Top of Form

Question 1: **Correct**

1. public class Test {
2. private static int [] arr;
3. public static void main(String [] args) {
4. if(arr.length > 0 && arr != null) {
5. System.out.println(arr[0]);
6. }
7. }
8. }

Predict Output, if the above code is run with given command?

java Test

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

* 

**No Output**

* 

**NullPointerException is thrown at runtime**

**(Correct)**

* 

**Compilation error**

**Explanation**

Variable arr is a class variable of int [] type, so by default it is initialized to null.

In if block, arr.length > 0 is checked first. Accessing length property on null reference throws NullPointerException.

Correct logical if block declaration should be:

if(arr != null && arr.length > 0)

First check for null and then access properties/methods.

Bottom of Form

Top of Form

Question 2: **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> fruits = new ArrayList<>();
9. fruits.add("apple");
10. fruits.add("orange");
11. fruits.add("grape");
12. fruits.add("mango");
13. fruits.add("banana");
14. fruits.add("grape");
16. if(fruits.remove("grape"))
17. fruits.remove("apple");
19. System.out.println(fruits);
20. }
21. }

* 

**[orange, grape, mango, banana]**

* 

**An exception is thrown at runtime**

* 

**[orange, mango, banana, grape]**

**(Correct)**

* 

**Compilation error**

**Explanation**

remove(Object) method of List interface removes the first occurrence of the specified element from the list, if it is present. If this list does not contain the element, it is unchanged. remove(Object) method returns true, if removal was successful otherwise false.

Initially list has: [apple, orange, grape, mango, banana, grape]. fruits.remove("grape") removes the first occurrence of "grape" and after the successful remove, list has: [apple, orange, mango, banana, grape]. fruits.remove("grape") returns true, control goes inside if block and executes fruits.remove("apple");

fruits list contains "apple", so after the removal list has: [orange, mango, banana, grape].

Bottom of Form

Top of Form

Question 3: **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("X");
24. } catch (IOException e) {
25. System.out.print("Y");
26. } finally {
27. System.out.print("Z");
28. }
29. }
30. }

* 

**YZ**

* 

**Compilation Error**

* 

**XYZ**

* 

**XZ**

**(Correct)**

**Explanation**

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

A catch handler for FileNotFoundException is available and hence X is printed on to the console.

After that finally block is executed, which prints Z to the console.

Bottom of Form

Top of Form

Question 4: **Incorrect**

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

1. package com.udayan.oca;
3. public class Test {
4. static String msg; //Line 2
5. public static void main(String[] args) {
6. String msg; //Line 4
7. if(args.length > 0) {
8. msg = args[0]; //Line 6
9. }
10. System.out.println(msg); //Line 8
11. }
12. }

* 

**Line 8 causes compilation failure**

**(Correct)**

* 

**An exception is thrown at runtime by Line 6**

* 

**Line 4 causes compilation failure**

**(Incorrect)**

* 

**Line 2 causes compilation failure**

* 

**null**

**Explanation**

Line 4 code shadows the variable at Line 2. msg variable created at Line 4 is a local variable and should be initialized before it is used.

Initialization code is inside if-block, so compiler is not sure about msg variable's initialization. Hence, Line 8 causes compilation failure.

Bottom of Form

Top of Form

Question 5: **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 i;
6. for(i=0; i<=2; i++){}
7. System.out.println(i);
8. }
9. }

* 

**0**

* 

**3**

**(Correct)**

* 

**2**

* 

**Compilation error**

**Explanation**

There is nothing inside loop body, hence loop body is blank.

This loop executes 3 times, for i = 0, i = 1 and i = 2. For i = 3, control goes out of the for loop.

Now, as i is declared outside for loop, hence it is accessible outside loop body.

System.out.println(i); prints 3 to the console.

Bottom of Form

Top of Form

Question 6: **Correct**

Given code:

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

What is the result?

* 

**NullPointerException is thrown at runtime**

* 

**true:true**

* 

**false:false**

* 

**null:null**

**(Correct)**

**Explanation**

Array elements are initialized to their default values.

arr is referring to an array of Boolean type, which is reference type and hence both the array elements are initialized to null and therefore in the output null:null is printed.

Bottom of Form

Top of Form

Question 7: **Correct**

For the code below, what should be the name of java file?

1. package com.udayan.oca;
3. public class HelloWorld {
4. public static void main(String [] args) {
5. System.out.println("Hello World!");
6. }
7. }

* 

**HelloWorld.java**

**(Correct)**

* 

**World.java**

* 

**Hello.java**

* 

**helloworld.java**

**Explanation**

Java is case sensitive language. File name should match with public class's name, which is "HelloWorld".

"helloworld" is different from "HelloWorld".

Bottom of Form

Top of Form

Question 8: **Correct**

Consider following code snippet:

1. package com.udayan.test;
3. public class Exam {
4. public static void main(String [] args) {
5. System.out.println("All the best!");
6. }
7. }

Location of files:

1. D:.
2. └───WORK
3. └───QUIZ
4. └───SEC07
5. ├───classes
6. │ └───com
7. │ └───udayan
8. │ └───test
9. │ Exam.class
10. │
11. └───src
12. └───com
13. └───udayan
14. └───test
15. Exam.java

You are currently at WORK folder.

D:\WORK>

Which of the following java command will show All the best! on to the console?

* 

**java -cp Quiz\Sec07\classes\ com.udayan.test.Exam**

**(Correct)**

* 

**java -cp Quiz\Sec07\classes\com\udayan\test\ Exam**

* 

**java com.udayan.test.Exam**

* 

**java Exam**

**Explanation**

To execute Exam class from WORK folder, you should specify the classpath (Quiz\Sec07\classes\) which contains whole path of the class(com\udayan\test\Exam.class).

And you should also use fully qualified name of the class, which is com.udayan.test.Exam.

Hence correct option is: java -cp Quiz\Sec07\classes\ com.udayan.test.Exam

Bottom of Form

Top of Form

Question 9: **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<String> days = new ArrayList<>();
9. days.add("SUNDAY");
10. days.add("SUNDAY");
11. days.add("MONDAY");
12. System.out.println(days.size());
13. days.clear();
14. System.out.println(days.size());
15. }
16. }

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

* 

**3  
3**

* 

**An exception is thrown at runtime**

* 

**2  
0**

* 

**3  
0**

**(Correct)**

**Explanation**

ArrayList can have duplicate elements, so after addition, list is: [SUNDAY, SUNDAY, MONDAY]. days.size() returns 3 so 3 is printed on to the console.

days.clear(); removes all the elements from the days list, in fact days list will be empty after successful execution of days.clear();

So 2nd System.out.println statement prints 0 on to the console.

Bottom of Form

Top of Form

Question 10: **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 obj = LocalDate.now();
9. System.out.println(obj.getHour());
10. }
11. }

Which of the following statement is correct?

* 

**It will print any int value between 1 and 24**

* 

**Code fails to compile**

**(Correct)**

* 

**It will print any int value between 0 and 23**

* 

**Code compiles successfully but throws Runtime exception**

**Explanation**

LocalDate.now(); retrieves the current date from the system clock. There is no issue with this statement.

obj is of LocalDate type and getHour() method is not defined in LocalDate class, it is defined in LocalTime and LocalDateTime class. Hence obj.getHour() causes compilation failure.

Bottom of Form

Top of Form

Question 11: **Correct**

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

1. package com.udayan.oca;
3. public class Test {
4. private static String s;
5. public static void main(String[] args) {
6. try {
7. System.out.println(s.length());
8. } catch(NullPointerException | RuntimeException ex) {
9. System.out.println("DONE");
10. }
11. }
12. }

* 

**Executes successfully but no output**

* 

**DONE**

* 

**Compilation error**

**(Correct)**

* 

**None of the above**

**Explanation**

NullPointerException extends RuntimeException and in multi-catch syntax we can't specify multiple Exceptions related to each other in multilevel inheritance.

Bottom of Form

Top of Form

Question 12: **Correct**

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

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. }

* 

**Integer var = 7;**

* 

**int var = '7';**

**(Correct)**

* 

**None of the other options**

* 

**int var = 7;**

**Explanation**

int var = 7; => DEFAULT,

Integer var = 7; => var is of Integer type and case contains char '7'. char '7' cannot be compared with Integer and hence compilation error. case '7' can easily be compared with int value but not with Integer type.

int var = '7'; => Lucky no. 7

HINT: There is no need to remember. case '7' value means you are trying to equate or compare var (Integer value) with '7' (char).

If assignment operation works then method invocation, switch expression parameter etc. will also work. Integer var = 7; is possible but Integer var = '7'; causes compilation error as char cannot be converted to Integer.

Bottom of Form

Top of Form

Question 13: **Correct**

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

1. package com.udayan.oca;
3. public class Wall {
4. public static void main(String args[]) {
5. double area = 5.7;
6. String color;
7. if (area < 7)
8. color = "green";
10. System.out.println(color);
11. }
12. }

* 

**NullPointerException**

* 

**null**

* 

**Compilation error**

**(Correct)**

* 

**green**

**Explanation**

color is LOCAL variable and it must be initialized before it can be used.

As area is not compile time constant, java compiler doesn't have an idea of the value of variable area.

There is no else block available as well.

So compiler cannot be sure of whether variable color will be initialized or not, therefore System.out.println(color); causes compilation error.

Bottom of Form

Top of Form

Question 14: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
5. import java.time.format.DateTimeFormatter;
7. public class Test {
8. public static void main(String [] args) {
9. LocalDate date1 = LocalDate.parse("1947-08-15", DateTimeFormatter.ISO\_DATE);
10. LocalDate date2 = LocalDate.parse("1947-08-15",
11. DateTimeFormatter.ISO\_LOCAL\_DATE);
12. LocalDate date3 = LocalDate.of(1947, 8, 15);
14. System.out.println(date1.equals(date2) + " : " + date2.equals(date3));
15. }
16. }

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

* 

**false : true**

* 

**Runtime exception**

* 

**true : true**

**(Correct)**

* 

**false : false**

* 

**true : false**

**Explanation**

ISO\_LOCAL\_DATE formatter formats the date without the offset, such as "1947-08-15".

ISO\_DATE formatter formats the date with offset (if available), such as "1947-08-15" or "1947-08-15+05:30", but remember LocalDate object doesn't contain any offset information.

In this case, all the three date instances are meaningfully equal.

For the OCA exam, you can check following DateTimeFormatter types: BASIC\_ISO\_DATE, ISO\_DATE, ISO\_LOCAL\_DATE, ISO\_TIME, ISO\_LOCAL\_TIME, ISO\_DATE\_TIME, ISO\_LOCAL\_DATE\_TIME.

Bottom of Form

Top of Form

Question 15: **Correct**

Consider below code:

1. package com.udayan.oca;
3. public class Test {
5. private static void add(int i, int j) {
6. System.out.println("int version");
7. }
9. private static void add(Integer i, Integer j) {
10. System.out.println("Integer version");
11. }
13. public static void main(String[] args) {
14. add(10, 20);
15. }
17. }

Which modifications, done independently, print "Integer version" on to the console?

Select 3 options.

* 

**Replace add(10, 20);  by add(new Integer(10), new Integer(20));**

**(Correct)**

* 

**Replace add(10, 20);  by add(10.0, 20.0);**

* 

**Remove add(int i, int j)  method declaration and definition.**

**(Correct)**

* 

**Replace add(10, 20);  by add(null, null);**

**(Correct)**

**Explanation**

Method add is overloaded in Test class. Which overloaded method is invoked is decided at the compile time. add(10, 20); tags to int version as 10, 20 are int literals and direct match is available. So without any changes, above code prints "int version" on to the console.

To print "Integer version" on to the console, add(Integer, Integer); method needs to be invoked. Let's check all the options one by one:

"Remove add(int i, int j) method declaration and definition." add(10, 20); => auto-boxing will convert literal 10 and 20 to Integer instances and will call the add(Integer, Integer) method. So this option is valid.

Replace add(10, 20); by add(new Integer(10), new Integer(20)); => This statement is specifically calling add(Integer, Integer); So this option is also valid.

Replace add(10, 20); by add(10.0, 20.0);  10.0 and 20.0 are default literals and can't be mapped to int or Integer types, hence this gives compilation error. Not a valid option.

Replace add(10, 20); by add(null, null);  As Integer is reference type hence add(null, null); maps to add(Integer, Integer); So this is also a valid option.

Bottom of Form

Top of Form

Question 16: **Correct**

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(3);
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?

* 

**VET**

**(Correct)**

* 

**VETO**

* 

**VTEO**

* 

**VTE**

* 

**Compilation error**

* 

**Runtime error**

**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(3);.

list.remove(3); => Removes last element of the list. list --> [V,E,T].

for(char ch : list) => First list item is Character object, which is auto-unboxed and assigned to ch. This means in first iteration ch = 'V'; And after this it is simple enhanced for loop. Output is VET.

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. StringBuilder sb = new StringBuilder("Good"); //Line 3
6. change(sb); //Line 4
7. System.out.println(sb); //Line 5
8. }
10. private static void change(StringBuilder s) {
11. s.append("\_Morning"); //Line 9
12. }
13. }

* 

**\_Morning**

* 

**None of the other options**

* 

**Good**

* 

**Good\_Morning**

**(Correct)**

**Explanation**

When change method is called, both variable s and sb refers to same StringBuilder object.

Line 9 modifies the passed object and appends "\_Morning" to it. As a result s now refers to "Good\_Morning" and sb also refers to "Good\_Morning" so when control goes back to calling method main(String[]) Line 5 prints "Good\_Morning" on to the console.

Bottom of Form

Top of Form

Question 18: **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. abstract class Animal {}
7. class Dog extends Animal{}
9. public class Test {
10. public static void main(String [] args) {
11. List<Animal> list = new ArrayList<Dog>();
12. list.add(0, new Dog());
13. System.out.println(list.size() > 0);
14. }
15. }

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

* 

**true**

* 

**Compilation error**

**(Correct)**

* 

**false**

* 

**Runtime exception**

**Explanation**

List is super type and ArrayList is sub type, hence List l = new ArrayList(); is valid syntax.

Animal is super type and Dog is sub type, hence Animal a = new Dog(); is valid syntax. Both depicts Polymorphism.

But in generics syntax, Parameterized types are not polymorphic, this means ArrayList<Animal> is not super type of ArrayList<Dog>. Remember this point. So below syntaxes are not allowed:

ArrayList<Animal> list = new ArrayList<Dog>(); OR List<Animal> list = new ArrayList<Dog>();

Bottom of Form

Top of Form

Question 19: **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. char [][] arr = {
6. {'A', 'B', 'C'},
7. {'D', 'E', 'F'},
8. {'G', 'H', 'I'}
9. };
11. for(int i = 0; i < arr.length; i++) {
12. for(int j = 0; j < arr[i].length; j++) {
13. System.out.print(arr[i][1]);
14. }
15. System.out.println();
16. }
17. }
18. }

* 

**ABC  
DEF  
GHI**

* 

**BBB  
EEE  
HHH**

**(Correct)**

* 

**CCC  
FFF  
III**

* 

**AAA  
DDD  
GGG**

**Explanation**

NOTE: System.out.print statement is printing arr[i][1],

which means it prints 2nd array element of a particular row, for each iteration of inner loop.

That is why output is:

BBB

EEE

HHH

To get all the array elements printed correctly, use arr[i][j] in System.out.print statement.

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 {
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 Double(10.0));
15. }
17. }

* 

**double version: 20.0**

* 

**Compilation error**

**(Correct)**

* 

**An exception is thrown at runtime**

* 

**Double version: 20.0**

**Explanation**

add(10.0, new Double(10.0)); is an ambiguous call as compiler can't decide whether to convert 1st argument to Double reference type or 2nd argument to double primitive type.

So, add(10.0, new Double(10.0)); causes compilation error.

Bottom of Form

Top of Form

Question 21: **Correct**

How many String objects are there in the HEAP memory, when control is at Line 9?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String s1 = new String("Java"); //Line 3
6. String s2 = "JaVa"; //Line 4
7. String s3 = "JaVa"; //Line 5
8. String s4 = "Java"; //Line 6
9. String s5 = "Java"; //Line 7
11. int i = 1; //Line 9
13. }
14. }

* 

**4**

* 

**2**

* 

**3**

**(Correct)**

* 

**5**

**Explanation**

String s1 = new String("Java"); -> Creates 2 objects: 1 String Pool and 1 non-pool. s1 refers to non-pool object.

String s2 = "JaVa"; -> Creates 1 String pool object and s2 refers to it.

String s3 = "JaVa"; -> Doesn't create a new object, s3 refers to same String pool object referred by s2.

String s4 = "Java"; -> Doesn't create a new object, s4 refers to String Pool object created at Line 3.

String s5 = "Java"; -> Doesn't create a new object, s5 also refers to String Pool object created at Line 3.

So, at Line 9, 3 String objects are available in the HEAP memory: 2 String pool and 1 non-pool.

Bottom of Form

Top of Form

Question 22: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. public class Test {
5. public static void main(String[] args) {
6. String javaworld = "JavaWorld";
7. String java = "Java";
8. String world = "World";
9. java += world;
10. System.out.println(java == javaworld);
11. }
12. }

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

* 

**true**

* 

**World**

* 

**Java**

* 

**false**

**(Correct)**

* 

**JavaWorld**

**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.

`java += world;` is same as `java = java + world;` and `java + world` is not a constant expression and hence is calculated at runtime and returns a non pool String object "JavaWorld", which is referred by variable 'java'.

On the other hand, variable 'javaworld' refers to String Pool object "JavaWorld". As both the variables 'java' and 'javaworld' refer to different String objects, hence `java == javaworld` returns false.

Bottom of Form

Top of Form

Question 23: **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 = 3;
6. System.out.println(a++ == 3 || --a == 3 && --a == 3);
7. }
8. }

* 

**true**

**(Correct)**

* 

**Compilation error**

* 

**false**

**Explanation**

a++ == 3 || --a == 3 && --a == 3; [Given expression].

(a++) == 3 || --a == 3 && --a == 3; [Postfix has got higher precedence than other operators].

(a++) == 3 || (--a) == 3 && (--a) == 3; [After postfix, precedence is given to prefix].

((a++) == 3) || ((--a) == 3) && ((--a) == 3); [== has higher precedence over && and ||].

((a++) == 3) || (((--a) == 3) && ((--a) == 3)); [&& has higher precedence over ||].

Let's start solving it:

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

(3 == 3) || (((--a) == 3) && ((--a) == 3)); [a=4, res=false].

true || (((--a) == 3) && ((--a) == 3)); [a=4, res=false].  || is a short-circuit operator, hence no need to evaluate expression on the right.

res is true and a is 4.

Bottom of Form

Top of Form

Question 24: **Correct**

Wrapper classes are defined in which of the following package?

* 

**java.util**

* 

**default package**

* 

**java.lang**

**(Correct)**

* 

**java.io**

**Explanation**

All the wrapper classes are defined in java.lang package.

String and StringBuilder are also defined in java.lang package and that is why import statement is not required to use these classes.

Bottom of Form

Top of Form

Question 25: **Correct**

Consider below code:

1. package com.udayan.oca;
3. public class Counter {
4. int count;
6. private static void increment(Counter counter) {
7. counter.count++;
8. }
10. public static void main(String [] args) {
11. Counter c1 = new Counter();
12. Counter c2 = c1;
13. Counter c3 = null;
14. c2.count = 1000;
15. increment(c2);
16. }
17. }

On executing Counter class, how many Counter objects are created in the memory?

* 

**1**

**(Correct)**

* 

**2**

* 

**3**

* 

**4**

**Explanation**

new Counter(); is invoked only once, hence only one Counter object is created in the memory.

c1, c2, c3 and counter are reference variables of Counter type and not Counter objects.

Bottom of Form

Top of Form

Question 26: **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(0, 1000, 0);
9. System.out.println(period);
10. }
11. }

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

* 

**p1000m**

* 

**P1000M**

**(Correct)**

* 

**P0Y1000M0D**

* 

**p0y1000m0d**

**Explanation**

Check the toString() method of Period class. 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 27: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalTime;
6. public class Test {
7. public static void main(String [] args) {
8. LocalTime time = LocalTime.of(23, 60);
9. System.out.println(time);
10. }
11. }

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

* 

**00:01**

* 

**23:60**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**00:00**

**Explanation**

LocalTime.of(int hour, int minute) creates an instance of LocalTime class. Valid value for hour is: 0 to 23 and valid value for minute is 0 to 59.

java.time.DateTimeException is thrown if invalid values are passed as arguments.

NOTE: There are other overloaded of methods available:

LocalTime of(int hour, int minute, int second) and

LocalTime of(int hour, int minute, int second, int nanoOfSecond).

Valid value for second is: 0 to 59 and valid value for nano second is: 0 to 999,999,999.

Bottom of Form

Top of Form

Question 28: **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 fName = "James";
6. String lName = "Gosling";
7. System.out.println(fName = lName);
8. }
9. }

* 

**false**

* 

**None of the other options**

**(Correct)**

* 

**true**

* 

**Compilation error**

**Explanation**

Both fName and lName are of reference type. fName refers to "James" and lName refers to "Gosling".

In System.out.println() statement, we have used assignment operator (=) and not equality operator (==). So result is never boolean.

fName = lName means copy the contents of lName to fName.

As lName is referring to "Gosling" and so after the assignment, fName starts referring to "Gosling" as well.

System.out.println() finally prints the String referred by fName, which is "Gosling".

This option is is not available, hence correct answer is "None of the other options"

Bottom of Form

Top of Form

Question 29: **Correct**

Which of the following can be used as a constructor for the class given below?

1. public class Planet {
2. }

* 

**None of the other options**

* 

**public void Planet(){}**

* 

**public void Planet(int x){}**

* 

**public Planet(String str) {}**

**(Correct)**

**Explanation**

Constructor has the same name as the class, doesn't have return type and can accept parameters.

Bottom of Form

Top of Form

Question 30: **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 (true);
9. System.out.println("Bye");
10. }
11. }

* 

**Compiles successfully and prints "Bye"**

* 

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

* 

**100  
Bye**

* 

**Unreachable code compilation error**

**(Correct)**

**Explanation**

Boolean expression of do-while loop uses literal true (compile-time constant), hence Java compiler knows that this loop is an infinte loop.

It also knows that once at runtime Java Control enters an infinite loop, none of the statements after loop block will get executed.

Hence it marks all the codes after infinite loop as Unreachable Code, which results in compilation error.

If boolean variable was used instead of boolean literal, then this program would have compiled and executed successfully.

1. public class DoWhileTest1 {
2. public static void main(String[] args) {
3. boolean flag = true;
4. do {
5. System.out.println(100);
6. } while (flag);
8. System.out.println("Bye");
9. }
10. }

Above program prints 100 in infinite loop and "Bye" never gets printed.

Bottom of Form

Top of Form

Question 31: **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<>(4);
10. list.add(0, "Array");
11. list.add(2, "List");
13. System.out.println(list);
14. }
15. }

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

* 

**[Array, List]**

* 

**[Array, null, List, null]**

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**Compilation error**

**Explanation**

ArrayList are different than arrays, though behind the scene ArrayList uses Object[] to store its elements.

There are 2 things related to ArrayList, one is capacity and another is actual elements stored in the list, returned by size() method. If you don't pass anything to the ArrayList constructor, then default capacity is 10 but this doesn't mean that an ArrayList instance will be created containing 10 elements and all will be initialized to null.

In fact, size() method will still return 0 for this list. This list still doesn't contain even a single element. You need to use add method or its overloaded counterpart to add items to the list. Even if you want to add null values, you should still invoke some methods, nothing happens automatically.

In this question, new ArrayList<>(4); creates an ArrayList instance which can initially store 4 elements but currently it doesn't store any data.

Another point you should remember for the certification exam: Addition of elements in ArrayList should be continuous. If you are using add(index, Element) method to add items to the list, then index should be continuous, you simply can't skip any index.

In this case, list.add(0, "Array"); adds "Array" to 0th index. so after this operation list --> [Array].  You can now add at 0th index (existing elements will be shifted right) or you can add at index 1 but not at index 2. list.add(2, "List"); throws an instance of java.lang.IndexOutOfBoundsException.

Bottom of Form

Top of Form

Question 32: **Correct**

Consider below code:

1. package com.udayan.oca;
3. public class Test {
4. static Double d1;
5. int x = d1.intValue();
7. public static void main(String[] args) {
8. System.out.println("HELLO");
9. }
10. }

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

* 

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

**(Correct)**

* 

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

**Explanation**

To invoke the special main method, JVM loads the class in the memory. At that time, static fields of Test class are initialized. d1 is of Double type so null is assigned to it.

x is not static variable, so int x = d1.intValue(); is not executed. Class is loaded successfully in the memory and "HELLO" is printed on to the console.

NOTE: new Test() will throw NullPointerException but not ExceptionInInitializerError.

Bottom of Form

Top of Form

Question 33: **Correct**

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. }

There is a compilation error in the Student class.

Which modifications, done independently, print "James:25" on to the console?

Select 2 options.

* 

**Replace Student("James", 25);  with this("James", 25);**

**(Correct)**

* 

**Replace Student("James", 25);  with super("James", 25);**

* 

**Add below code in the Student class:**

* 1. void Student(String name, int age) {
  2. this.name = name;
  3. this.age = age;
  4. }

**(Correct)**

* 

**Replace Student("James", 25);  with this.Student("James", 25);**

**Explanation**

First find out the reason for compilation error, all the options are giving hint :)

no-arg constructor of Student class calling another overloaded constructor by the name and this causes compilation error. This problem can be fixed in 2 ways:

1st one: replace Student("James", 25); with this("James", 25) OR 2nd one: add void Student(String, int) method in the Student class.

Method can have same name as the class name and constructor can call other methods.

Bottom of Form

Top of Form

Question 34: **Correct**

Which of the method of String class is used to remove leading and trailing white spaces?

* 

**trim()**

**(Correct)**

* 

**trimBoth()**

* 

**rtrim()**

* 

**ltrim()**

**Explanation**

ltrim(), rtrim() and trimBoth() are not defined in String class.

trim() method is used for removing leading and trailing white spaces.

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();
6. System.out.println(sb.append("").append("").append("").length());
7. }
8. }

* 

**0**

**(Correct)**

* 

**2**

* 

**3**

* 

**1**

**Explanation**

As "" is empty string, hence nothing is appended to the StringBuilder instance and length() method returns 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. int a = 7;
6. boolean res = a++ == 7 && ++a == 9 || a++ == 9;
7. System.out.println("a = " + a);
8. System.out.println("res = " + res);
9. }
10. }

* 

**a = 10  
res = true**

* 

**a = 9  
res = true**

**(Correct)**

* 

**Compilation error**

* 

**a = 10  
res = false**

**Explanation**

boolean res = a++ == 7 && ++a == 9 || a++ == 9;

= (a++) == 7 && ++a == 9 || (a++) == 9;

= (a++) == 7 && (++a) == 9 || (a++) == 9;

= ((a++) == 7) && ((++a) == 9) || ((a++) == 9);

= ((a++) == 7) && ((++a) == 9) || ((a++) == 9);

= (((a++) == 7) && ((++a) == 9)) || ((a++) == 9);

= ((7 == 7) && ((++a) == 9)) || ((a++) == 9); //a = 8

= (true && ((++a) == 9)) || ((a++) == 9); //a = 8

= (true && (9 == 9)) || ((a++) == 9); //a = 9

= (true && true) || ((a++) == 9); //a = 9

= true || ((a++) == 9); //a = 9

= true; //a = 9

So,

a = 9

res = true

Bottom of Form

Top of Form

Question 37: **Correct**

Consider below code of Test.java file:

1. package com.udayan.oca;
3. class Document {
4. int pages;
5. Document(int pages) {
6. this.pages = pages;
7. }
8. }
10. class Word extends Document {
11. String type;
12. Word(String type) {
13. super(20); //default pages
14. /\*INSERT-1\*/
15. }
17. Word(int pages, String type) {
18. /\*INSERT-2\*/
19. super.pages = pages;
20. }
21. }
23. public class Test {
24. public static void main(String[] args) {
25. Word obj = new Word(25, "TEXT");
26. System.out.println(obj.type + "," + obj.pages);
27. }
28. }

Currently above code causes compilation error.

Which of the options can successfully print TEXT,25 on to the console?

* 

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

**super.type = type;**

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

**super(type);**

* 

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

**this.type = type;**

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

**this(type);**

**(Correct)**

* 

**None of the other options**

* 

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

**this(type);**

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

**this.type = type;**

* 

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

**super.type = type;**

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

**this(type);**

**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 Word's constructor: Word(int pages, String type) { super(); } but Document class doesn't have a no-argument constructor and that is why Word's constructor `Word(int pages, String type)` causes compilation error.

Word(String) constructor is actually not setting the passed type argument. Replace /\*INSERT-1\*/ with: `this.type = type;` will set the value to type variable.

As the first statement inside Word(int pages, String type){} constructor, you can either have `super(pages);` or `this(type);` but not both.

Replacing /\*INSERT-2\*/ with `super(pages);` will be redundant as in the next statement `super.pages = pages;`, pages variable of Document class is set. Hence, replacing /\*INSERT-2\*/ with `this(type);` is needed to set the type variable.

Bottom of Form

Top of Form

Question 38: **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(1 + 2 + 3 + 4 + "Hello");
6. }
7. }

* 

**10Hello**

**(Correct)**

* 

**64Hello**

* 

**1234Hello**

* 

**10 Hello**

**Explanation**

As expression contains + operator only, which is left to right associative. Let us group the expression.

1 + 2 + 3 + 4 + "Hello"

= (1 + 2) + 3 + 4 + "Hello"

= ((1 + 2) + 3) + 4 + "Hello"

= (((1 + 2) + 3) + 4) + "Hello"

[Let us solve it now,]

= ((3 + 3) + 4) + "Hello"

= (6 + 4) + "Hello"

= 10 + "Hello"

[+ operator with String behaves as concatenation operator.]

= 10Hello

Bottom of Form

Top of Form

Question 39: **Incorrect**

What will be the result of compiling and executing Circus

1. //Circus.java
2. package com.udayan.oca;
4. class Animal {
5. protected void jump() {
6. System.out.println("Animal");
7. }
8. }
10. class Cat extends Animal {
11. public void jump(int a) {
12. System.out.println("Cat");
13. }
14. }
16. class Deer extends Animal {
17. public void jump() {
18. System.out.println("Deer");
19. }
20. }
22. public class Circus {
23. public static void main(String[] args) {
24. Animal cat = new Cat();
25. Animal deer = new Deer();
26. cat.jump();
27. deer.jump();
28. }
29. }

* 

**Cat  
Deer**

* 

**Animal  
Animal**

**(Incorrect)**

* 

**Animal  
Deer**

**(Correct)**

* 

**Cat  
Animal**

**Explanation**

Cat class doesn't override the jump() method of Animal class, in fact jump(int) method is overloaded in Cat class.

Deer class overrides jump() method of Animal class.

Reference variable cat is of Animal type, cat.jump() syntax is fine and as Cat doesn't override jump() method hence Animal version is invoked, which prints Animal to the console.

Even though reference variable deer is of Animal type but at runtime deer.jump(); invokes overriding method of Deer class, this prints Deer to the console.

Bottom of Form

Top of Form

Question 40: **Correct**

Which of the following is the correct package declaration to declare Test class in com.exam.oca package?

* 

**package com.exam.oca;**

**(Correct)**

* 

**package com.exam.oca.\*;**

* 

**package com.exam.oca.Test;**

* 

**Package com.exam.oca;**

**Explanation**

To declare Test class in com.exam.oca package, use following declaration:

package com.exam.oca;

No wildcard (\*) allowed in package declaration. Don't include class name in package declaration.

NOTE: all small case letters in package keyword.

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. int [] arr1 = {1, 2, 3};
6. char [] arr2 = {'A', 'B'}; //ASCII code of 'A' is 65, 'B' is 66
7. arr1 = arr2;
8. for(int i = 0; i < arr1.length; i++) {
9. System.out.print(arr1[i] + " ");
10. }
11. }
12. }

* 

**A B**

* 

**1 2 3**

* 

**65 66**

* 

**Compilation error**

**(Correct)**

**Explanation**

Initially arr1 refers to an int array object of 3 elements: 1, 2, 3

And arr2 refers to an char array object of 2 elements: 'A', 'B'.

Statement arr1 = arr2; gives compilation error as char [] is not compatible with int [] even though char is compatible with int.

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 date1 = LocalDate.parse("1980-03-16");
9. LocalDate date2 = LocalDate.parse("1980-03-16");
10. System.out.println(date1.equals(date2) + " : " + date1.isEqual(date2));
11. }
12. }

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

* 

**true : true**

**(Correct)**

* 

**false : false**

* 

**false : true**

* 

**true : false**

**Explanation**

Both the methods "public boolean isEqual(ChronoLocalDate)" and "public boolean equals(Object)" return true if date objects are equal otherwise false.

NOTE: LocalDate implements ChronoLocalDate.

Bottom of Form

Top of Form

Question 43: **Correct**

Which of the following method is declared in Predicate<T> interface?

* 

**boolean check(T t);**

* 

**boolean verify(T t);**

* 

**boolean test(T t);**

**(Correct)**

* 

**boolean validate(T t);**

**Explanation**

Single abstract method declared in Predicate<T> interface is boolean test(T t);

NOTE: If you are confused, then check other questions on Predicate and from there you will know about the method declared in Predicate interface.

Bottom of Form

Top of Form

Question 44: **Correct**

Which of the following will give you current system time?

Select 2 options.

* 

**System.out.println(new LocalDate());**

* 

**System.out.println(new LocalDateTime());**

* 

**System.out.println(LocalDateTime.now());**

**(Correct)**

* 

**System.out.println(new LocalTime());**

* 

**System.out.println(LocalTime.now());**

**(Correct)**

* 

**System.out.println(LocalDate.now());**

**Explanation**

new LocalDate(), new LocalTime() and new LocalDateTime() give compilation error as constructor of these classes are declared private.

System.out.println(LocalDate.now()); => Prints current date only.

System.out.println(LocalTime.now()); => Prints current time only.

System.out.println(LocalDateTime.now()); => Prints current date and time both.

Bottom of Form

Top of Form

Question 45: **Correct**

Which of the following statement is correct for below code?

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. final boolean flag;
6. flag = false;
7. while(flag) {
8. System.out.println("Good Morning!");
9. }
10. }
11. }

* 

**Program compiles and executes successfully but produces no output.**

**(Correct)**

* 

**Infinite loop.**

* 

**Compilation error.**

* 

**It will print "Good Morning!" once.**

**Explanation**

final boolean flag; flag = false; doesn't make flag a compile time constant.

Compiler doesn't know flag's value at compile-time and hence it allows this syntax.

At runtime, as boolean expression of while loop is false, loop doesn't execute even once and hence no output.

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. extractInt(2.7);
6. extractInt(2);
7. }
9. private static void extractInt(Double obj) {
10. System.out.println(obj.intValue());
11. }
12. }

* 

**2  
2**

* 

**Compilation error in main method**

**(Correct)**

* 

**An exception is thrown at runtime**

* 

**3  
2**

* 

**Compilation error in extractInt method**

**Explanation**

extractInt method accepts argument of Double type.

extractInt(2.7); => 2.7 is double literal, so Java compiler would box it into Double type. At runtime obj.intValue() would print int portion of the Double data, which is 2.

extractInt(2); => Java compiler either does implicit casting or Wrapping but not both. 2 is int literal, Java compiler can't implicit cast it to double and then box it to Double. So this statement causes compilation failure.

Bottom of Form

Top of Form

Question 47: **Correct**

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

1. package com.udayan.oca;
3. public class DivModTest {
4. public static void main(String[] args) {
5. System.out.println( 23 / 2.0 );
6. System.out.println( 23 % 2.0 );
7. }
8. }

* 

**11.5  
0.0**

* 

**11.5  
1.0**

**(Correct)**

* 

**11.0  
1.0**

* 

**11  
1**

**Explanation**

As floating point numbers are used in the expression, hence result should be in floating point number.

Correct result is:

23 / 2.0 = 11.5

23 % 2.0 = 1.0

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 print() {
5. System.out.println("static method");
6. }
8. public static void main(String[] args) {
9. Test obj = null;
10. obj.print();
11. }
12. }

* 

**static method**

**(Correct)**

* 

**NullPointerException is thrown**

* 

**Compilation error**

* 

**None of the other options**

**Explanation**

print() is static method of class Test. So correct syntax to call method print() is Test.print();

but static methods can also be invoked using reference variable: obj.print(); Warning is displayed in this case.

Even though obj has null value, we don't get NullPointerException as objects are not needed to call static methods.

Bottom of Form

Top of Form

Question 49: **Correct**

Given the following definitions of the class Insect and the interface Flyable, the task is to declare a class Mosquito that inherits from the class Insect and implements the interface Flyable.

1. class Insect {}
2. interface Flyable {}

Select the correct option to accomplish this task:

* 

**class Mosquito extends Insect implements Flyable{}**

**(Correct)**

* 

**class Mosquito implements Insect, Flyable{}**

* 

**class Mosquito extends Insect, Flyable{}**

* 

**class Mosquito implements Insect extends Flyable{}**

**Explanation**

A class in Java extends class and implements interface.

Bottom of Form

Top of Form

Question 50: **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. Boolean [] arr = new Boolean[2];
10. List<Boolean> list = new ArrayList<>();
11. list.add(arr[0]);
12. list.add(arr[1]);
14. if(list.remove(0)) {
15. list.remove(1);
16. }
18. System.out.println(list);
19. }
20. }

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

* 

**[true]**

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

* 

**Compilation error**

**(Incorrect)**

* 

**[false]**

* 

**NullPointerException is thrown at runtime**

**(Correct)**

* 

**[]**

**Explanation**

Default values are assigned to all array elements. As Boolean is of reference type, hence arr[0] = null and arr[1] = null. After addition list contains [null, null].

list.remove(0) removes and returns the Boolean object referring to null. If expression can specify Boolean type, so no compilation error over here. At this point list contains [null].

For the boolean expression of if-block, Java runtime tries to extract the stored boolean value using booleanValue() method, and this throws an instance of NullPointerException as booleanValue() method is invoked on null reference.

Bottom of Form

Top of Form

Question 51: **Correct**

Consider below code:

1. public class Test {
2. public static void main(String[] args) {
3. System.out.println("ONE");
4. }
6. public static void main(Integer[] args) {
7. System.out.println("TWO");
8. }
10. public static void main(byte [] args) {
11. System.out.println("THREE");
12. }
13. }

What will be the result if Test class is executed by below command?

java Test 10

* 

**ONE**

**(Correct)**

* 

**THREE**

* 

**Compilation error**

* 

**TWO**

**Explanation**

Like any other method, main method can also be overloaded. But main method called by JVM is always with String [] parameter.

Don't get confused with 10 as it is passed as "10". Run above class with any command line arguments or 0 command line argument, output will always be ONE.

Bottom of Form

Top of Form

Question 52: **Correct**

Consider codes of 3 java files:

1. //Class1.java
2. package com.udayan.oca;
4. import java.io.FileNotFoundException;
6. public class Class1 {
7. public void read() throws FileNotFoundException {}
8. }
9. //Class2.java
10. package com.udayan.oca;
12. public class Class2 {
13. String Class2;
14. public void Class2() {}
15. }
16. //Class3.java
17. package com.udayan.oca;
19. public class Class3 {
20. private void print() {
21. private String msg = "HELLO";
22. System.out.println(msg);
23. }
24. }

Which of the following statement is true?

* 

**Only Class1.java compiles successfully**

* 

**Class2.java and Class3.java compile successfully**

* 

**Class1.java and Class2.java compile successfully**

**(Correct)**

* 

**Only Class2.java compiles successfully**

* 

**Only Class3.java compiles successfully**

* 

**Class1.java and Class3.java compile successfully**

**Explanation**

Method declaring checked exception in its throws clause doesn't mean that it should have code to actually throw that type of Exceptions.

So even though read() method of Class1 declares to throw FileNotFoundException but its body doesn't actually throw an instance of FileNotFoundException.

Variable and method name can be same as class name, so code of Class2 is also valid.

Remember: Though you don't get any compilation error but it is not recommended to use the Class name for variable and method names.

LOCAL variable can be declared with final modifier only.

msg variable inside print() method of Class3 is declared private and this causes compilation error.

Bottom of Form

Top of Form

Question 53: **Correct**

**What is the signature of special main method?**

* 

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

**(Correct)**

* 

**public static void main() {}**

* 

**public static void main(String args) {}**

* 

**private static void main(String [] args) {}**

**Explanation**

Special main method should have public access specifier and it takes argument of String [] type.

String [] argument can use any identifier name, even though in most of the cases you will see "args" is used.

Bottom of Form

Top of Form

Question 54: **Correct**

Given code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. byte b1 = (byte) (127 + 21);
6. System.out.println(b1);
7. }
8. }

What is the result?

* 

**Compilation error**

* 

**-108**

**(Correct)**

* 

**-128**

* 

**148**

**Explanation**

127 + 21 = 148 = 00000000 00000000 00000000 10010100

Above binary number is +ve, as left most bit is 0.

Same binary number after type-casting to byte: 10010100, -ve number as left most bit is 1.

10010100 = -108.

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.function.Predicate;
5. public class Test {
6. public static void main(String[] args) {
7. String [] arr = {"\*", "\*\*", "\*\*\*", "\*\*\*\*", "\*\*\*\*\*", "\*\*\*\*\*\*"};
8. Predicate<String> 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. }

* 

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

**(Correct)**

* 

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

* 

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

* 

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

**Explanation**

Lambda expression for Predicate is: s -> s.length() < 4. This means return true if passed string's length is < 4.

So first three array elements are printed.

Bottom of Form

Top of Form

Question 56: **Correct**

DateTimeFormatter is defined inside which package?

* 

**java.time**

* 

**java.time.format**

**(Correct)**

* 

**java.util**

* 

**java.text**

**Explanation**

DateTimeFormatter is a part of "java.time.format" package, whereas LocalDate, LocalTime, LocalDateTime and Period are defined inside "java.time" package.

Bottom of Form

Top of Form

Question 57: **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 i;
6. outer:
7. do {
8. i = 5;
9. inner:
10. while (true) {
11. System.out.println(i--);
12. if (i == 4) {
13. break outer;
14. }
15. }
16. } while (true);
17. }
18. }

* 

**Prints 5 once.**

**(Correct)**

* 

**Compilation error.**

* 

**Prints 5 in an infinite loop.**

* 

**5  
3  
2  
1**

**Explanation**

"outer" and "inner" are valid label names.

On execution, control enters main method and creates int variable i.

On encountering do-while loop, control goes inside and initializes variable i to 5.

Then it executes while loop and it's boolean expression is always true.

System.out.println(i--); prints 5 to the console first, and then decrements the value of i by 1. So, i becomes 4.

Boolean expression of if(i == 4) evaluates to true. break outer; statement executes and takes the control out of do-while loop.

main method ends and program terminates successfully.

So, 5 gets printed only once.

Bottom of Form

Top of Form

Question 58: **Correct**

Given Code:

1. package com.udayan.oca;
3. import java.io.\*;
5. class ReadTheFile {
6. static void print() { //Line 4
7. throw new IOException(); //Line 5
8. }
9. }
11. public class Test {
12. public static void main(String[] args) { //Line 10
13. ReadTheFile.print(); //Line 11
14. //Line 12
15. }
16. }

Which 2 changes are necessary so that code compiles successfully?

* 

**Surround Line 11 with below try-catch block:**

* 1. try {
  2. ReadTheFile.print();
  3. } catch(Exception e) {
  4. e.printStackTrace();
  5. }

**(Correct)**

* 

**Surround Line 11 with below try-catch block:**

* 1. try {
  2. ReadTheFile.print();
  3. } catch(IOException e) {
  4. e.printStackTrace();
  5. }
* 

**Surround Line 11 with below try-catch block:**

* 1. try {
  2. ReadTheFile.print();
  3. } catch(IOException | Exception e) {
  4. e.printStackTrace();
  5. }
* 

**Replace Line 4 with static void print() throws Throwable {**

* 

**Replace Line 10 with public static void main(String[] args) throws IOException {**

* 

**Replace Line 4 with static void print() throws Exception {**

**(Correct)**

**Explanation**

This question is tricky as 2 changes are related and not independent. Let's first check the reason for compilation error. Line 5 throws a checked exception, IOException but it is not declared in the throws clause. So, print method should have throws clause for IOException or the classes in top hierarchy such as Exception or Throwable.

Based on this deduction, Line 4 can be replaced with either "static void print() throws Exception {" or "static void print() throws Throwable" but we will have to select one out of these as after replacing Line 4, Line 11 will start giving error as we are not handling the checked exception at Line 11.

This part is easy, do we have other options, which mention "Throwable"? NO. Then mark the first option as "Replace Line 4 with static void print() throws Exception {".

As, print() method throws Exception, so main method should handle Exception or its super type and not it's subtype. Two options working only with IOException can be ruled out.

Multi-catch statement "catch(IOException | Exception e)" gives compilation error as IOException and Exception are related to each other in multilevel inheritance. So you are left with only one option to pair with our 1st choice:

Surround Line 11 with below try-catch block:

1. try {
2. ReadTheFile.print();
3. } catch(Exception e) {
4. e.printStackTrace();
5. }

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. class A {
4. A() {
5. this(1);
6. System.out.println("M");
7. }
9. A(int i) {
10. System.out.println("N");
11. }
12. }
14. class B extends A {
16. }
18. public class Test {
19. public static void main(String[] args) {
20. new B();
21. }
22. }

* 

**N  
M**

**(Correct)**

* 

**M**

* 

**N**

* 

**M  
N**

**Explanation**

Default constructor added by Java compiler in B class is:

1. B() {
2. super();
3. }

On executing new B(); statement, class B's default constructor is invoked, which invokes no-arg constructor of class A [super();].

no-arg constructor of class A invokes parameterized constructor of class A [this(1);].

N is printed first and after that M is printed.

Bottom of Form

Top of Form

Question 60: **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. Integer i = 10;
9. List<Integer> list = new ArrayList<>();
10. list.add(i);
11. list.add(new Integer(i));
12. list.add(i);
14. list.removeIf(i -> i == 10);
16. System.out.println(list);
17. }
18. }

* 

**[]**

* 

**[10, 10]**

* 

**Compilation Error**

**(Correct)**

* 

**[10, 10, 10]**

* 

**[10]**

**Explanation**

Variable "i" used in lambda expression clashes with another local variable "i" and hence causes compilation error.

Bottom of Form

Top of Form

Question 61: **Correct**

What will be the output 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 a = 5;
6. int x = 10;
7. switch(x) {
8. case 10:
9. a \*= 2;
10. case 20:
11. a \*= 3;
12. case 30:
13. a \*= 4;
14. }
15. System.out.println(a);
16. }
17. }

* 

**5**

* 

**30**

* 

**10**

* 

**120**

**(Correct)**

**Explanation**

Matching case block "case 10:" is found, a \*= 2; is executed, which means a = a \* 2; => a = 5 \* 2; => a = 10;

No break statement, hence it enters in fall-through.

a \*= 3; is executed, which means a = a \* 3; => a = 10 \* 3; => a = 30;

a \*= 4; is executed, which means a = a \* 4; => a = 30 \* 4; => a = 120;

Bottom of Form

Top of Form

Question 62: **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> dryFruits = new ArrayList<>();
10. dryFruits.add("Walnut");
11. dryFruits.add("Apricot");
12. dryFruits.add("Almond");
13. dryFruits.add("Date");
15. for(String dryFruit : dryFruits) {
16. if(dryFruit.startsWith("A")) {
17. dryFruits.remove(dryFruit);
18. }
19. }
21. System.out.println(dryFruits);
22. }
23. }

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

* 

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

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**[Walnut, Date]**

* 

**Compilation error**

**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.

PLEASE NOTE: for-each loop internally implements Iterator and invokes hasNext() and next() methods.

For the given code, 'dryFruits' list is being iterated using for-each loop (internally as an Iterator).

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 successfully 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 63: **Correct**

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

1. package com.udayan.oca;
3. public class Greetings {
4. String msg = null;
5. public Greetings() {
6. }
8. public Greetings(String str) {
9. msg = str;
10. }
12. public void display() {
13. System.out.println(msg);
14. }
16. public static void main(String [] args) {
17. Greetings g1 = new Greetings();
18. Greetings g2 = new Greetings("Good Evening!");
19. g1.display();
20. g1.display();
21. }
22. }

* 

**Good Evening!  
null**

* 

**null  
null**

**(Correct)**

* 

**null  
Good Evening!**

* 

**Compilation error**

**Explanation**

Greetings g1 = new Greetings(); invokes no-arg constructor. Property msg (of object referred by g1) is assigned to null.

Greetings g2 = new Greetings("Good Evening!"); invokes parameterized constructor, which assigns "Good Evening!" to msg of object referred by g2.

g1.display(); prints null

Again we have same call g1.display(); which prints null.

NOTE: We haven't called display() on object referred by g2.

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. public class Test {
4. public static void main(String[] args) {
5. String [] arr = {"A", "B", "C", "D"};
6. arr[0] = arr[1];
7. arr[1] = "E";
8. for(String s : arr) {
9. System.out.print(s + " ");
10. }
11. }
12. }

* 

**E E C D**

* 

**A E C D**

* 

**B E C D**

**(Correct)**

* 

**An exception is thrown at runtime**

* 

**Compilation error**

**Explanation**

arr[0] -> "A" and  arr[1] -> "B".

arr[0] = arr[1]; => arr[0] -> "B" and arr[1] -> "B".

arr[1] = "E"; => arr[0] -> "B" and arr[1] -> "E".

Hence output is: B E C D

Bottom of Form

Top of Form

Question 65: **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(2068, 4, 15);
9. System.out.println(date.getMonth() + ":" + date.getMonthValue());
10. }
11. }

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

* 

**APRIL:3**

* 

**April:3**

* 

**APRIL:4**

**(Correct)**

* 

**April:4**

**Explanation**

date.getMonth() returns the month of the year filed, using Month enum, all the enum constant names are in upper case.

date.getMonthValue() returns the value of the month.

NOTE: month value starts with 1 and it is different from java.util.Date API, where month value starts with 0.

Bottom of Form

Top of Form

Question 66: **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. StringBuilder sb = new StringBuilder("Hello");
10. List<StringBuilder> list = new ArrayList<>();
11. list.add(sb);
12. list.add(new StringBuilder("Hello"));
13. list.add(sb);
14. sb.append("World!");
16. System.out.println(list);
17. }
18. }

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

* 

**[HelloWorld!, HelloWorld!, HelloWorld!]**

* 

**[HelloWorld!, Hello, HelloWorld!]**

**(Correct)**

* 

**[HelloWorld!, Hello, Hello]**

* 

**[Hello, Hello, Hello]**

**Explanation**

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

sb.append("World!"); means sb --> {HelloWorld!}, which means 1st and 3rd items of ArrayList now refers to StringBuilder instance containing HelloWorld!

In the output, [HelloWorld!, Hello, HelloWorld!] is printed.

Bottom of Form

Top of Form

Question 67: **Correct**

Consider below code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. StringBuilder sb = new StringBuilder();
6. try {
7. for(;;) {
8. sb.append("OCA");
9. }
10. } catch(Exception e) {
11. System.out.println("Exception!!!");
12. }
13. System.out.println("Main ends!!!");
14. }
15. }

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

* 

**Program terminates abruptly**

**(Correct)**

* 

**"Main ends!!!" is printed on to the console and program terminates successfully**

* 

**"Exception!!!" and "Main ends!!!" are printed on to the console and program terminates successfully**

* 

**"Exception!!!" is printed on to the console and program terminates successfully**

* 

**"Exception!!!" is printed on to the console and program terminates abruptly**

**Explanation**

for(;;) is an infinite loop and hence `sb.append("OCA");` causes OutOfMemoryError which is a subclass of Error class.

main(String []) method throws OutOfMemoryError and program terminates abruptly.

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. String [] arr = {"abc", "TrUe", "false", null, "FALSE"};
6. for(String s : arr) {
7. System.out.print(Boolean.valueOf(s) ? "T" : "F");
8. }
9. }
10. }

* 

**TTFTT**

* 

**NullPointerException is thrown at runtime**

* 

**TTTFT**

* 

**FTFFF**

**(Correct)**

* 

**FFFFF**

**Explanation**

Boolean.valueOf(String s) returns true if passed String argument is not null and is equal, ignoring case, to the String "true". In all other cases it returns false.

Boolean.valueOf("abc") => false. As "abc".equalsIgnoreCase("true") is false.

Boolean.valueOf("TrUe") => true. As "TrUe".equalsIgnoreCase("true") is true.

Boolean.valueOf("false") => false. As "false".equalsIgnoreCase("true") is false.

Boolean.valueOf(null) => false. As passed argument is null.

Boolean.valueOf("FALSE") => false. As "FALSE".equalsIgnoreCase("true") is false.

Bottom of Form

Top of Form

Question 69: **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. //C.java
18. package com.udayan.oca;
20. public class C extends A {
21. public void print() {
22. System.out.println("C");
23. }
24. }
25. //Test.java
26. package com.udayan.oca.test;
28. import com.udayan.oca.\*;
30. public class Test {
31. public static void main(String[] args) {
32. A obj1 = new C();
33. A obj2 = new B();
34. C obj3 = (C)obj1;
35. C obj4 = (C)obj2;
36. obj3.print();
37. }
38. }

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

* 

**A**

* 

**ClassCastException is thrown at runtime**

**(Correct)**

* 

**C**

* 

**B**

* 

**Compilation error**

**Explanation**

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

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

Let's check the code inside main method.

A obj1 = new C(); => obj1 refers to an instance of C class, it is polymorphism.

A obj2 = new B(); => obj2 refers to an instance of B class, it is polymorphism.

C obj3 = (C)obj1; => obj1 actually refers to an instance of C class, so at runtime obj3 (C type) will refer to an instance of C class. As obj1 is of A type so explicit typecasting is necessary.

C obj4 = (C)obj2; => obj2 actually refers to an instance of B class, so at runtime obj4 (C type) will refer to an instance of B class. B and C are siblings and can't refer to each other, so this statement will throw ClassCastException at runtime.

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. int a = 3;
6. m(++a, a++);
7. System.out.println(a);
8. }
10. private static void m(int i, int j) {
11. i++;
12. j--;
13. }
14. }

* 

**5**

**(Correct)**

* 

**3**

* 

**4**

* 

**6**

**Explanation**

Method m works on copies and changes done to i and j are local to method m only.

m(++a, a++); [a=3].

m(4, a++); [a=4].

m(4, 4); [a=5].

System.out.println(a); => Prints 5 on to the console.

Bottom of Form