Top of Form

Question 1: **Correct**

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 = {"\*", "\*\*", "\*\*\*", "\*\*\*\*", "\*\*\*\*\*"};
8. Predicate pr1 = s -> s.length() < 4;
9. print(arr, pr1);
10. }
12. private static void print(String [] arr, Predicate<String> predicate) {
13. for(String str : arr) {
14. if(predicate.test(str)) {
15. System.out.println(str);
16. }
17. }
18. }
19. }

* 

**\*  
\*\*  
\*\*\***

* 

**\*  
\*\*  
\*\*\*  
\*\*\*\*  
\*\*\*\*\***

* 

**\*  
\*\*  
\*\*\*  
\*\*\*\***

* 

**Compilation error**

**(Correct)**

**Explanation**

Though Predicate is a generic interface but raw type is also allowed. Type of the variable in lambda expression is inferred by the generic type of Predicate<T> interface.

In this case, Predicate pr1 = s -> s.length() < 4; Predicate is considered of Object type so variable "s" is of Object type and Object class doesn't have length() method. So, s.length() causes compilation error.

Bottom of Form

Top of Form

Question 2: **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.parse("2018-1-01");
9. System.out.println(date);
10. }
11. }

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

* 

**2018-1-1**

* 

**2018-1-01**

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**2018-01-01**

**Explanation**

LocalDate.parse(CharSequence) method accepts String in "9999-99-99" format only. Single digit month and day value are padded with 0 to convert it to 2 digits.

To represent 9th June 2018, format String must be "2018-06-09".

If correct format string is not passed then an instance of java.time.format.DateTimeParseException is thrown.

Bottom of Form

Top of Form

Question 3: **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, e -> e.getAge() > 20);
47. }
49. private static void process(List<Employee> list, Predicate<Employee> predicate) {
50. Iterator<Employee> iterator = list.iterator();
51. while(iterator.hasNext()) {
52. Employee e = iterator.next();
53. if(predicate.test(e))
54. System.out.print(e + " ");
55. }
56. }
57. }

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

* 

**Compilation error**

* 

**James Lucy Bill Jack Liya**

* 

**James Lucy Bill**

**(Correct)**

* 

**Jack Liya**

**Explanation**

process(List, Predicate) method prints all the records passing the Predicate's test and test is to process the records having age greater than 20.

There are 3 records with age > 20 and these are printed in the insertion order.

NOTE: toString() method just returns the name.

Bottom of Form

Top of Form

Question 4: **Correct**

Which of the following are Java Exception classes?

Select 3 options.

* 

**IllegalArgumentException**

**(Correct)**

* 

**ArrayIndexException**

* 

**ClassCastException**

**(Correct)**

* 

**NullException**

* 

**NumberFormatException**

**(Correct)**

**Explanation**

ClassCastException, NumberFormatException and IllegalArgumentException are Runtime exceptions.

There are no exception classes in java with the names: NullException and ArrayIndexException.

Bottom of Form

Top of Form

Question 5: **Correct**

Choose the options that meets the following specification:

Create a well encapsulated class Clock with one instance variable model.

The value of model should be accessible and modifiable outside Clock.

* 
  1. public class Clock {
  2. public String model;
  3. public String getModel() { return model; }
  4. public void setModel(String val) { model = val; }
  5. }
* 
  1. public class Clock {
  2. public String model;
  3. private String getModel() { return model; }
  4. private void setModel(String val) { model = val; }
  5. }
* 
  1. public class Clock {
  2. public String model;
  3. }
* 
  1. public class Clock {
  2. private String model;
  3. public String getModel() { return model; }
  4. public void setModel(String val) { model = val; }
  5. }

**(Correct)**

**Explanation**

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

Bottom of Form

Top of Form

Question 6: **Correct**

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

1. package com.udayan.oca;
3. class Vehicle {
4. public int getRegistrationNumber() {
5. return 1;
6. }
7. }
9. class Car {
10. public int getRegistrationNumber() {
11. return 2;
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. Vehicle obj = new Car();
18. System.out.println(obj.getRegistrationNumber());
19. }
20. }

* 

**An exception is thrown at runtime**

* 

**2**

* 

**1**

* 

**Compilation error**

**(Correct)**

**Explanation**

class Car doesn't extend from Vehicle class, this means Vehicle is not super type of Car.

Hence, Vehicle obj = new Car(); causes compilation error.

Bottom of Form

Top of Form

Question 7: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. class Parent {
5. public String toString() {
6. return "Inner ";
7. }
8. }
10. class Child extends Parent {
11. public String toString() {
12. return super.toString().concat("Peace!");
13. }
14. }
16. public class Test {
17. public static void main(String[] args) {
18. System.out.println(new Child());
19. }
20. }

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

* 

**Inner Peace!**

**(Correct)**

* 

**Peace!**

* 

**Compilation error**

* 

**Inner**

**Explanation**

System.out.println(new Child()); invokes the toString() method on Child's instance.

Parent class's method can be invoked by super keyword. super.toString() method returns "Inner " and "Inner ".concat("Peace!") returns "Inner Peace!".

Bottom of Form

Top of Form

Question 8: **Correct**

Consider codes below:

1. //A.java
2. package com.udayan.oca;
4. public class A {
5. public int i1;
6. protected int i2;
7. int i3;
8. private int i4;
9. }
10. //TestA.java
11. package com.udayan.oca.test;
13. import com.udayan.oca.A; //Line 3
15. public class TestA {
16. public static void main(String[] args) {
17. A obj = new A(); //Line 7
18. System.out.println(obj.i1); //Line 8
19. System.out.println(obj.i2); //Line 9
20. System.out.println(obj.i3); //Line 10
21. System.out.println(obj.i4); //Line 11
22. }
23. }

Which of the following 3 statements are true?

* 

**Line 7 causes compilation error**

* 

**Line 10 causes compilation error**

**(Correct)**

* 

**Line 8 causes compilation error**

* 

**Line 11 causes compilation error**

**(Correct)**

* 

**Line 9 causes compilation error**

**(Correct)**

* 

**Line 3 causes compilation error**

**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 TestA is defined in com.udayan.oca.test package, to use class A either use import statement "import com.udayan.oca.A;" or fully qualified name of the class com.udayan.oca.A. No issues at Line 3 and LIne 7.

As TestA is in different package so it can only access public members of class A using object reference. Line 8 compiles successfully.

protected, default and private members are not accessible outside com.udayan.oca package using object reference.

NOTE: protected members can be accessed outside but only through inheritance and not object reference.

Bottom of Form

Top of Form

Question 9: **Correct**

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**

* 

**01-11-11**

* 

**01-11-12**

* 

**Runtime exception**

**(Correct)**

* 

**11-11-12**

**Explanation**

While working with dates, programmers get confused with M & m and D & d.

Easy way to remember is that Bigger(Upper case) letters represent something bigger. M represents month & m represents minute, D represents day of the year & d represents day of the month.

LocalDate's object doesn't have time component, mm represents minute and not months so at runtime format method throws exception.

Bottom of Form

Top of Form

Question 10: **Correct**

What will be the result of executing Test class using below command?

java Test good morning everyone

1. private class Test
2. {
3. public static void main(String args[])
4. {
5. System.out.println(args[1]);
6. }
7. }

* 

**Compilation Error**

**(Correct)**

* 

**morning**

* 

**good**

* 

**everyone**

**Explanation**

Top level class can have two access modifiers: public and default.

Over here Test class has private modifier and hence compilation error.

Bottom of Form

Top of Form

Question 11: **Correct**

How can you force JVM to run Garbage Collector?

* 

**By setting the reference variable to null.**

* 

**By calling: System.gc();**

* 

**JVM cannot be forced to run Garbage Collector.**

**(Correct)**

* 

**By calling: Runtime.getRuntime().gc();**

**Explanation**

Both Runtime.getRuntime().gc(); and System.gc(); do the same thing, these make a request to JVM to run Garbage Collector.

JVM makes the best effort to run Garbage Collector but nothing is guaranteed.

Setting the reference variable to null will make the object eligible for Garbage Collection, if there are no other references to this object. But this doesn't force JVM to run the Garbage Collector. Garbage Collection cannot be forced.

Bottom of Form

Top of Form

Question 12: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
6. public class Test {
7. public static void main(String[] args) {
8. ArrayList<Integer> original = new ArrayList<>();
9. original.add(new Integer(10));
11. ArrayList<Integer> cloned = (ArrayList<Integer>) original.clone();
12. Integer i1 = cloned.get(0);
13. ++i1;
15. System.out.println(cloned);
16. }
17. }

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

* 

**An exception is thrown at runtime**

* 

**Compilation error**

* 

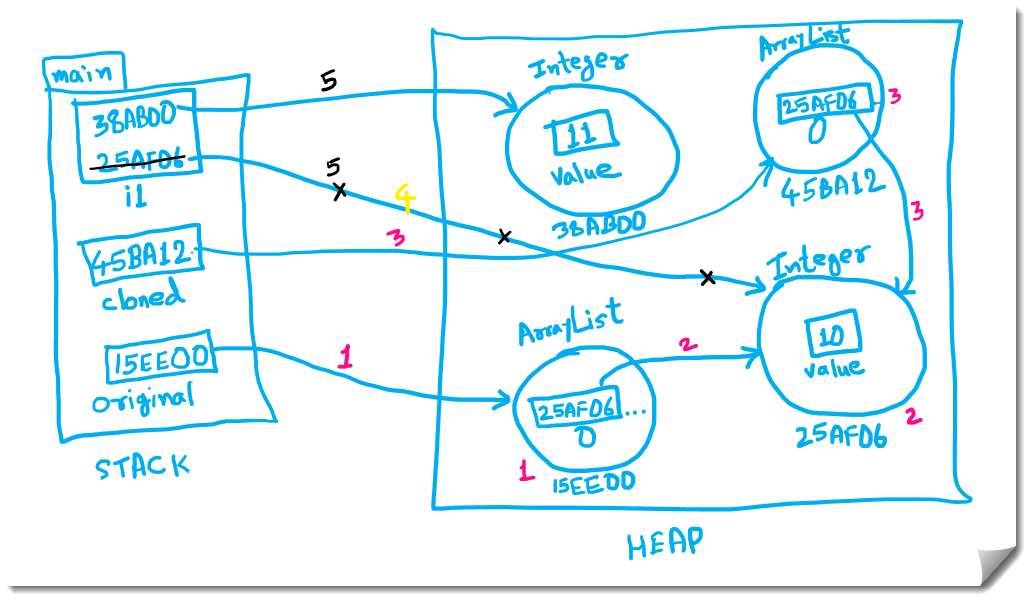
**[11]**

* 

**[10]**

**(Correct)**

**Explanation**



Let's see what is happening during execution:

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

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

2. original.add(new Integer(10)); => It creates an Integer 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 Integer object at the memory location 25AF06.

3. ArrayList<Integer> cloned = (ArrayList<Integer>) 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 Integer 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 Integer object.

4. Integer i1 = cloned.get(0); => cloned.get(0) returns the Integer object stored at the memory location 25AF06 and variable 'i1' refers to it.

5. ++i1; => As Integer object is immutable, hence ++i1; creates a new Integer object with value 11 and suppose this newly created Integer object is stored at memory location 38AB00. This means variable 'i1' stops referring to Integer object at the memory location 25AF06 and starts referring to Integer object at the memory location 38AB00.

Cloned list stays intact and still refers to Integer object at memory location 25AF06.

6. System.out.println(cloned); => Prints [10] on to the console as cloned contains an element which refers to Integer object containing value 10.

Bottom of Form

Top of Form

Question 13: **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 = {3, 2, 1};
6. for(int i : arr) {
7. System.out.println(arr[i]);
8. }
9. }
10. }

* 

**1  
2  
3**

* 

**Compilation error**

* 

**3  
2  
1**

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

**(Correct)**

**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 3 but System.out.println statement prints arr[3] and this causes java runtime to throw the instance of ArrayIndexOutOfBoundsException.

Bottom of Form

Top of Form

Question 14: **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<String> list = new ArrayList<>();
9. list.add(null);
10. list.add(null);
11. list.add(null);
12. System.out.println(list.remove(0) + ":" + list.remove(null));
13. }
14. }

* 

**NullPointerException is thrown at runtime**

* 

**null:true**

**(Correct)**

* 

**true:false**

* 

**true:true**

* 

**null:null**

**Explanation**

It is possible to add null to ArrayList instant.

Initially list has 3 elements: [null, null, null].

remove(int) returns the deleted member of the list. In this case `list.remove(0);` returns null as null was deleted from the 0th index. So, list is left with 2 elements: [null, null].

remove(Object) returns true if deletion was successful otherwise false. In this case `list.remove(null)` removes first null from the list and returns true and list is left with just one element: [null].

Hence, the output is: 'null:true'.

Bottom of Form

Top of Form

Question 15: **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>();
10. list.add(27);
11. list.add(27);
13. list.add(new Integer(27));
14. list.add(new Integer(27));
16. System.out.println(list.get(0) == list.get(1));
17. System.out.println(list.get(2) == list.get(3));
18. }
19. }

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

* 

**false  
true**

* 

**true  
true**

* 

**true  
false**

**(Correct)**

* 

**false  
false**

**Explanation**

This is bit tricky. Just remember this:

Two instances of following wrapper objects, created through auto-boxing will always be same, if their primitive values are same:

Boolean,

Byte,

Character from \u0000 to \u007f (7f equals to 127),

Short and Integer from -128 to 127.

For 1st statement, list.add(27); => Auto-boxing creates an integer object for 27.

For 2nd statement, list.add(27); => Java compiler finds that there is already an Integer object in the memory with value 27, so it uses the same object.

That is why System.out.println(list.get(0) == list.get(1)); returns true.

new Integer(27) creates a new Object in the memory, so System.out.println(list.get(2) == list.get(3)); returns false.

Bottom of Form

Top of Form

Question 16: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.List;
6. import java.util.ListIterator;
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. ListIterator<String> iterator = dryFruits.listIterator();
17. while(iterator.hasNext()) {
18. if(iterator.next().startsWith("A")) {
19. iterator.remove();
20. }
21. }
23. System.out.println(dryFruits);
24. }
25. }

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

* 

**Compilation error**

* 

**[Walnut, Date]**

**(Correct)**

* 

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

* 

**An exception is thrown at runtime**

**Explanation**

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.

In this case ListIterator.remove() method is used. startsWith("A") returns true for "Apricot" and "Almond" so these elements are removed from the list. In the output, [Walnut, Date] is displayed.

Bottom of Form

Top of Form

Question 17: **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[][] arr = { { "7", "6", "5" }, { "4", "3" }, { "2", "1" } };
6. for (int i = 0; i < arr.length; i++) { //Line n1
7. for (int j = 0; j < arr[i].length; j++) { //Line n2
8. switch (arr[i][j]) { //Line n3
9. case "2":
10. case "4":
11. case "6":
12. break; //Line n4
13. default:
14. continue; //Line n5
15. }
16. System.out.print(arr[i][j]); //Line n6
17. }
18. }
19. }
20. }

* 

**7654321**

* 

**753**

* 

**75**

* 

**7**

* 

**6**

* 

**7531**

* 

**64**

* 

**642**

**(Correct)**

**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 (arr[i][j])] is of String type.

Please note that break; statement at Line n4 takes the control to Line n6 (outside switch-case block) and not out of the inner for loop, where as, continue; statement at Line n5 takes the control to the update expression (j++) of Line n2.

arr.length is 3, so outer loop executes 3 times.

1st iteration of outer loop, i=0.

-1st iteration of inner loop, i=0, j=0 and arr[0].length = 3. 0 < 3 evaluates to true.

  arr[0][0] = "7", Line n5 is executed, and it takes the control to j++ (j = 1). 1 < 3 evaluates to true.

-2nd iteration of inner loop

  arr[0][1] = "6", Line n4 is executed, and it takes the control to Line n6. 6 is printed on to the console. Control goes to j++ (j = 2). 2 < 3 evaluates to true.

-3rd iteration of inner loop

  arr[0][2] = "5", Line n5 is executed, and it takes the control to j++ (j = 3).   As 3 < 3 evaluates to false, control exits inner loop and goes to i++.

You must have noticed that 1st iteration of outer loop prints the even number of 1st array { "7", "6", "5" }

Similarly, 2nd iteration of outer loop prints the even number of 2nd array { "4", "3" }, which is 4

and 3rd iteration of outer loop prints the even number of 3rd array { "2", "1" }, which is 2.

Therefore, the output is: 642.

Bottom of Form

Top of Form

Question 18: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. public class Test {
5. public static void main(String[] args) {
6. final String fName = "James";
7. String lName = "Gosling";
8. String name1 = fName + lName;
9. String name2 = fName + "Gosling";
10. String name3 = "James" + "Gosling";
11. System.out.println(name1 == name2);
12. System.out.println(name2 == name3);
13. }
14. }

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

* 

**false**

**false**

* 

**false**

**true**

**(Correct)**

* 

**true**

**false**

* 

**true**

**true**

**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.

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

fName is a constant variable and lName is a non-constant variable.

`fName + lName` is not a constant expression and hence the expression will be computed at run-time and the resultant String object "JamesGosling" will not be referred by String Pool.

As fName is constant variable and "Gosling" is String literal, hence the expression `fName + "Gosling"` is a constant expression, therefore expression is computed at compile-time and results in String literal "JamesGosling".

So, during compilation, Java compiler translates the statement

String name2 = fName + "Gosling";

to

String name2 = "JamesGosling";

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

So, at runtime name1 and name2 refer to different String object and that is why name1 == name2 returns false.

`"James" + "Gosling"` is also a constant expression and hence Java compiler translates the statement

String name3 = "James" + "Gosling";

to

String name3 = "JamesGosling";

This means at runtime, variable 'name3' will refer to the same String pool object "JamesGosling", which is referred by variable 'name3'.

So, name2 and name3 refer to same String object and that is why name2 == name3 returns true.

Bottom of Form

Top of Form

Question 19: **Incorrect**

Given code of LogHelper.java file:

1. package com.udayan.oca;
3. abstract class Helper {
4. int num = 100;
5. String operation = null;
7. protected abstract void help();
9. void log() {
10. System.out.println("Helper-log");
11. }
12. }
14. public class LogHelper extends Helper {
15. private int num = 200;
16. protected String operation = "LOGGING";
18. void help() {
19. System.out.println("LogHelper-help");
20. }
22. void log() {
23. System.out.println("LogHelper-log");
24. }
26. public static void main(String [] args) {
27. new LogHelper().help();
28. }
29. }

Which of the following changes, done independently, allows the code to compile and on execution prints LogHelper-help?

Select 3 options.

* 

**Add the public modifier to the help() method of LogHelper class**

**(Correct)**

* 

**Add the public modifier to the log() method of LogHelper class**

**(Incorrect)**

* 

**Add the protected modifier to the help() method of LogHelper class**

**(Correct)**

* 

**Remove the private modifier from the num variable of LogHelper class**

* 

**Add the protected modifier to the log() method of LogHelper class**

**(Incorrect)**

* 

**Remove the protected modifier from the operation variable of LogHelper class**

**(Incorrect)**

* 

**Add the protected modifier to the log() method of Helper class**

* 

**Remove the protected modifier from the help() method of Helper class**

**(Correct)**

**Explanation**

Let us first find out the issue:

As instance variables are hidden by subclasses and not overridden, hence overriding rules are not for the instance variables. There are no issues with variables 'num' and 'operation'.

log() method is declared with default modifier in both the classes, hence no issue with log() method as well.

abstract method help() is declared with protected modifier in Helper class and in LogHelper class, it is overridden with default modifier and this causes compilation error. So below solutions to resolved this issue:

1. Remove the protected modifier from the help() method of Helper class: Both the overridden and overriding methods will have same default modifier, which is allowed

OR

2. Add the protected modifier to the help() method of LogHelper class: Both the overridden and overriding methods will have same protected modifier, which is allowed

OR

3. Add the public modifier to the help() method of LogHelper class: Overridden method will have protected modifier and overriding method will have public modifier, which is allowed

Bottom of Form

Top of Form

Question 20: **Correct**

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

1. package com.udayan.oca;
3. public class Test {
4. public String name;
5. public void Test() {
6. name = "James";
7. }
9. public static void main(String [] args) {
10. Test obj = new Test();
11. System.out.println(obj.name);
12. }
13. }

* 

**James**

* 

**Compilation error**

* 

**null**

**(Correct)**

* 

**None of the above**

**Explanation**

public void Test() is method and not constructor, as return type is void.

method can have same name as the class name, so no issues with Test() method declaration.

As there are no constructors available for this class, java compiler adds following constructor.

public Test() {}

Test obj = new Test(); invokes the default constructor but it doesn't change the value of name property (by default null is assigned to name property)

System.out.println(obj.name); prints null.

Bottom of Form

Top of Form

Question 21: **Correct**

Consider below code:

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. processStringArray(arr, /\*INSERT\*/);
10. }
12. private static void processStringArray(String [] arr,
13. Predicate<String> predicate) {
14. for(String str : arr) {
15. if(predicate.test(str)) {
16. System.out.println(str);
17. }
18. }
19. }
20. }

Which of the following options can replace /\*INSERT\*/ such that on executing Test class all the array elements are displayed in the output? Select ALL that apply.

* 

**p -> p.length() >= 1**

**(Correct)**

* 

**p -> !false**

**(Correct)**

* 

**p -> p.length() < 10**

**(Correct)**

* 

**p -> true**

**(Correct)**

**Explanation**

p -> true means test method returns true for the passed String.

p -> !false means test method returns true for the passed String.

p -> p.length() >= 1 means test method returns true if passed String's length is greater than or equal to 1 and this is true for all the array elements.

p -> p.length() < 10 means test method returns true if passed String's length is less than 10 and this is true for all the array elements.

Bottom of Form

Top of Form

Question 22: **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 s1 = "OcA";
6. String s2 = "oCa";
7. System.out.println(s1.equals(s2));
8. }
9. }

* 

**None of the other options**

* 

**Compilation error**

* 

**false**

**(Correct)**

* 

**true**

**Explanation**

equals(String str) method of String class matches two String objects and it takes character's case into account while matching.

Alphabet A in upper case and alphabet a in lower case are not equal according to this method.

As String objects referred by s1 and s2 have different cases, hence output is false.

Bottom of Form

Top of Form

Question 23: **Incorrect**

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> list = new ArrayList<>();
10. list.add("ONE");
11. list.add("TWO");
12. list.add("THREE");
13. list.add("THREE");
15. if(list.remove(2)) {
16. list.remove("THREE");
17. }
19. System.out.println(list);
20. }
21. }

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

* 

**An exception is thrown at runtime**

* 

**[ONE, TWO, THREE]**

* 

**[ONE, TWO, THREE, THREE]**

* 

**Compilation error**

**(Correct)**

* 

**[ONE, TWO]**

**(Incorrect)**

**Explanation**

list.remove(Object) method returns boolean result but list.remove(int index) returns the removed item from the list, which in this case is of String type and not Boolean type and hence if(list.remove(2)) causes compilation error.

Bottom of Form

Top of Form

Question 24: **Correct**

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

1. //Test.java
2. package com.udayan.oca;
4. class Point {
5. static int x;
6. int y, z;
8. public String toString() {
9. return "Point(" + x + ", " + y + ", " + z + ")";
10. }
11. }
13. public class Test {
14. public static void main(String[] args) {
15. Point p1 = new Point();
16. p1.x = 17;
17. p1.y = 35;
18. p1.z = -1;
20. Point p2 = new Point();
21. p2.x = 19;
22. p2.y = 40;
23. p2.z = 0;
25. System.out.println(p1); //Line n1
26. System.out.println(p2); //Line n2
27. }
28. }

* 

**Point(17, 35, -1)**

**Point(19, 40, 0)**

* 

**Point(17, 35, -1)**

**Point(17, 35, -1)**

* 

**Point(19, 35, -1)**

**Point(19, 35, -1)**

* 

**Compilation error**

* 

**Point(17, 35, -1)**

**Point(17, 40, 0)**

* 

**Point(19, 35, -1)**

**Point(19, 40, 0)**

**(Correct)**

* 

**Point(19, 40, 0)**

**Point(19, 40, 0)**

**Explanation**

Point class correctly overrides the toString() method. Even though variable x is static, but it can be easily accessed by instance method toString().

Variables x, y and z are declared with default scope, so can be accessed in same package. There is no compilation error in the code.

There is only one copy of static variable for all the instances of the class. Variable x is shared by p1 and p2 both.

p1.x = 17; sets the value of static variable x to 17, p2.x = 19; modifies the value of static variable x to 19. As there is just one copy of x, hence p1.x = 19

Please note: p1.x and p2.x don't cause any compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Point.x is the preferred syntax.

Each object has its own copy of instance variables, so just before executing Line n1, p1.y = 35 & p1.z = -1 AND p2.y = 40 & p2.z = 0

Output is:

Point(19, 35, -1)

Point(19, 40, 0)

Bottom of Form

Top of Form

Question 25: **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 d1 = LocalDate.parse("1999-09-09");
9. LocalDate d2 = LocalDate.parse("1999-09-09");
10. LocalDate d3 = LocalDate.of(1999, 9, 9);
11. LocalDate d4 = LocalDate.of(1999, 9, 9);
12. System.out.println((d1 == d2) + ":" + (d2 == d3) + ":" + (d3 == d4));
13. }
14. }

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

* 

**true:false:true**

* 

**false:false:true**

* 

**false:false:false**

**(Correct)**

* 

**true:true:true**

**Explanation**

"parse" and "of" methods create new instances, so in this case you get 4 different instance of LocalDate stored at 4 different memory addresses.

Bottom of Form

Top of Form

Question 26: **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. System.out.println("Output is: " + (10 != 5));
6. }
7. }

* 

**Output is: false**

* 

**Compilation error**

* 

**Output is: (10 != 5)**

* 

**Output is: true**

**(Correct)**

**Explanation**

"Output is: " + (10 != 5) [Nothing to evaluate at left side, so let's evaluate the right side of +, 10 != 5 is true.]

= "Output is: " + true [+ operator behaves as concatenation operator]

= "Output is: true"

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("2018-7-11"));
12. dates.add(LocalDate.parse("1919-10-25"));
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. }

* 

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

**(Incorrect)**

* 

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

* 

**Runtime exception**

**(Correct)**

* 

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

**Explanation**

LocalDate.parse(CharSequence text) method accepts String in "9999-99-99" format only, in which month and day part in the passed object referred by text should be of 2 digits, such as to represent MARCH, use 03 and not 3 & to represent 4th day of the month, use 04 and not 4.

Single digit month and day value are not automatically padded with 0 to convert it to 2 digits.

To represent 9th June 2018, format String must be "2018-06-09".

If you pass "2018-6-9" or "2018-06-9" or "2018-6-09" (not in correct formats), then an instance of java.time.format.DateTimeParseException will be thrown.

In this question, LocalDate.parse("2018-7-11") throws an exception at runtime as JULY is represented as 7, whereas it should be represented as 07.

Bottom of Form

Top of Form

Question 28: **Correct**

Which of the following correctly defines class Printer?

* 
  1. package com.udayan.oca;
  2. package com.udayan.ocp;
  3. public class Printer {
  5. }
* 
  1. package com.udayan.oca;
  2. package com.udayan.ocp;
  3. import java.io.\*
  4. public class Printer {
  6. }
* 
  1. public class Printer {
  2. package com.udayan.oca;
  3. }
* 
  1. package com.udayan.oca;
  2. import java.util.\*;
  3. public class Printer {
  5. }

**(Correct)**

**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.

Multiple package statements are not allowed.

Bottom of Form

Top of Form

Question 29: **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 x = 5;
6. while (x < 10)
7. System.out.println(x);
8. x++;
9. }
10. }

* 

**It will go in an infinite loop**

**(Correct)**

* 

**5  
6  
7  
8  
9**

* 

**Produces no output**

* 

**Compilation error**

**Explanation**

while loop doesn't have curly bracket over here, so only System.out.println(x) belongs to while loop.

Above syntax can be written as follows:

1. int x = 5;
2. while (x < 10) {
3. System.out.println(x);
4. }
5. x++;

As x++; is outside loop, hence value of x is always 5 within loop, 5 < 10 is true for all the iterations and hence infinite loop.

Bottom of Form

Top of Form

Question 30: **Incorrect**

Which is not a valid statement based on given code?

1. class A{}
2. class B extends A{}

* 

**B a = new B();**

* 

**A a = new B();**

**(Incorrect)**

* 

**B b = new A();**

**(Correct)**

* 

**A a = new A();**

**Explanation**

B b = new A(); -> child class reference cannot refer to parent class object. This will give compilation error.

A a = new B(); -> parent class reference can refer to child class object. This is Polymorphism.

B a = new B(); -> No issues at all.

A a = new A(); -> No issues at all.

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 = 2;
6. boolean res = false;
7. res = a++ == 2 || --a == 2 && --a == 2;
8. System.out.println(a);
9. }
10. }

* 

**3**

**(Correct)**

* 

**1**

* 

**Compilation error**

* 

**2**

**Explanation**

a++ == 2 || --a == 2 && --a == 2; [Given expression].

(a++) == 2 || --a == 2 && --a == 2; [Postfix has got higher precedence than other operators].

(a++) == 2 || (--a) == 2 && (--a) == 2; [After postfix, precedence is given to prefix].

((a++) == 2) || ((--a) == 2) && ((--a) == 2); [== has higher precedence over && and ||].

((a++) == 2) || (((--a) == 2) && ((--a) == 2)); [&& has higher precedence over ||].

Let's start solving it:

((a++) == 2) || (((--a) == 2) && ((--a) == 2)); [a=2, res=false].

(2 == 2) || (((--a) == 2) && ((--a) == 2)); [a=3, res=false].

true || (((--a) == 2) && ((--a) == 2)); [a=3, res=false].  || is a short-circuit operator, hence no need to evaluate expression on the right.

res is true and a is 3.

Bottom of Form

Top of Form

Question 32: **Correct**

What is the output if below program is run with the command line:

java Test

1. public class Test {
2. public static void main(String[] args) {
3. System.out.println(args.length);
4. }
5. }

* 

**NullPointerException**

* 

**0**

**(Correct)**

* 

**ArrayIndexOutOfBoundsException**

* 

**1**

**Explanation**

We have not passed any command-line arguments, hence args refers to an array object of Size 0.

args.length prints 0. args is not null and hence no NullPointerException.

Also we are not accessing array element so no question of ArrayIndexOutOfBoundsException as well.

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. 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(new Integer(100));
15. System.out.println(list);
16. }
17. }

* 

**Compilation error**

* 

**[200]**

* 

**[100, 200, 200]**

* 

**[200, 200]**

* 

**Exception is thrown at runtime**

* 

**[200, 100, 200]**

**(Correct)**

**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: [100, 200, 100, 200]. list.remove(new Integer(100)); removes the first occurrence of 100 from the list, which means the 1st element of the list.

After removal list contains: [200, 100, 200].

NOTE: String class and all the wrapper classes override equals(Object) method, hence at the time of removal when another instance is passes[new Integer(100)], there is no issue in removing the matching item.

Bottom of Form

Top of Form

Question 34: **Correct**

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(2, 1, 0).ofYears(10).ofMonths(5).ofDays(2);
9. System.out.println(period);
10. }
11. }

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

* 

**P2Y1M0D**

* 

**P2D**

**(Correct)**

* 

**P2Y1M**

* 

**P12Y6M2D**

**Explanation**

of and ofXXX methods are static methods and not instance methods.

Period.of(2, 1, 0) => returns an instance of Period type.

static methods can be invoked using class\_name or using reference variable. In this case ofYears(10) is invoked on period instance but compiler uses Period's instance to resolve the type, which is period. A new Period instance {P10Y} is created, after that another Period instance {P5M} is created and finally Period instance {P2D} is created.

This instance is assigned to period reference variable and hence P2D is printed on to the console.

Bottom of Form

Top of Form

Question 35: **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 = "Java";
9. System.out.println(s1 == s2);
10. }
11. }

* 

**false**

**(Correct)**

* 

**true**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**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 36: **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 m1() throws Exception {
5. throw new Exception();
6. }
8. public static void main(String[] args) {
9. try {
10. m1();
11. } finally {
12. System.out.println("A");
13. }
14. }
15. }

* 

**Compilation error.**

**(Correct)**

* 

**A is printed to the console and program ends normally.**

* 

**A is printed to the console, stack trace is printed and then program ends normally.**

* 

**A is printed to the console, stack trace is printed and then program ends abruptly.**

**Explanation**

Method m1() throws Exception (checked) and it declares to throw it, so no issues with method m1().

But main() method neither provides catch handler nor throws clause and hence main method gives Compilation error.

Handle or Declare rule should be followed for checked exception if you are not re-throwing it.

Bottom of Form

Top of Form

Question 37: **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 [] arr = new String[7];
6. System.out.println(arr);
7. }
8. }

* 

**NullPointerException**

* 

**null**

* 

**Some String containing @ symbol**

**(Correct)**

* 

**Compilation Error**

**Explanation**

Variable arr refers to an array object of String of 7 elements.

Variable arr contains the memory address of String array object.

arr is of reference type, hence it prints some String Containing @ symbol.

Bottom of Form

Top of Form

Question 38: **Correct**

Consider the following interface declaration:

1. public interface I1 {
2. void m1() throws java.io.IOException;
3. }

Which of the following incorrectly implements interface I1?

* 
  1. public class C3 implements I1 {
  2. public void m1() throws java.io.IOException{}
  3. }
* 
  1. public class C4 implements I1 {
  2. public void m1() throws Exception{}
  3. }

**(Correct)**

* 
  1. public class C2 implements I1 {
  2. public void m1() throws java.io.FileNotFoundException{}
  3. }
* 
  1. public class C1 implements I1 {
  2. public void m1() {}
  3. }

**Explanation**

NOTE: Question is asking for "incorrect" implementation and not "correct" implementation.

According to overriding rules, if super class / interface method declares to throw a checked exception, then overriding method of sub class / implementer class has following options:

1. May not declare to throw any checked exception,

2. May declare to throw the same checked exception thrown by super class / interface method,

3. May declare to throw the sub class of the exception thrown by super class / interface method,

4. Cannot declare to throw the super class of the exception thrown by super class / interface method

Bottom of Form

Top of Form

Question 39: **Correct**

What will be the result of compiling and executing the following program?

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 (IOException e) {
23. System.out.print("A");
24. } catch(FileNotFoundException e) {
25. System.out.print("B");
26. } finally {
27. System.out.print("C");
28. }
29. }
30. }

* 

**class Sub gives compilation error**

* 

**BC**

* 

**AC**

* 

**class Test gives compilation error**

**(Correct)**

**Explanation**

FileNotFoundException extends IOException and hence catch block of FileNotFoundException should appear before the catch block of IOException.

Therefore, class Test causes compilation error.

Bottom of Form

Top of Form

Question 40: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
5. import java.time.Period;
7. public class Test {
8. public static void main(String [] args) {
9. LocalDate date = LocalDate.parse("2000-01-01");
10. Period period = Period.ofYears(-3000);
11. System.out.println(date.plus(period));
12. }
13. }

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

* 

**Compilation error**

* 

**5000-01-01**

* 

**-1000-01-01**

**(Correct)**

* 

**Runtime exception**

* 

**1000-01-01**

**Explanation**

The minimum supported LocalDate is: {-999999999-01-01} and maximum supported LocalDate is: {+999999999-12-31}.

If period of -3000 years is added to 1st Jan 2000, then result is 1st Jan -1000.

Bottom of Form

Top of Form

Question 41: **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. System.out.println(new Boolean("ture"));
6. }
7. }

* 

**false**

**(Correct)**

* 

**true**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

Boolean class code uses equalsIgnoreCase method to validate the passed String, so if passed String is "true" ('t', 'r', 'u' and 'e' can be in any case), then boolean value stored in Boolean object is true otherwise false.

In this question passed String is "ture" and not "true" and that is why false is printed on to the console.

Bottom of Form

Top of Form

Question 42: **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.parse("1980-03-16");
9. System.out.println(date.minusYears(-5));
10. }
11. }

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

* 

**Compilation error**

* 

**Runtime exception**

* 

**1975-03-16**

* 

**1985-03-16**

**(Correct)**

**Explanation**

minusYears, minusMonths, minusWeeks, minusDays methods accept long parameter so you can pass either positive or negative value.

If positive value is passed, then that specified value is subtracted and if negative value is passed, then that specified value is added. I think you still remember: minus minus is plus.

Similarly plusYears, plusMonths, plusWeeks, plusDays methods work in the same manner.

If positive value is passed, then that specified value is added and if negative value is passed, then that specified value is subtracted.

Bottom of Form

Top of Form

Question 43: **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("Hurrah! I Passed...");
6. sb.delete(0, 100);
7. System.out.println(sb.length());
8. }
9. }

* 

**16**

* 

**19**

* 

**StringIndexOutOfBoundsException is thrown at runtime**

* 

**0**

**(Correct)**

**Explanation**

'delete' method accepts 2 parameters: delete(int start, int end), where start is inclusive and end is exclusive.

This method throws StringIndexOutOfBoundsException for following scenarios:

A. start is negative

B. start is greater than sb.length()

C. start is greater than end

If end is greater than the length of StringBuilder object, then StringIndexOutOfBoundsException is not thrown and end is set to sb.length().

So, in this case, `sb.delete(0, 100);` is equivalent to `sb.delete(0, sb.length());` and this deletes all the characters from the StringBuilder object.

Hence, System.out.println(sb.length()); prints 0 on to the console.

Bottom of Form

Top of Form

Question 44: **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 = " ";
6. boolean b1 = str1.isEmpty();
7. System.out.println(b1);
8. str1.trim();
9. b1 = str1.isEmpty();
10. System.out.println(b1);
11. }
12. }

* 

**false  
true**

* 

**true  
true**

* 

**true  
false**

* 

**false  
false**

**(Correct)**

**Explanation**

str1 refers to single space character and isEmpty() method of String returns true if no characters are there in the String. As str1 contains single space, hence b1 is false.

false is first printed on to the console.

str1.trim(); => creates an empty string "" but str1 still refers to single space string " ".

b1 = str1.isEmpty(); assigns false to b1 and last System.out.println statement prints false on to the console. So output is:

false

false

Bottom of Form

Top of Form

Question 45: **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 = 75;
6. if(grade > 60)
7. System.out.println("Congratulations");
8. System.out.println("You passed");
9. else
10. System.out.println("You failed");
11. }
12. }

* 

**You failed**

* 

**Congratulations  
You passed**

* 

**Congratulations**

* 

**Compilation error**

**(Correct)**

**Explanation**

As there is no brackets after if, hence only one statement is part of if block and other is outside.

Above code can be written as below:

1. if(grade > 60) {
2. System.out.println("Congratulations");
3. }
4. System.out.println("You passed");
5. else
6. System.out.println("You failed");

There should not be anything between if-else block but in this case, System.out.println("You passed"); is between if-else and thus Compilation error.

Bottom of Form

Top of Form

Question 46: **Correct**

For the class Test, which options, if used to replace /\*INSERT\*/, will print TEN on to the console? Select 4 options.

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. /\*INSERT\*/
6. switch(var) {
7. case 10:
8. System.out.println("TEN");
9. break;
10. default:
11. System.out.println("DEFAULT");
12. }
13. }
14. }

* 

**long var = 10;**

* 

**Integer var = 10;**

**(Correct)**

* 

**byte var = 10;**

**(Correct)**

* 

**double var = 10;**

* 

**char var = 10;**

**(Correct)**

* 

**Short var = 10;**

**(Correct)**

**Explanation**

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

In this case long and double are invalid values to be passed in switch expression. char uses 16 bits (2 Bytes) and its range is 0 to 65535 (no signed bit reserved) so it can easily store value 10.

Bottom of Form

Top of Form

Question 47: **Correct**

Which of the following keywords is used to manually throw an exception?

* 

**thrown**

* 

**throws**

* 

**throw**

**(Correct)**

* 

**catch**

**Explanation**

catch is for catching the exception and not throwing it.

thrown is not a java keyword.

throws is used to declare the exceptions a method can throw.

To manually throw an exception, throw keyword is used. e.g., throw new Exception();

Bottom of Form

Top of Form

Question 48: **Correct**

Following statement in a Java program compiles successfully:

student.report(course);

What can you say for sure?

* 

**student is the class name**

* 

**course must be of String type**

* 

**student is the reference variable name**

* 

**report is the method name**

**(Correct)**

**Explanation**

It is good practice to have first character of class name in upper case, but it is not mandatory.

student can be either class name or reference variable name.

Syntax to invoke static method is: Class\_Name.method\_name(<arguments>); OR reference\_variable\_name.method\_name(<arguments>);

Syntax to invoke instance method is: reference\_variable\_name.method\_name(<arguments>);

If student represents class\_name or refernce\_variable\_name, then report might be the static method of the class.

If student represents reference\_variable\_name, then report is the instance method of the class.

In both the cases, report must be the method name.

Type of argument cannot be found out by looking at above syntax.

Bottom of Form

Top of Form

Question 49: **Correct**

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

1. package com.udayan.oca;
3. public class Test {
5. private static void add(double d1, double d2) {
6. System.out.println("double version: " + (d1 + d2));
7. }
9. private static void add(Double d1, Double d2) {
10. System.out.println("Double version: " + (d1 + d2));
11. }
13. public static void main(String[] args) {
14. add(10.0, null);
15. }
17. }

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**Compilation error**

* 

**double version: 10.0**

* 

**Double version: 10.0**

**Explanation**

add(10.0, null); => Compiler can't convert null to double primitive type, so 2nd argument is tagged to Double reference type.

So to match the method call, 10.0 is converted to Double object by auto-boxing and add(10.0, null); is tagged to add(Double, Double); method.

But at the time of execution, d2 is null so System.out.println("Double version: " + (d1 + d2)); throws NullPointerException.

Bottom of Form

Top of Form

Question 50: **Correct**

Consider the code snippet:

1. import java.util.ArrayList;
2. import java.util.List;
4. public class Test {
5. List list1 = new ArrayList<String>(); //Line 5
6. List<String> list2 = new ArrayList(); //Line 6
7. List<> list3 = new ArrayList<String>(); //Line 7
8. List<String> list4 = new ArrayList<String>(); //Line 8
9. List<String> list5 = new ArrayList<>(); //Line 9
10. }

Which of the following statements compile without any warning?

Select 2 options.

* 

**Line 8**

**(Correct)**

* 

**Line 7**

* 

**Line 9**

**(Correct)**

* 

**Line 5**

* 

**Line 6**

**Explanation**

Line 8's syntax was added in JDK 5 and it compiles without any warnings.

Line 9's syntax was added in JDK 7, in which type parameter can be ignored from right side of the statement, it is inferred from left side, so Line 9 also compiles without any warning.

Type parameter can't be removed from declaration part, hence Line 7 gives compilation error.

Both Line 5 and Line 6 are mixing Generic type with Raw type and hence warning is given by the compiler.

Bottom of Form

Top of Form

Question 51: **Correct**

Consider below code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. char c = 'Z';
6. long l = 100\_00l;
7. int i = 9\_2;
8. float f = 2.02f;
9. double d = 10\_0.35d;
10. l = c + i;
11. f = c \* l \* i \* f;
12. f = l + i + c;
13. i = (int)d;
14. f = (long)d;
15. }
16. }

Does above code compile successfully?

* 

**Yes**

**(Correct)**

* 

**No**

**Explanation**

For readability purpose underscore (\_) is used to separate numeric values. This is very useful in representing big numbers such as credit card numbers (1234\_7654\_9876\_0987). long data can be suffixed by l, float by f and double by d. So first 5 variable declaration and assignment statements inside main(String []) method don't cause any compilation error.

Let's check rest of the statements:

l = c + i; => Left side variable 'l' is of long type and right side expression evaluates to an int value, which can easily be assigned to long type. No compilation error here.

f = c \* l \* i \* f; => Left side variable 'f' is of float type and right side expression evaluates to float value, which can easily be assigned to float type. Hence, it compiles successfully.

f = l + i + c; => Left side variable 'f' is of float type and right side expression evaluates to long value, which can easily be assigned to float type. Hence, no issues here.

i = (int)d; => double can't be assigned to int without explicit casting, right side expression `(int)d;` is casting double to int, so no issues.

f = (long)d; => double can't be assigned to float without explicit casting, right side expression `(long)d;` is casting double to long, which can easily be assigned to float type. It compiles successfully.

Bottom of Form

Top of Form

Question 52: **Correct**

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

1. //Test.java
2. package com.udayan.oca;
4. class Parent {
5. int i = 10;
6. Parent(int i) {
7. super();
8. this.i = i;
9. }
10. }
12. class Child extends Parent {
13. int j = 20;
15. Child(int j) {
16. super(0);
17. this.j = j;
18. }
20. Child(int i, int j) {
21. super(i);
22. this(j);
23. }
25. }
27. public class Test {
28. public static void main(String[] args) {
29. Child child = new Child(1000, 2000);
30. System.out.println(child.i + ":" + child.j);
31. }
32. }

* 

**Compilation error for Child(int, int) Constructor**

**(Correct)**

* 

**Compilation error for Parent(int) constructor**

* 

**1000:2000**

* 

**Compilation error for Test class**

* 

**1000:0**

* 

**Compilation error for Child(int) constructor**

**Explanation**

super(); inside Parent(int) constructor invokes the no-arg constructor of Object class and hence no compilation error for Parent(int) constructor.

super(0); inside Child(int) constructor invokes Parent(int) constructor, which is available and hence no issues.

Child(int, int) constructor has both super(i) and this(j) statements. A constructor should have super(...) or this(...) but not both. Hence Child(int, int) causes compilation failure.

As all the classes are defined in Test.java file under com.udayan.oca.test package, hence child.i and child.j don't give compilation error. i and j are declared with package scope.

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

* 

**MANGO  
BANANA**

* 

**MANGO  
BANANA  
ANY FRUIT WILL DO**

* 

**MANGO  
ANY FRUIT WILL DO**

* 

**ANY FRUIT WILL DO**

**(Correct)**

* 

**MANGO**

**Explanation**

"mango" is different from "Mango", so there is no matching case available.

default block is executed and as it is the last block inside switch hence after printing "ANY FRUIT WILL DO" control goes out of switch block, main method ends and program terminates successfully.

Bottom of Form

Top of Form

Question 54: **Correct**

Consider code of Test.java file:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String [] args) {
5. int [] arr = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
6. String str = process(arr, 3, 8); //Line 5
7. System.out.println(str);
8. }
10. /\*INSERT\*/
11. }

Line 5 is giving compilation error as process method is not found.

Which of the following method definitions, if used to replace /\*INSERT\*/, will resolve the compilation error?

* 
  1. private static int[] process(int [] arr, int start, int end) {
  2. return null;
  3. }
* 
  1. private static String process(int [] arr, int start, int end) {
  2. return null;
  3. }

**(Correct)**

* 
  1. private static String[] process(int [] arr, int start, int end) {
  2. return null;
  3. }
* 
  1. private static int process(int [] arr, int start, int end) {
  2. return null;
  3. }

**Explanation**

It is clear from Line 5 that, method name should be process, it should be static method, it should accept 3 parameters (int[], int, int) and its return type must be String.

Bottom of Form

Top of Form

Question 55: **Incorrect**

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

1. package com.udayan.oca;
3. public class Test {
5. private static void add(double d1, double d2) {
6. System.out.println("double version: " + (d1 + d2));
7. }
9. private static void add(Double d1, Double d2) {
10. System.out.println("Double version: " + (d1 + d2));
11. }
13. public static void main(String[] args) {
14. add(10.0, new Integer(10));
15. }
17. }

* 

**An exception is thrown at runtime**

**(Incorrect)**

* 

**Compilation error**

* 

**double version: 20.0**

**(Correct)**

* 

**Double version: 20.0**

**Explanation**

int can be converted to double but Integer type can't be converted to Double type as Integer and Double are siblings (both extends from Number class) so can't be casted to each other.

add(10.0, new Integer(10)); => 1st parameter is tagged to double primitive type and 2nd parameter is converted to int, is tagged to double primitive type as well. So, add(double, double); method is invoked.

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. m1(null);
6. }
8. static void m1(CharSequence s) {
9. System.out.println("CharSequence");
10. }
12. static void m1(String s) {
13. System.out.println("String");
14. }
16. static void m1(Object s) {
17. System.out.println("Object");
18. }
19. }

* 

**Compilation Error**

* 

**Object**

* 

**CharSequence**

* 

**String**

**(Correct)**

**Explanation**

Method m1 is overloaded to accept 3 different parameters: String, CharSequence and Object.

String implements CharSequence and Object is the super Parent class in Java. There is no conflict among the overloaded methods for the call m1(null) as it is mapped to the class lowest in hierarchy, which is String class. Hence, output will be "String".

Now if you add one more overloaded method, `static void m1(StringBuilder s) {...}` in the Test class, then `m1(null);` would cause compilation error as it would match to both m1(StringBuilder) and m1(String) methods. So m1(null) in that case would be ambiguous call and would cause compilation error.

For the same reason, System.out.println(null); causes compilation error as println method is overloaded to accept 3 reference types Object, String and char [] along with primitive types.

System.out.println(null); matches to both println(char[]) and println(String), so it is an ambiguous call and hence the compilation error.

Bottom of Form

Top of Form

Question 57: **Correct**

Which of the following array declarations and initializations is NOT legal?

* 

**int [] arr3 = new int[3]{10, 20, 30};**

**(Correct)**

* 

**byte [] val = new byte[10];**

* 

**char [] arr1 [] = new char[5][];**

* 

**int [] arr2 = {1, 2, 3, 4, 5};**

**Explanation**

You can't specify size at the time of initializing with data, hence new int[3]{10, 20, 30}; gives compilation error.

Bottom of Form

Top of Form

Question 58: **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 eventDate = LocalDate.of(2018, 1, 1);
10. boolean flag1 = newYear.isAfter(eventDate);
11. boolean flag2 = newYear.isBefore(eventDate);
12. System.out.println(flag1 + ":" + flag2);
13. }
14. }

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

* 

**true:true**

* 

**false:true**

* 

**false:false**

**(Correct)**

* 

**true:false**

**Explanation**

isAfter and isBefore method can be interpreted as:

Does 1st Jan 2018 come after 1st Jan 2018? No, false.  Does 1st Jan 2018 come before 1st Jan 2018? No, false.

Bottom of Form

Top of Form

Question 59: **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> list = new ArrayList<>();
10. list.add(0, "Array");
11. list.add(0, "List");
13. System.out.println(list);
14. }
15. }

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

* 

**An exception is thrown at runtime**

* 

**[Array]**

* 

**[Array, List]**

* 

**[List]**

* 

**[List, Array]**

**(Correct)**

**Explanation**

list.add(0, "Array"); means list --> [Array],

list.add(0, "List"); means insert "List" to 0th index and shift "Array" to right. So after this operation, list --> [List, Array]. In the console, [List, Array] is printed.

Bottom of Form

Top of Form

Question 60: **Incorrect**

Consider the code of Test.java file:

1. package com.udayan.oca;
3. class Student {
4. String name;
5. int age;
7. Student() {
8. Student("James", 25);
9. }
11. 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?

* 

**Compilation error**

**(Correct)**

* 

**null:0**

**(Incorrect)**

* 

**An exception is thrown at runtime**

* 

**James:25**

**Explanation**

A constructor can call another constructor by using this(...) and not the constructor name.

Hence Student("James", 25); causes compilation error.

Bottom of Form

Top of Form

Question 61: **Correct**

Consider given code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String [][] fruits = {{"apple", "mango"}, {"orange", "grape"}};
6. /\*INSERT\*/
7. }
8. }

For the class Test, which options, if used to replace /\*INSERT\*/, will print "apple mango orange grape " on to the console?

Select 2 options.

* 
  1. for(int i = 1; i < fruits.length; i++)
  2. for(int j = 1; j < fruits[i].length ; j++)
  3. System.out.print(fruits[i][j] + " ");
* 
  1. for(int i = 0; i < fruits.length; i++)
  2. for(int j = 0; j < fruits[i].length ; j++)
  3. System.out.print(fruits[i][j] + " ");

**(Correct)**

* 
  1. for(String [] arr : fruits)
  2. for(String fruit : arr)
  3. System.out.print(fruit + " ");

**(Correct)**

* 
  1. for(int i = 1; i <= fruits.length; i++)
  2. for(int j = 1; j <= fruits[i].length ; j++)
  3. System.out.print(fruits[i][j] + " ");

**Explanation**

Easy question on iterating through 2-dimensional array. Starting index should be 0 and not 1. Enhanced for loop syntax is correct.

As for loops contain 1 statement, hence curly brackets can be ignored.

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. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<String> list = new ArrayList<>();
9. list.add("X");
10. list.add("Y");
11. list.add("X");
12. list.add("Y");
13. list.add("Z");
14. list.remove(new String("Y"));
15. System.out.println(list);
16. }
17. }

* 

**Compilation error**

* 

**[X, Z]**

* 

**[X, Y, Z]**

* 

**[X, X, Y, Z]**

**(Correct)**

* 

**[X, X, Z]**

* 

**Exception is thrown at runtime**

**Explanation**

After all the add statements are executed, list contains: [X, Y, X, Y, Z].

list.remove(new String("Y")); removes the first occurrence of "Y" from the list, which means the 2nd element of the list. After removal list contains: [X, X, Y, Z].

NOTE: String class and all the wrapper classes override equals(Object) method, hence at the time of removal when another instance is passes [new String("Y")], there is no issue in removing the  matching item.

Bottom of Form

Top of Form

Question 63: **Correct**

Which of these keywords can be used to prevent inheritance of a class?

* 

**class**

* 

**final**

**(Correct)**

* 

**super**

* 

**constant**

**Explanation**

Class declared as final can't be inherited. Examples are: String, Integer, System etc.

Bottom of Form

Top of Form

Question 64: **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.msg = "Happy Holidays!"; //Line n6
14. }
16. public static void main(String[] args) {
17. Message obj = new Message(); //Line n1
18. obj.print(); //Line n2
19. change(obj); //Line n3
20. obj.print(); //Line n4
21. }
22. }

* 

**Happy New Year!  
Happy Holidays!**

**(Correct)**

* 

**null  
null**

* 

**null  
Happy Holidays!**

* 

**Happy Holidays!  
Happy Holidays!**

**Explanation**

Message class doesn't specify any constructor, hence Java compiler adds below default constructor:

Message() {super();}

Line n1 creates an instance of Message class and initializes instance variable 'msg' to "Happy New Year!". Variable 'obj' refers to this instance.

Line n2 prints Happy New Year! on to the console.

Line n3 invokes change(Message) method, as it is a static method defined in TestMessage class, hence `change(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 'm' copies the content of variable 'obj' (which stores the address to Message instance created at Line n1). 'm' also refers to the same instance referred by 'obj'.

Line n6, assigns "Happy Holidays!" to the 'msg' variable of the instance referred by 'm'. As 'obj' and 'm' refer to the same instance, hence obj.msg also refers to "Happy Holidays!". change(Message) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg refers to "Happy Holidays!", so this statement prints Happy Holidays! on to the console.

Hence in the output, you get:

Happy New Year!

Happy Holidays!

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. int score = 60;
6. switch (score) {
7. default:
8. System.out.println("Not a valid score");
9. case score < 70:
10. System.out.println("Failed");
11. break;
12. case score >= 70:
13. System.out.println("Passed");
14. break;
15. }
16. }
17. }

* 

**Compilation error**

**(Correct)**

* 

**Not a valid score**

**Failed**

* 

**Failed**

* 

**Passed**

**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 (score)] is of int type.

But case expressions, score < 70 and score >= 70 are of boolean type and hence compilation error.

Bottom of Form

Top of Form

Question 66: **Correct**

Consider below code:

1. //Test.java
2. public class Test {
3. private static boolean flag = !true;
5. public static void main(String [] args) {
6. System.out.println(!flag ? args[0] : args[1]);
7. }
8. }

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

javac Test.java  
java Test AM PM

* 

**ExceptionInInitializerError is thrown while loading the Test class**

* 

**PM**

* 

**AM**

**(Correct)**

* 

**Compilation error**

**Explanation**

There is no compilation error. When Test class is loaded by JVM to invoked main(String []) method, static variable declaration and initialization statement is executed and false is assigned to flag as !true is false.

As java Test AM PM command is used, so args[0] --> "AM" and args[1] --> "PM".

In ternary operator, boolean expression is evaluated first, !flag evaluates to true and therefore agrs[0] is returned.

AM is printed on to the console.

Bottom of Form

Top of Form

Question 67: **Correct**

Consider below code:

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<StringBuilder> days = new ArrayList<>();
9. days.add(new StringBuilder("Sunday"));
10. days.add(new StringBuilder("Monday"));
11. days.add(new StringBuilder("Tuesday"));
13. if(days.contains(new StringBuilder("Sunday"))) {
14. days.add(new StringBuilder("Wednesday"));
15. }
17. System.out.println(days.size());
18. }
19. }

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

* 

**Compilation error**

* 

**3**

**(Correct)**

* 

**Runtime exception**

* 

**4**

**Explanation**

StringBuilder class doesn't override equals(Object) method and hence days.contains(new StringBuilder("Sunday")) returns false.

Code inside if-block is not executed and days.size() returns 3.

Bottom of Form

Top of Form

Question 68: **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. }
20. public boolean equals(Object obj) {
21. if(obj instanceof Student) {
22. Student stud = (Student)obj;
23. if(this.name.equals(stud.name) && this.age == stud.age) {
24. return true;
25. }
26. }
27. return false;
28. }
29. }
31. public class Test {
32. public static void main(String[] args) {
33. List<Student> students = new ArrayList<>();
34. students.add(new Student("James", 25));
35. students.add(new Student("James", 27));
36. students.add(new Student("James", 25));
37. students.add(new Student("James", 25));
39. students.remove(new Student("James", 25));
41. for(Student stud : students) {
42. System.out.println(stud);
43. }
44. }
45. }

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

* 

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

**(Correct)**

* 

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

* 

**Student[James, 27]**

* 

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

**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 has been overridden by the Student class and equates the object based on their name and age.

3 matching Student objects are found in the list and 1st list element is removed from the list. Remaining 3 list elements are printed in the insertion order.

Bottom of Form

Top of Form

Question 69: **Correct**

Consider below code:

1. //Guest.java
2. class Message {
3. public static void main(String [] args) {
4. System.out.println("Welcome " + args[0] + "!");
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 James Gosling

What is the result?

* 

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

* 

**Welcome James!**

**(Correct)**

* 

**Welcome Gosling!**

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

**Explanation**

Both the classes contain special main method. No compilation error with the code: file is correctly names as Guest.java (name of public class).

java Guest James Gosling passes new String [] {"James", "Gosling"} to args of Guest.main method. Apart from being special main method, Message.main is static method so Guest.main method invokes Message.main method with the same argument: new String [] {"James", "Gosling"}.

args[0] is "James" hence "Welcome James!" gets printed on to the console.

Bottom of Form

Top of Form

Question 70: **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. Boolean b1 = new Boolean("tRuE");
6. Boolean b2 = new Boolean("fAlSe");
7. Boolean b3 = new Boolean("abc");
8. Boolean b4 = null;
9. System.out.println(b1 + ":" + b2 + ":" + b3 + ":" + b4);
10. }
11. }

* 

**Compilation error**

* 

**false:false:true:null**

* 

**false:false:false:null**

* 

**true:false:false:null**

**(Correct)**

**Explanation**

Boolean class code uses equalsIgnoreCase method to validate the passed String, so if passed String is "true" ('t', 'r', 'u' and 'e' can be in any case), then boolean value stored in Boolean object is true otherwise false.

b1 stores true, b2 stores false, b3 stores false and as b4 is of reference type, hence it can store null as well.

Output is: true:false:false:null

Bottom of Form

Retake test

**Continue**