. m = new Message(); //Line n6
14. m.msg = "Happy Holidays!"; //Line n7
15. }
17. public static void main(String[] args) {
18. Message obj = new Message(); //Line n1
19. obj.print(); //Line n2
20. change(obj); //Line n3
21. obj.print(); //Line n4
22. }
23. }

* 

**Happy New Year!  
Happy Holidays!**

* 

**Happy Holidays!  
Happy Holidays!**

* 

**null  
Happy New Year!**

* 

**Happy New Year!  
Happy New Year!**

**(Correct)**

**Explanation**

It is pass-by-reference scheme.

Initially, msg = "Happy New Year!"

Call to method change(Message) doesn't modify the msg property of passed object rather it creates another Message object and modifies the msg property of new object to "Happy Holidays!"

So, the instance of Message referred by obj remains unchanged.

Hence in the output, you get:

Happy New Year!

Happy New Year!

Bottom of Form

Top of Form

Question 34: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
6. public class Test {
7. public static void main(String [] args) {
8. LocalDate date = LocalDate.of(2020, 9, 31);
9. System.out.println(date);
10. }
11. }

What will be the result of compiling and executing Test class?

* 

**2020-10-01**

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**2020-09-30**

* 

**Compilation error**

**Explanation**

LocalDate.of(...) method first validates year, then month and finally day of the month.

September can't have 31 days so LocalDate.of(...) method throws an instance of java.time.DateTimeException class.

Bottom of Form

Top of Form

Question 35: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. public class Test {
5. public static void main(String[] args) {
6. StringBuilder sb = new StringBuilder(100);
7. System.out.println(sb.length() + ":" + sb.toString().length());
8. }
9. }

What will be the result of compiling and executing Test class?

* 

**16:16**

* 

**100:100**

* 

**16:0**

* 

**100:0**

* 

**0:0**

**(Correct)**

**Explanation**

`new StringBuilder(100);` creates a StringBuilder instance, whose internal char array's length is 100 but length() method of StringBuilder object returns the number of characters stored in the internal array and in this case it is 0. So, `sb.length()` returns 0.

sb.toString() is the String representation of StringBuilder instance and in this case as there are no characters inside the StringBuilder object, hence sb.toString() returns an empty String "", so `sb.toString().length()` also returns 0.

Output is 0:0.

Bottom of Form

Top of Form

Question 36: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. byte var = 100;
6. switch(var) {
7. case 100:
8. System.out.println("var is 100");
9. break;
10. case 200:
11. System.out.println("var is 200");
12. break;
13. default:
14. System.out.println("In default");
15. }
16. }
17. }

* 

**Compilation error**

**(Correct)**

* 

**var is 100**

* 

**var is 200**

* 

**In default**

**Explanation**

case values must evaluate to the same type / compatible type as the switch expression can use.

switch expression can accept following:

char or Character

byte or Byte

short or Short

int or Integer

An enum only from Java 6

A String expression only from Java 7

In this case, switch expression [switch (var)] is of byte type.

byte range is from -128 to 127. But in case expression [case 200], 200 is outside byte range and hence compilation error.

Bottom of Form

Top of Form

Question 37: **Correct**

For the class Test, which options, if used to replace /\*INSERT\*/, will print "Lucky no. 7" on to the console? Select 3 options.

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. /\*INSERT\*/
6. switch(var) {
7. case 7:
8. System.out.println("Lucky no. 7");
9. break;
10. default:
11. System.out.println("DEFAULT");
12. }
13. }
14. }

* 

**char var = '7';**

* 

**char var = 7;**

**(Correct)**

* 

**Integer var = 7;**

**(Correct)**

* 

**Character var = '7';**

* 

**Character var = 7;**

**(Correct)**

**Explanation**

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums.

In this case, all are valid values but only 3 executes "case 7:". case is comparing integer value 7.

NOTE: character seven, '7' is different from integer value seven, 7. So "char var = '7';" and "Character var = '7';" will print DEFAULT on to the console.

Bottom of Form

Top of Form

Question 38: **Correct**

Consider the following class:

1. package com.udayan.oca;
3. public class Employee {
4. public int passportNo; //line 2
5. }

Which of the following is the correct way to make the variable 'passportNo' read only for any other class?

* 

**Make 'passportNo' private.**

* 

**Remove 'public' from the 'passportNo' declaration. i.e., change line 2 to int passportNo;**

* 

**Make 'passportNo' static and provide a public static method getPassportNo() which will return its value.**

* 

**Make 'passportNo' private and provide a public method getPassportNo() which will return its value.**

**(Correct)**

**Explanation**

'passportNo' should be read-only for any other class.

This means make 'passportNo' private and provide public getter method. Don't provide public setter as then 'passportNo' will be read-write property.

If passportNo is declared with default scope, then other classes in the same package will be able to access passportNo for read-write operation.

Bottom of Form

Top of Form

Question 39: **Correct**

\_\_\_\_\_\_\_\_\_\_\_\_ uses access modifiers to protect variables and hide them within a class.

Which of the following options accurately fill in the blanks above?

* 

**Polymorphism**

* 

**Abstraction**

* 

**Encapsulation**

**(Correct)**

* 

**Inheritance**

**Explanation**

Encapsulation is all about having private instance variable and providing public getter and setter methods.

Bottom of Form

Top of Form

Question 40: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. String[] names = { "Smith", "Brown", "Thomas", "Taylor", "Jones" };
9. List<String> list = new ArrayList<>();
10. for (int x = 0; x < names.length; x++) {
11. list.add(names[x]);
12. switch (x) {
13. case 2:
14. continue;
15. }
16. break;
17. }
18. System.out.println(list.size());
19. }
20. }

* 

**0**

* 

**2**

* 

**None of the other options**

* 

**1**

**(Correct)**

* 

**5**

**(Incorrect)**

* 

**3**

* 

**4**

**Explanation**

break; and continue; are used inside for-loop, hence no compilation error.

In 1st iteration, x = 0. "Smith" is added to the list. There is no matching case found, hence control just goes after the switch-case block and executes break; statement, which takes the control out of the for loop. `System.out.println(list.size());` is executed and this prints 1 on to the console.

Bottom of Form

Top of Form

Question 41: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.List;
7. public class Test {
8. public static void main(String[] args) {
9. String s = new String("Hello");
10. List<String> list = new ArrayList<>();
11. list.add(s);
12. list.add(new String("Hello"));
13. list.add(s);
14. s.replace("l", "L");
16. System.out.println(list);
17. }
18. }

What will be the result of compiling and executing Test class?

* 

**[HeLLo, Hello, Hello]**

* 

**[HeLLo, Hello, HeLLo]**

* 

**[HeLLo, HeLLo, HeLLo]**

* 

**[Hello, Hello, Hello]**

**(Correct)**

**Explanation**

ArrayList's 1st and 3rd items are referring to same String instance referred by s [s --> "Hello"] and 2nd item is referring to another instance of String.

String is immutable, which means s.replace("l", "L"); creates another String instance "HeLLo" but s still refers to "Hello" [s --> "Hello"].

[Hello, Hello, Hello] is printed in the output.

Bottom of Form

Top of Form

Question 42: **Correct**

A bank's swift code is generally of 11 characters and used in international money transfers.   
An example of swift code: ICICINBBRT4  
ICIC: First 4 letters for bank code  
IN: Next 2 letters for Country code  
BB: Next 2 letters for Location code  
RT4: Next 3 letters for Branch code

Which of the following code correctly extracts country code from the swift code referred by String reference variable swiftCode?

* 

**swiftCode.substring(4, 5);**

* 

**swiftCode.substring(5, 7);**

* 

**swiftCode.substring(5, 6);**

* 

**swiftCode.substring(4, 6);**

**(Correct)**

**Explanation**

substring(int beginIndex, int endIndex) is used to extract the substring. The substring begins at "beginIndex" and extends till "endIndex - 1".

Country code information is stored at index 4 and 5, so the correct substring method to extract country code is: swiftCode.substring(4, 6);

Bottom of Form

Top of Form

Question 43: **Correct**

Fill in the blanks for the definition of java.lang.Error class:

public class java.lang.Error extends \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ {...}

* 

**RuntimeException**

* 

**Throwable**

**(Correct)**

* 

**Exception**

**Explanation**

An Error is a subclass of Throwable class.

Bottom of Form

Top of Form

Question 44: **Correct**

Consider 3 files:

1. //Order.java
2. package orders;
4. public class Order {
6. }
7. //Item.java
8. package orders.items;
10. public class Item {
12. }
13. //Shop.java
14. package shopping;
16. /\*INSERT\*/
18. public class Shop {
19. Order order = null;
20. Item item = null;
21. }

For the class Shop, which options, if used to replace /\*INSERT\*/, will resolve all the compilation errors?

Select 2 options.

* 
  1. import orders.\*;
  2. import orders.items.\*;

**(Correct)**

* 
  1. import orders.Order;
  2. import orders.items.Item;

**(Correct)**

* 
  1. import orders.\*;
  2. import items.\*;
* 

**import orders.\*;**

* 

**import orders.items.\*;**

**Explanation**

If you check the directory structure, you will find that directory "orders" contains "items", but orders and orders.items are different packages.import orders.\*; will only import all the classes in orders package but not in orders.items package.

You need to import Order and Item classes. To import Order class, use either import orders.Order; OR import orders.\*; and to import Item class, use either import orders.items.Item; OR import orders.items.\*;

Bottom of Form

Top of Form

Question 45: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int [] arr = {2, 1, 0};
6. for(int i : arr) {
7. System.out.println(arr[i]);
8. }
9. }
10. }

* 

**0  
1  
2**

**(Correct)**

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

* 

**2  
1  
0**

* 

**Compilation error**

**Explanation**

Inside enhanced for loop, System.out.println(arr[i]); is used instead of System.out.println(i);

When loop executes 1st time, i stores the first array element, which is 2 but System.out.println statement prints arr[2] which is 0. Loop executes in this manner and prints 0 1 2 on to the console.

Bottom of Form

Top of Form

Question 46: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String str1 = new String("Core");
6. String str2 = new String("CoRe");
7. System.out.println(str1 = str2);
8. }
9. }

* 

**true**

* 

**false**

* 

**CoRe**

**(Correct)**

* 

**Core**

**Explanation**

System.out.println(str1 = str2) has assignment(=) operator and not equality(==) operator.

After the assignment, str1 refers to "CoRe" and System.out.println prints "CoRe" to the console.

Bottom of Form

Top of Form

Question 47: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. class SpecialString {
5. String str;
6. SpecialString(String str) {
7. this.str = str;
8. }
9. }
11. public class Test {
12. public static void main(String[] args) {
13. Object [] arr = new Object[4];
14. for(int i = 1; i <=3; i++) {
15. switch(i) {
16. case 1:
17. arr[i] = new String("Java");
18. break;
19. case 2:
20. arr[i] = new StringBuilder("Java");
21. break;
22. case 3:
23. arr[i] = new SpecialString("Java");
24. break;
25. }
26. }
27. for(Object obj : arr) {
28. System.out.println(obj);
29. }
30. }
31. }

What will be the result of compiling and executing Test class?

* 

**Java**

**Java**

**Java**

* 

**Java**

**Java**

**<Some text containing @ symbol>**

**null**

* 

**Java**

**Java**

**Java**

**null**

* 

**Java**

**<Some text containing @ symbol>**

**<Some text containing @ symbol>**

**null**

* 

**null**

**Java**

**Java**

**Java**

* 

**null**

**Java**

**<Some text containing @ symbol>**

**<Some text containing @ symbol>**

* 

**null**

**Java**

**Java**

**<Some text containing @ symbol>**

**(Correct)**

* 

**Java**

**<Some text containing @ symbol>**

**<Some text containing @ symbol>**

* 

**Java**

**Java**

**<Some text containing @ symbol>**

**Explanation**

Variable 'arr' refers to an Object array of size 4 and null is assigned to all 4 elements of this array.

for-loop starts with i = 1, which means at 1st index String instance is stored, at 2nd index StringBuiler instance is stored and at 3rd index SpecialString instance is stored. null is stored at 0th index.

So, first null will be printed on to the console.

String and StringBuilder classes override toString() method, which prints the text stored in these classes. SpecialString class doesn't override toString() method and hence when instance of SpecialString is printed on to the console, you get: <fully qualified name of SpecialString class>@<hexadecimal representation of hashcode>.

Therefore output will be:

null

Java

Java

<Some text containing @ symbol>

Bottom of Form

Top of Form

Question 48: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String str = "java";
6. StringBuilder sb = new StringBuilder("java");
8. System.out.println(str.equals(sb) + ":" + sb.equals(str));
9. }
10. }

* 

**false:false**

**(Correct)**

* 

**Compilation error**

* 

**false:true**

* 

**true:false**

* 

**true:true**

**Explanation**

equals method declared in Object class has the declaration: public boolean equals(Object). Generally, equals method is used to compare different instances of same class but if you pass any other object, there is no compilation error. Parameter type is Object so it can accept any Java object.

str.equals(sb) => String class overrides equals(Object) method but as "sb" is of StringBuilder type so this returns false.

StringBuilder class doesn't override equals(Object) method. So Object version is invoked which uses == operator, hence sb.equals(str) returns false as well.

false:false is printed on to the console.

Bottom of Form

Top of Form

Question 49: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. double [] arr = new int[2]; //Line 3
6. System.out.println(arr[0]); //Line 4
7. }
8. }

* 

**Line 4 causes runtime exception**

**(Incorrect)**

* 

**0**

* 

**0.0**

* 

**Line 3 causes compilation error**

**(Correct)**

**Explanation**

int variable can easily be assigned to double type but double [] and int [] are not compatible. In fact, both are siblings and can't be assigned to each other, so Line 3 causes compilation failure.

Bottom of Form

Top of Form

Question 50: **Incorrect**

For the class Test, which options, if used to replace /\*INSERT\*/, will print "Hurrah! I passed..." on to the console? Select 2 options.

1. public class Test {
2. /\*INSERT\*/ {
3. System.out.println("Hurrah! I passed...");
4. }
5. }

* 

**protected static void main(String [] args)**

* 

**public void main(String [] args)**

* 

**static public void Main(String [] args)**

**(Incorrect)**

* 

**public void static main(String [] args)**

* 

**public static void main(String [] a)**

**(Correct)**

* 

**static public void main(String [] args)**

**(Correct)**

**Explanation**

As System.out.println needs to be executed on executing the Test class, this means special main method should replace /\*INSERT\*/.

Special main method's name should be "main" (all characters in lower case), should be static, should have public access specifier and it accepts argument of String [] type. String [] argument can use any identifier name, even though in most of the cases you will see "args" is used.

Position of static and public can be changed but return type must come just before the method name.

Bottom of Form

Top of Form

Question 51: **Incorrect**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. short [] args = new short[]{50, 50};
6. args[0] = 5;
7. args[1] = 10;
8. System.out.println("[" + args[0] + ", " + args[1] + "]");
9. }
10. }

* 

**An exception is thrown at runtime**

* 

**[50, 50]**

* 

**[5, 10]**

**(Incorrect)**

* 

**Compilation error**

**(Correct)**

**Explanation**

main method's parameter variable name is "args" and it is a local to main method.

So, same name "args" can't be used directly within the curly brackets of main method.

short [] args = new short[]{50, 50}; gives compilation error for using same name for local variable.

Bottom of Form

Top of Form

Question 52: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.Iterator;
6. import java.util.List;
7. import java.util.function.Predicate;
9. class Employee {
10. private String name;
11. private int age;
12. private double salary;
14. public Employee(String name, int age, double salary) {
15. this.name = name;
16. this.age = age;
17. this.salary = salary;
18. }
20. public String getName() {
21. return name;
22. }
24. public int getAge() {
25. return age;
26. }
28. public double getSalary() {
29. return salary;
30. }
32. public String toString() {
33. return name;
34. }
35. }
37. public class Test {
38. public static void main(String [] args) {
39. List<Employee> list = new ArrayList<>();
40. list.add(new Employee("James", 25, 15000));
41. list.add(new Employee("Lucy", 23, 12000));
42. list.add(new Employee("Bill", 27, 10000));
43. list.add(new Employee("Jack", 19, 5000));
44. list.add(new Employee("Liya", 20, 8000));
46. process(list, /\*INSERT\*/);
48. System.out.println(list);
49. }
51. private static void process(List<Employee> list, Predicate<Employee> predicate) {
52. Iterator<Employee> iterator = list.iterator();
53. while(iterator.hasNext()) {
54. if(predicate.test(iterator.next()))
55. iterator.remove();
56. }
57. }
58. }

Which of the following lambda expressions, if used to replace /\*INSERT\*/, prints [Jack, Liya] on to the console?

Select 2 options.

* 

**(Employee e) -> { return e.getSalary() >= 10000; }**

**(Correct)**

* 

**e -> e.getSalary() >= 10000**

**(Correct)**

* 

**(e) -> { e.getSalary() >= 10000; }**

* 

**e - > e.getSalary() >= 10000**

* 

**e -> { e.getSalary() >= 10000 }**

**Explanation**

Jack's salary is 5000 and Liya's salary is 8000. If Employee's salary is >= 10000 then that Employee object is removed from the list.

Allowed lambda expression is:

(Employee e) -> { return e.getSalary() >= 10000; },

Can be simplified to:  (e) -> { return e.getSalary() >= 10000; } => type can be removed from left side of the expression.

Further simplified to: e -> { return e.getSalary() >= 10000; } => if there is only one parameter in left part, then round brackets (parenthesis) can be removed.

Further simplified to: e -> e.getSalary() >= 10000 => if there is only one statement in the right side then semicolon inside the body, curly brackets and return statement can be removed. But all 3 [return, {}, ;] must be removed together.

NOTE: there should not be any space between - and > of arrow operator.

Bottom of Form

Top of Form

Question 53: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. private static void m(int x) {
5. System.out.println("int version");
6. }
8. private static void m(char x) {
9. System.out.println("char version");
10. }
12. public static void main(String [] args) {
13. int i = '5';
14. m(i);
15. m('5');
16. }
17. }

* 

**char version  
int version**

* 

**char version  
char version**

* 

**Compilation error**

* 

**int version  
char version**

**(Correct)**

* 

**int version  
int version**

**Explanation**

Method m is overloaded. Which overloaded method to invoke is decided at compile time. m(i) is tagged to m(int) as i is of int type and m('5') is tagged to m(char) as '5' is char literal.

Bottom of Form

Top of Form

Question 54: **Incorrect**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
5. import java.time.Period;
6. import java.time.format.DateTimeFormatter;
8. public class Test {
9. public static void main(String [] args) {
10. LocalDate date = LocalDate.of(2012, 1, 11);
11. Period period = Period.ofMonths(2);
12. DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM-dd-yy");
13. System.out.print(formatter.format(date.minus(period)));
14. }
15. }

What will be the result of compiling and executing Test class?

* 

**11-11-11**

**(Correct)**

* 

**01-11-11**

**(Incorrect)**

* 

**Runtime exception**

* 

**01-11-12**

* 

**11-11-12**

**Explanation**

date --> {2012-01-11}, period --> {P2M}, date.minus(period) --> {2011-11-11} [subtract 2 months period from {2012-01-11}, month is changed to 11 and year is changed to 2011].

formatter -> {MM-dd-yy}, when date {2011-11-11} is formatter in this format 11-11-11 is printed on to the console.

Bottom of Form

Top of Form

Question 55: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<Integer> list = new ArrayList<>();
9. list.add(100);
10. list.add(200);
11. list.add(100);
12. list.add(200);
13. list.remove(100);
15. System.out.println(list);
16. }
17. }

* 

**Exception is thrown at runtime**

**(Correct)**

* 

**[200, 200]**

* 

**Compilation error**

* 

**[100, 200, 200]**

* 

**[200, 100, 200]**

* 

**[200]**

**Explanation**

List cannot accept primitives, it can accept objects only. So, when 100 and 200 are added to the list, then auto-boxing feature converts these to wrapper objects of Integer type.

So, 4 items gets added to the list. One can expect the same behavior with remove method as well that 100 will be auto-boxed to Integer object.

But remove method is overloaded in List interface: remove(int) => Removes the element from the specified position in this list

and remove(Object)  => Removes the first occurrence of the specified element from the list.

As remove(int) version is available, which perfectly matches with the call remove(100); hence compiler does not do auto-boxing in this case.

But at runtime remove(100) tries to remove the element at 100th index and this throws IndexOutOfBoundsException.

Bottom of Form

Top of Form

Question 56: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. double price = 90000;
6. String model;
7. if(price > 100000) {
8. model = "Tesla Model X";
9. } else if(price <= 100000) {
10. model = "Tesla Model S";
11. }
12. System.out.println(model);
13. }
14. }

* 

**Compilation Error**

**(Correct)**

* 

**null**

* 

**Tesla Model X**

* 

**Tesla Model S**

**Explanation**

In this case "if - else if" block is used and not "if - else" block.

90000 is assigned to variable 'price' but you can assign parameter value or call some method returning double value, such as:

'double price = currentTemp();'.

In these cases compiler will not know the exact value until runtime, hence Java Compiler is not sure which boolean expression will be evaluated to true and so variable model may not be initialized.

Usage of LOCAL variable, 'model' without initialization gives compilation error. Hence, System.out.println(model); gives compilation error.

Bottom of Form

Top of Form

Question 57: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
5. import java.time.LocalTime;
7. public class Test {
8. public static void main(String [] args) {
9. LocalDate date = LocalDate.parse("1947-08-14");
10. LocalTime time = LocalTime.MAX;
11. System.out.println(date.atTime(time));
12. }
13. }

What will be the result of compiling and executing Test class?

* 

**1947-08-14T23:59:59**

* 

**1947-08-14T23:59:59.0**

* 

**1947-08-14T23:59:59.999999999**

**(Correct)**

* 

**1947-08-14T23:59:59.999**

**Explanation**

LocalTime.MIN --> {00:00}, LocalTime.MAX --> {23:59:59.999999999}, LocalTime.MIDNIGHT --> {00:00}, LocalTime.NOON --> {12:00}.

date.atTime(LocalTime) method creates a LocalDateTime instance by combining date and time parts.

toString() method of LocalDateTime class prints the date and time parts separated by T in upper case.

Bottom of Form

Top of Form

Question 58: **Incorrect**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. public class Test {
5. public static void main(String[] args) {
6. String s1 = "OCAJP";
7. String s2 = "OCAJP" + "";
8. System.out.println(s1 == s2);
9. }
10. }

What will be the result of compiling and executing Test class?

* 

**true**

**(Correct)**

* 

**Compilation error**

* 

**false**

**(Incorrect)**

* 

**OCAJP**

**Explanation**

Please note that Strings computed by concatenation at compile time, will be referred by String Pool during execution. Compile time String concatenation happens when both of the operands are compile time constants, such as literal, final variable etc.

For the statement, String s2 = "OCAJP" + "";, `"OCAJP" + ""` is a constant expression as both the operands "OCAJP" and "" are String literals, which means the expression `"OCAJP" + ""` is computed at compile-time and results in String literal "OCAJP".

So, during compilation, Java compiler translates the statement

String s2 = "OCAJP" + "";

to

String s2 = "OCAJP";

As "OCAJP" is a String literal, hence at runtime it will be referred by String Pool.

When Test class is executed,

s1 refers to "OCAJP" (String Pool object).

s2 also refers to same String pool object "OCAJP".

s1 and s2 both refer to the same String object and that is why s1 == s2 returns true.

Please note that Strings computed by concatenation at run time (if the resultant expression is not constant expression) are newly created and therefore distinct.

For below code snippet:

1. String s1 = "OCAJP";
2. String s2 = s1 + "";
3. System.out.println(s1 == s2);

Output is false, as s1 is a variable and `s1 + ""` is not a constant expression, therefore this expression is computed only at runtime and a new non-pool String object is created.

Bottom of Form

Top of Form

Question 59: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String fruit = "mango";
6. switch (fruit) {
7. default:
8. System.out.println("ANY FRUIT WILL DO");
9. case "Apple":
10. System.out.println("APPLE");
11. case "Mango":
12. System.out.println("MANGO");
13. case "Banana":
14. System.out.println("BANANA");
15. break;
16. }
17. }
18. }

* 

**MANGO  
BANANA**

* 

**MANGO**

* 

**ANY FRUIT WILL DO**

* 

**ANY FRUIT WILL DO  
APPLE  
MANGO  
BANANA**

**(Correct)**

**Explanation**

"mango" is different from "Mango", so there is no matching case available. default block is executed, "ANY FRUIT WILL DO" is printed on to the screen.

No break statement inside default, hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends.

So in this case, it prints APPLE, MANGO, BANANA one after another and break; statement takes control out of switch block. main method ends and program terminates successfully.

Bottom of Form

Top of Form

Question 60: **Incorrect**

Consider below code fragment:

1. interface Printable {
2. public void setMargin();
3. public void setOrientation();
4. }
6. abstract class Paper implements Printable { //Line 7
7. public void setMargin() {}
8. //Line 9
9. }
11. class NewsPaper extends Paper { //Line 12
12. public void setMargin() {}
13. //Line 14
14. }

Above code is currently giving compilation error. Which 2 modifications, done independently, enable the code to compile?

* 

**Insert at Line 9: public abstract void setOrientation();**

**(Incorrect)**

* 

**Replace the code at Line 7 with: class Paper implements Printable {**

* 

**Replace the code at Line 12 with: abstract class NewsPaper extends Paper {**

**(Correct)**

* 

**Insert at Line 14: public void setOrientation() {}**

**(Correct)**

**Explanation**

First you should find out the reason for compilation error. Methods declared in Printable interface are implicitly abstract, no issues with Printable interface.

class Paper is declared abstract and it implements Printable interface, it overrides setMargin() method but setOrientation() method is still abstract. No issues with class Paper as it is an abstract class and can have 0 or more abstract methods.

class NewsPaper is concrete class and it extends Paper class (which is abstract). So class NewsPaper must override setOrientation() method OR it must be declared abstract.

Replacing Line 9 with 'public abstract void setOrientation();' is not necessary and it will not resolve the compilation error in NewsPaper class.

Replacing Line 7 with 'class Paper implements Printable {' will cause compilation failure of Paper class as it inherits abstract method 'setOrientation'.

Bottom of Form

Top of Form

Question 61: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
6. class MyLocalDate extends LocalDate {
7. @Override
8. public String toString() {
9. return super.getDayOfMonth() + "-" + super.getMonthValue() +
10. "-" + super.getYear();
11. }
12. }
14. public class Test {
15. public static void main(String [] args) {
16. MyLocalDate date = LocalDate.parse("1980-03-16");
17. System.out.println(date);
18. }
19. }

What will be the result of compiling and executing Test class?

* 

**16-03-1980**

* 

**Compilation error**

**(Correct)**

* 

**16-3-1980**

* 

**An exception is thrown at runtime**

* 

**1980-03-16**

**Explanation**

LocalDate is a final class so cannot be extended.

Bottom of Form

Top of Form

Question 62: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. StringBuilder sb = new StringBuilder();
6. System.out.println(sb.append(null).length());
7. }
8. }

* 

**4**

* 

**NullPointerException is thrown at runtime**

* 

**Compilation error**

**(Correct)**

* 

**1**

**Explanation**

'append' method is overloaded in StringBuilder class: append(String), append(StringBuffer) and append(char[]) etc.

In this case compiler gets confused as to which method `append(null)` can be tagged because String, StringBuffer and char[] are not related to each other in multilevel inheritance. Hence `sb.append(null)` causes compilation error.

Bottom of Form

Top of Form

Question 63: **Correct**

Which of the following is not a valid array declaration?

* 

**int [][] arr2 = new int[8][8];**

* 

**int [] arr3 [] = new int[8][];**

* 

**int arr4[][] = new int[][8];**

**(Correct)**

* 

**int [] arr1 = new int[8];**

**Explanation**

1st array dimension must be specified at the time of declaration. new int[][8]; gives compilation error as 1st dimension is not specified.

Bottom of Form

Top of Form

Question 64: **Correct**

Consider below code of Guest.java file:

1. //Guest.java
2. class Message {
3. static void main(String [] args) {
4. System.out.println("Welcome " + args[2] + "!");
5. }
6. }
8. public class Guest {
9. public static void main(String [] args) {
10. Message.main(args);
11. }
12. }

And the commands:  
javac Guest.java

java Guest Clare Waight Keller

What is the result?

* 

**Welcome Clare!**

* 

**Welcome Keller!**

**(Correct)**

* 

**Compilation error as main method is not public in Message class**

* 

**Some other error as main method can't be invoked manually**

* 

**Welcome Waight!**

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

**Explanation**

Class Guest has special main method but main method defined in Message class is not public and hence it can't be called by JVM. But there is no issue with the syntax hence no compilation error.

**java Guest Clare Waight Keller**passes new String [] {"Clare", "Waight", "Keller"} to args of Guest.main method.

Guest.main method invokes Message.main method with the same argument: new String [] {"Clare", "Waight", "Keller"}. args[2] is "Keller" hence "Welcome Keller!" gets printed on to the console.

Bottom of Form

Top of Form

Question 65: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. StringBuilder sb = new StringBuilder("Java");
6. String s1 = sb.toString();
7. String s2 = sb.toString();
9. System.out.println(s1 == s2);
10. }
11. }

* 

**Compilation error**

* 

**true**

* 

**An exception is thrown at runtime**

* 

**false**

**(Correct)**

**Explanation**

toString() method defined in StringBuilder class doesn't use String literal rather uses the constructor of String class to create the instance of String class.

So both s1 and s2 refer to different String instances even though their contents are same. s1 == s2 returns false.

Bottom of Form

Top of Form

Question 66: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. import java.time.LocalTime;
5. public class Test {
6. public static void main(String[] args) {
7. LocalTime time = LocalTime.of(16, 40);
8. String amPm = time.getHour() >= 12 ? (time.getHour() == 12) ? "PM" : "AM";
9. System.out.println(amPm);
10. }
11. }

* 

**AM**

* 

**Compilation error**

**(Correct)**

* 

**PM**

* 

**An exception is thrown at runtime**

**Explanation**

This question is on ternary operator (?:). If an expression has multiple ternary operators then number of ? and : should match.

Given expression contains 2 ? and 1 : and hence Compilation Error.

Bottom of Form

Top of Form

Question 67: **Correct**

Given the code of Test.java file:

1. package com.udayan.oca;
3. class Point {
4. int x;
5. int y;
6. void assign(int x, int y) {
7. x = this.x;
8. this.y = y;
9. }
11. public String toString() {
12. return "Point(" + x + ", " + y + ")";
13. }
14. }
16. public class Test {
17. public static void main(String[] args) {
18. Point p1 = new Point();
19. p1.x = 10;
20. p1.y = 20;
21. Point p2 = new Point();
22. p2.assign(p1.x, p1.y);
23. System.out.println(p1.toString() + ";" + p2.toString());
24. }
25. }

What will be the result of compiling and executing Test class?

* 

**None of the other options**

* 

**Point(10, 20);Point(10, 20)**

* 

**Point(0, 20);Point(0, 20)**

* 

**Point(10, 20);Point(0, 20)**

**(Correct)**

* 

**Point(0, 20);Point(10, 20)**

**Explanation**

HINT: First check if members are accessible or not. All the codes are in same file Test.java, and Point class & variable x, y are declared with default modifier hence these can be accessed within the same package. Class Test belongs to same package so no issues in accessing Point class and instance variables of Point class. Make use of pen and paper to draw the memory diagrams (heap and stack). It will be pretty quick to reach the result.

Point p1 = new Point(); means p1.x = 0 and p1.y = 0 as instance variable are initialized to respective zeros.

p1.x = 10; means replace 0 with 10 in p1.x,

p1.y = 20; means replace 0 with 20 in p1.y,

Point p2 = new Point(); means p2.x = 0 and p2.y = 0 as instance variable are initialized to respective zeros.

p2.assign(p1.x, p1.y); invokes the assign method, parameter variable x = 10 and y = 20.

As assign is invoked on p2 reference variable hence this and p2 refers to same Point object.

x = this.x; means assign 0 to parameter variable x, no changes in this.y, which means p2.x is unchanged.

this.y = y; means assign 20 to this.y, which means p2.y is now 20

So after assign method is invoked and control goes back to main method: p1.x = 10, p1.y = 20, p2.x = 0 and p2.y = 20.

Output is: Point(10, 20);Point(0, 20)

Bottom of Form

Top of Form

Question 68: **Correct**

What will be the result of compiling and executing Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. m1(); //Line 3
6. }
8. private static void m1() throws Exception { //Line 6
9. System.out.println("NOT THROWING ANY EXCEPTION"); //Line 7
10. }
11. }

* 

**NOT THROWING ANY EXCEPTION**

* 

**Compilation error at Line 6**

* 

**Compilation error at Line 3**

**(Correct)**

* 

**Compilation error at Line 7**

**Explanation**

If a method declares to throw Exception or its sub-type other than RuntimeException types, then calling method should follow handle or declare rule. In this case, as method m1() declares to throw Exception, so main method should either declare the same exception or its super type in its throws clause OR m1(); should be surrounded by try-catch block.

Line 3 in this case causes compilation error.

Bottom of Form

Top of Form

Question 69: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.List;
7. public class Test {
8. public static void main(String[] args) {
9. List<String> list1 = new ArrayList<>();
10. list1.add("A");
11. list1.add("D");
13. List<String> list2 = new ArrayList<>();
14. list2.add("B");
15. list2.add("C");
17. list1.addAll(1, list2);
19. System.out.println(list1);
20. }
21. }

What will be the result of compiling and executing Test class?

* 

**[A, B, C, D]**

**(Correct)**

* 

**[A, B, C]**

* 

**[A, D, B, C]**

* 

**[A, D]**

**Explanation**

list1 --> [A, D],

list2 --> [B, C],

list1.addAll(1, list2); is almost equal to list1.add(1, [B, C]); => Inserts B at index 1, C takes index 2 and D is moved to index 3. list1 --> [A, B, C, D]

Bottom of Form

Top of Form

Question 70: **Correct**

What will be the result of compiling and executing the Test class?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int grade = 60;
6. if(grade = 60)
7. System.out.println("You passed...");
8. else
9. System.out.println("You failed...");
10. }
11. }

* 

**Produces no output**

* 

**You failed…**

* 

**You passed…**

* 

**Compilation error**

**(Correct)**

**Explanation**

Following are allowed in boolean expression of if statement:

1. Any expression whose result is either true or false. e.g. age > 20

2. A boolean variable. e.g. flag

3. A boolean literal: true or false

4. A boolean assignment. e.g. flag = true

boolean expression in this case is: (grade = 60), which is an int assignment and not boolean assignment. Hence Compilation error.

Bottom of Form

Retake test

**Continue**