Top of Form

Question 1: **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 msg = "Hello";
6. boolean [] flag = new boolean[1];
7. if(flag[0]) {
8. msg = "Welcome";
9. }
10. System.out.println(msg);
11. }
12. }

* 

**Welcome**

* 

**ArrayIndexOutOfBoundsException**

* 

**NullPointerException**

* 

**Hello**

**(Correct)**

**Explanation**

Variable msg is referring to String object "Hello".

There is only one element in boolean array object and it is initialized to default value of boolean, which is false.

flag[0] is false, if-check fails and control doesn't enter if block.

System.out.println(msg) prints original value of msg, which is "Hello".

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.of(2020, 9, 6);
9. System.out.println(date);
10. }
11. }

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

* 

**2020-9-6**

* 

**2020-09-06**

**(Correct)**

* 

**2020-6-9**

* 

**2020-06-09**

**Explanation**

In LocalDate.of(int, int, int) method, 1st parameter is year, 2nd is month and 3rd is day of the month.

toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: "uuuu-MM-dd".

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. public class Test {
4. private static void m1() {
5. System.out.println(1/0);
6. }
8. public static void main(String[] args) {
9. try {
10. m1();
11. } finally {
12. System.out.println("A");
13. }
14. }
15. }

* 

**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.**

**(Correct)**

* 

**Compilation error.**

**Explanation**

Method m1() throws an instance of ArithmeticException and method m1() doesn't handle it, so it forwards the exception to calling method main.

Method main doesn't handle ArithmeticException so it forwards it to JVM, but just before that finally block is executed. This prints A on to the console.

After that JVM prints the stack trace and terminates the program abruptly.

Bottom of Form

Top of Form

Question 4: **Correct**

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

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String str = "Java Rocks!";
6. System.out.println(str.length() + " : " + str.charAt(10));
7. }
8. }

* 

**11 : s**

* 

**Compilation error.**

* 

**11 : !**

**(Correct)**

* 

**An exception is thrown at runtime.**

**Explanation**

String class has length() method, which returns number of characters in the String. So length() method returns 11.

String class has charAt(int index) method, which returns character at passed index. str.charAt(10) looks for character at index 10. index starts with 0. ! sign is at index 10.

Hence output is: 11 : !

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. class Base {
4. public void m1() throws NullPointerException {
5. System.out.println("Base: m1()");
6. }
7. }
9. class Derived extends Base {
10. public void m1() throws RuntimeException {
11. System.out.println("Derived: m1()");
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. Base obj = new Derived();
18. obj.m1();
19. }
20. }

* 

**Compilation error in Derived class**

* 

**Compilation error in Test class**

* 

**Derived: m1()**

**(Correct)**

* 

**Base: m1()**

**Explanation**

NullPointerException extends RuntimeException, but there are no overriding rules related to unchecked exceptions.

So, method m1() in Derived class correctly overrides Base class method.

Rest is simple polymorphism. obj refers to an instance of Derived class and hence obj.m1(); invokes method m1() of Derived class, which prints "Derived: m1()" to the console.

Bottom of Form

Top of Form

Question 6: **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("2000-06-25");
9. while(date.getDayOfMonth() >= 20) {
10. System.out.println(date);
11. date.plusDays(-1);
12. }
13. }
14. }

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

* 

**An exception is thrown at runtime**

* 

**System.out.println(date); is executed 6 times**

* 

**Compilation error**

* 

**System.out.println(date); is executed more than 6 times**

**(Correct)**

**Explanation**

date --> {2000-06-25}. date.getDayOfMonth() = 25, 25 >= 20 is true, hence control goes inside while loop and executes System.out.println(date); statement.

date.plusDays(-1); creates a new LocalDate object {2000-06-24} but date reference variable still refers to {2000-06-25}. date.getDayOfMonth() again returns 25, this is an infinite loop.

Bottom of Form

Top of Form

Question 7: **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> trafficLight = new ArrayList<>();
10. trafficLight.add(1, "RED");
11. trafficLight.add(2, "ORANGE");
12. trafficLight.add(3, "GREEN");
14. trafficLight.remove(new Integer(2));
16. System.out.println(trafficLight);
17. }
18. }

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

* 

**[RED, ORANGE]**

* 

**[RED, ORANGE, GREEN]**

* 

**[RED, GREEN]**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**(Correct)**

**Explanation**

There is no element at index 0 so call to add element at index 1, "trafficLight.add(1, "RED");" throws an instance of java.lang.IndexOutOfBoundsException.

trafficLight.remove(new Integer(2)); matches with trafficLight.remove(Object) and hence no compilation error.

Bottom of Form

Top of Form

Question 8: **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 joiningDate = LocalDate.parse("2006-03-16");
9. System.out.println(joiningDate.withDayOfYear(29));
10. }
11. }

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

* 

**None of the other options**

* 

**2006-03-29**

* 

**2006-01-29**

**(Correct)**

* 

**2006-01-01**

**Explanation**

joiningDate --> {2006-03-16}.

joiningDate.withDayOfYear(29) returns a new LocalDate object with the day of the Year altered.

A year has 365 days, so 29 means 29th day of the year, which is 29th Jan 2006.

NOTE: There are other with methods, you should know for the exam. withDayOfMonth(int), withMonth(int) and withYear(int).

Bottom of Form

Top of Form

Question 9: **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 b = new Boolean("tRUe");
6. switch(b) {
7. case true:
8. System.out.println("ONE");
9. case false:
10. System.out.println("TWO");
11. default:
12. System.out.println("THREE");
13. }
14. }
15. }

* 

**TWO  
THREE**

* 

**ONE  
TWO  
THREE**

* 

**None of the other options**

**(Correct)**

* 

**THREE**

**Explanation**

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

switch(b) causes compilation failure as b is of Boolean type.

Bottom of Form

Top of Form

Question 10: **Correct**

1. package com.udayan.oca;
3. class TestException extends Exception {
4. public TestException() {
5. super();
6. }
8. public TestException(String s) {
9. super(s);
10. }
11. }
13. public class Test {
14. public void m1() throws \_\_\_\_\_\_\_\_\_\_ {
15. throw new TestException();
16. }
17. }

For the above code, fill in the blank with one option.

* 

**Error**

* 

**Exception**

**(Correct)**

* 

**RuntimeException**

* 

**Object**

**Explanation**

Method m1() throws an instance of TestException, which is a checked exception as it extends Exception class.

So in throws clause we must provide:

1. Checked exception.

2. Exception of TestException type or it's super types (Exception, Throwable), Object cannot be used in throws clause.

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 extends String {
4. @Override
5. public String toString() {
6. return "TEST";
7. }
9. public static void main(String[] args) {
10. Test obj = new Test();
11. System.out.println(obj);
12. }
13. }

* 

**Output string contains @ symbol**

* 

**Exception is thrown at runtime**

* 

**TEST**

* 

**Compilation error**

**(Correct)**

**Explanation**

String is a final class so it cannot be extended.

Bottom of Form

Top of Form

Question 12: **Correct**

Suppose you have created a java file, "MyClass.java". Which of the following commands will compile the java file?

* 

**java MyClass.java**

* 

**javac MyClass**

* 

**java MyClass**

* 

**javac MyClass.java**

**(Correct)**

* 

**javac MyClass.class**

**Explanation**

Command to compile a java file: javac <java\_file\_name>.java [.java extension is compulsory]

Command to execute a java class: java <class\_file\_name> [.class extension should not be used]

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 i = 5;
6. if(i++ < 6) {
7. System.out.println(i++);
8. }
9. }
10. }

* 

**Program executes successfully but nothing is printed on to the console**

* 

**7**

* 

**5**

* 

**6**

**(Correct)**

**Explanation**

Initially i = 5. if(i++ < 6) means if(5 < 6) and then i = 6.

5 < 6 is true, control goes inside if-block and executes System.out.println(i++); This prints current value of i to the console, which is 6 and after that increments the value of i by 1, so i becomes 7.

Bottom of Form

Top of Form

Question 14: **Incorrect**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.time.LocalDate;
5. import java.time.Month;
6. import java.time.Period;
8. public class Test {
9. public static void main(String [] args) {
10. LocalDate date = LocalDate.of(2000, Month.JANUARY, 1);
11. Period period = Period.parse("p-30000y");
12. System.out.println(date.plus(period));
13. }
14. }

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

* 

**Compilation error**

* 

**Runtime exception**

* 

**28000-01-01**

**(Incorrect)**

* 

**32000-01-01**

* 

**-28000-01-01**

**(Correct)**

**Explanation**

There are 2 of methods available in LocalDate class: of(int, int, int) and of(int, Month, int). Month can either be passed as int value (1 to 12) or enum constants Month.JANUARY to Month.DECEMBER.

Period.parse(CharSequence) method accepts the String parameter in "PnYnMnD" format, over here P,Y,M and D can be in any case. "p-30000y" means Period of -30000 years.

The minimum supported LocalDate is: {-999999999-01-01} and maximum supported LocalDate is: {+999999999-12-31}. If period of -30000 years is added to 1st Jan 2000, then result is 1st Jan -28000.

Bottom of Form

Top of Form

Question 15: **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<StringBuilder> dryFruits = new ArrayList<>();
10. dryFruits.add(new StringBuilder("Walnut"));
11. dryFruits.add(new StringBuilder("Apricot"));
12. dryFruits.add(new StringBuilder("Almond"));
13. dryFruits.add(new StringBuilder("Date"));
15. for(int i = 0; i < dryFruits.size(); i++)
16. {
17. if(i == 0) {
18. dryFruits.remove(new StringBuilder("Almond"));
19. }
20. }
22. System.out.println(dryFruits);
23. }
24. }

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

* 

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

**(Correct)**

* 

**[Walnut, Date]**

* 

**[Walnut, Apricot, Date]**

* 

**An exception is thrown at runtime**

**Explanation**

In this example, code is trying to remove an item from the list while iterating using traditional for loop so one can think that this code would throw java.util.ConcurrentModificationException.

But note, java.util.ConcurrentModificationException will never be thrown for traditional for loop. It is thrown for for-each loop or while using Iterator/ListIterator.

In this case dryFruits.remove(new StringBuilder("Almond")); will never remove any items from the list as StringBuilder class doesn't override the equals(Object) method of Object class.

StringBuilder instances created at "dryFruits.add(new StringBuilder("Almond"));" and "dryFruits.remove(new StringBuilder("Almond"));" are at different memory locations and equals(Object) method returns false for these instances.

Bottom of Form

Top of Form

Question 16: **Correct**

When does a class get the default constructor?

* 

**If the class does not define any constructors explicitly.**

**(Correct)**

* 

**If you define parameterized constructor for the class.**

* 

**All classes in Java get a default constructor.**

* 

**You have to define at least one constructor to get the default constructor.**

**Explanation**

Default constructor (which is no-argument constructor) is added by Java compiler, only if there are no constructors in the class.

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. 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(110);
10. list.add(new Integer(110));
11. list.add(110);
13. list.removeIf(i -> i == 110);
14. System.out.println(list);
15. }
16. }

* 

**[110, 110, 110]**

* 

**[]**

**(Correct)**

* 

**[110, 110]**

* 

**[110]**

**Explanation**

As list can store only wrapper objects and not primitives, hence

for list.add(110); auto-boxing creates an Integer object {110}.

for list.add(new Integer(110)); as new keyword is used so another Integer object {110} is created.

for 3rd add method call, list.add(110); auto-boxing kicks in and as 110 is between -128 to 127, hence Integer object created at 1st statement is referred.

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

Boolean expression is : i == 110; in this expression i is wrapper object and 110 is int literal so java extracts int value of wrapper object, i and then equates. As all the 3 objects store 110, hence true is returned. All integer objects are removed form the list.

If list.removeIf(i -> i == new Integer(110)); was used, then all three list elements would return false as object references are equated and not contents.

Bottom of Form

Top of Form

Question 18: **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 obj = new LocalDate(2020, 2, 14);
10. System.out.println(obj.minus(Period.ofDays(10)));
11. }
12. }

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

* 

**2020-02-04**

* 

**Compilation error**

**(Correct)**

* 

**2020-02-03**

* 

**Runtime exception**

**Explanation**

Constructor of LocalDate is declared private so cannot be called from outside, hence new LocalDate(2020, 2, 14); causes compilation failure.

Overloaded static methods "of" and "parse" are provided to create the instance of LocalDate.

LocalTime, LocalDateTime, Period also specify private constructors and provide "of" and "parse" methods to create respective instances.

Bottom of Form

Top of Form

Question 19: **Correct**

Consider below code snippet:

1. public static void process(/\*INSERT\*/ list) {
2. list.add(100); //Line 2
3. int x = list.get(0); //Line 3
4. System.out.println(list.size() + ":" + x);
5. }

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

* 

**List<Integer>**

**(Correct)**

* 

**List**

* 

**List<int>**

* 

**List<Object>**

**Explanation**

Generic type can only be reference type and not primitive type, hence List<int> is not a valid syntax.

If you use raw type List or List<Object> then Line 3 will give compilation error as list.get(0) will return Object type. Object type cannot be converted to primitive type int, so List and List<Object> will cause compilation failure of Line 3.

List<Integer> is the only correct option left.

Bottom of Form

Top of Form

Question 20: **Correct**

Which of the following is true for code below?

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

* 

**ArrayIndexOutOfBoundsException**

**(Correct)**

* 

**Compilation error**

* 

**NullPointerException**

* 

**0**

**Explanation**

arr refers to an array object of size 0. That means arr stores some memory address.

So we will not get NullPointerException in this case. But index 0 is not available for an array object of size 0 and thus ArrayIndexOutOfBoundsException is thrown at runtime.

Bottom of Form

Top of Form

Question 21: **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. Error obj = new Error();
6. boolean flag1 = obj instanceof RuntimeException; //Line n1
7. boolean flag2 = obj instanceof Exception; //Line n2
8. boolean flag3 = obj instanceof Error; //Line n3
9. boolean flag4 = obj instanceof Throwable; //Line n4
10. System.out.println(flag1 + ":" + flag2 + ":" + flag3 + ":" + flag4);
11. }
12. }

* 

**Compilation error**

**(Correct)**

* 

**true:true:true:true**

* 

**false:false:true:true**

* 

**false:true:true:true**

* 

**false:false:true:false**

**Explanation**

class Error extends Throwable, so `obj instanceof Error;` and `obj instanceof Throwable;` return true.

But Error class is not related to Exception and RuntimeException classes in multilevel inheritance and that is why Line n1 and Line n2 causes compilation error.

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. import java.util.function.Predicate;
5. public class Test {
6. public static void main(String[] args) {
7. printNumbers(i -> i % 2 != 0);
8. }
10. private static void printNumbers(Predicate<Integer> predicate) {
11. for(int i = 1; i <= 10; i++) {
12. if(predicate.test(i)) {
13. System.out.print(i);
14. }
15. }
16. }
17. }

* 

**1234567891011**

* 

**246810**

* 

**12345678910**

* 

**1357911**

* 

**13579**

**(Correct)**

**Explanation**

In the boolean expression (predicate.test(i)): i is of primitive int type but auto-boxing feature converts it to Integer wrapper type.

test(Integer) method of Predicate returns true if passed number is an odd number, so given loop prints only odd numbers. for loops works for the numbers from 1 to 10.

Bottom of Form

Top of Form

Question 23: **Correct**

super keyword in java is used to:

* 

**refer to current class object.**

* 

**refer to static variable of the class.**

* 

**refer to static method of the class.**

* 

**refer to parent class object.**

**(Correct)**

**Explanation**

super refers to parent class object and this refers to currently executing object.

Bottom of Form

Top of Form

Question 24: **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<Boolean> list = new ArrayList<>();
10. list.add(true);
11. list.add(new Boolean("tRue"));
12. list.add(new Boolean("abc"));
14. if(list.remove(1)) {
15. list.remove(1);
16. }
18. System.out.println(list);
19. }
20. }

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

* 

**[true]**

**(Correct)**

* 

**[false]**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

* 

**[true, false]**

**Explanation**

list.add(true); => Auto-boxing converts boolean literal true to Boolean instance containing true. Element at index 0 represents true.

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.

list.add(new Boolean("tRue")); => Element at index 1 represents true.

list.add(new Boolean("abc")); => Element at index 2 represents false.

So initially list contains [true, true, false].

As generic list is used, so list.remove(1) removes the Boolean instance (true) stored at index 1 and returns it. So after this operation list contains [true, false].

For the boolean expression of if-block, Java runtime extracts the stored boolean value using booleanValue() method, which returns true. Control goes inside if-block and executes list.remove(1);  This removes element at index 1 so after this operation list contains [true] and [true] is printed on to the console.

Bottom of Form

Top of Form

Question 25: **Correct**

Given the code:

1. package com.udayan.oca;
3. public class Pen {
4. public static void main(String[] args) {
5. Pen p1 = new Pen(); //Line 1
6. Pen p2 = new Pen(); //Line 2
7. p1 = p2; //Line 3
8. p1 = null; //Line 4
9. }
10. }

When is the Pen object, created at Line 1 eligible for Garbage Collection?

* 

**After Line 3**

**(Correct)**

* 

**After Line 2**

* 

**After Line 4**

* 

**At the end of main method**

**Explanation**

At Line 3, p1 starts referring to the object referred by p2(Created at Line 2).

So, after Line 3, object created at Line 1 becomes unreachable and thus eligible for Garbage Collection.

Bottom of Form

Top of Form

Question 26: **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 x = 2;
6. switch (x) {
7. default:
8. System.out.println("Still no idea what x is");
9. case 1:
10. System.out.println("x is equal to 1");
11. break;
12. case 2:
13. System.out.println("x is equal to 2");
14. break;
15. case 3:
16. System.out.println("x is equal to 3");
17. break;
18. }
19. }
20. }

* 

**x is equal to 2**

**(Correct)**

* 

**Still no idea what x is  
x is equal to 1**

* 

**Compilation error**

* 

**Produces no output**

**Explanation**

Even though default block is available at the top but matching case is present.

So control goes inside matching case and prints "x is equal to 2" on to the console.

After that break; statement takes the control out of the switch- case block.

Bottom of Form

Top of Form

Question 27: **Correct**

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

1. package com.udayan.oca;
3. public class Test {
4. static Boolean[] arr = new Boolean[1];
5. public static void main(String[] args) {
6. if(arr[0]) {
7. System.out.println(true);
8. } else {
9. System.out.println(false);
10. }
11. }
12. }

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

* 

**NullPointerException is thrown at runtime**

**(Correct)**

* 

**false**

* 

**true**

* 

**Compilation error**

**Explanation**

All the array elements are initialized to their default values. arr is of Boolean type (reference type), so arr[0] is initialized to null.

if expression works with Boolean type variable, so "if(arr[0])" doesn't give compilation error but java runtime extracts the boolean value stored in arr[0] and it uses booleanValue() method.

arr[0].booleanValue() means booleanValue() method is invoked on null reference and hence NullPointerException is thrown at runtime.

Bottom of Form

Top of Form

Question 28: **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;
6. public Greetings() {
7. this("Good Morning!");
8. }
10. public Greetings(String str) {
11. msg = str;
12. }
14. public void display() {
15. System.out.println(msg);
16. }
18. public static void main(String [] args) {
19. Greetings g1 = new Greetings();
20. Greetings g2 = new Greetings("Good Evening!");
21. g1.display();
22. g2.display();
23. }
24. }

* 

**Good Morning!  
Good Evening!**

**(Correct)**

* 

**null  
null**

* 

**null  
Good Evening!**

* 

**Good Morning!  
null**

**Explanation**

Greetings g1 = new Greetings(); invokes no-arg constructor.

No-arg constructor calls parameterized constructor with the argument "Good Morning!"

Parameterized constructor assigns "Good Morning!" to msg variable of the object referred by g1.

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

g1.display(); prints Good Morning!

g2.display(); prints Good Evening!

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. String [] arr = new String[1];
6. System.out.println(arr[0].isEmpty());
7. }
8. }

* 

**true**

* 

**false**

* 

**ArrayIndexOutOfBoundsException is thrown at runtime**

* 

**NullPointerException is thrown at runtime**

**(Correct)**

**Explanation**

All the elements of array are initialized to respective zeros (in case of primitive type) or null (in case of reference type).

So, arr[0] refers to null.

Method 'isEmpty()' is invoked on null reference and hence NullPointerException is thrown at runtime.

Bottom of Form

Top of Form

Question 30: **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. for:
6. for (int i = 2; i <= 100; i = i + 2) {
7. for(int j = 1; j <= 10; j++) {
8. System.out.print(i \* j + "\t");
9. }
10. System.out.println();
11. if(i == 10) {
12. break for;
13. }
14. }
15. }
16. }

* 

**Compilation error**

**(Correct)**

* 

**Total 100 rows will be there in the output**

* 

**Total 5 rows will be there in the output**

* 

**Total 50 rows will be there in the output**

**Explanation**

for is a keyword and hence can't be used as a label.

Java labels follow the identifier naming rules and one rule is that we can't use java keywords as identifier. Hence, Compilation error.

Bottom of Form

Top of Form

Question 31: **Incorrect**

Given code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. int [] arr = {1, 2, 3, 4, 5};
6. int x = 0;
7. for(/\*INSERT\*/) {
8. x += arr[n];
9. }
10. System.out.println(x);
11. }
12. }

Which 3 options, if used to replace /\*INSERT\*/, on execution will print 9 on to the console?

* 

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

**(Incorrect)**

* 

**int n = 1; n < arr.length - 1; n++**

**(Correct)**

* 

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

**(Correct)**

* 

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

* 

**int n = 3; n < arr.length; n++**

**(Correct)**

**Explanation**

Logic in for loop is adding array elements. You need to find out which array elements when added will result in 9. Possible options are: {1+3+5, 2+3+4, 4+5}.

Based on these 3 combinations you can select 3 correct options.

Bottom of Form

Top of Form

Question 32: **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(Student 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, 25]  
Student[James, 27]  
Student[James, 25]  
Student[James, 25]**

**(Correct)**

* 

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

* 

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

* 

**Student[James, 27]**

**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 NOT been overridden by the Student class. In fact, equals(Student) is overloaded. But overloaded version is not invoked while equating the Student objects.

equals(Object) method defined in Object class is invoked and equals(Object) method defined in Object class uses == operator to check the equality and in this case as all the Student objects are stored at different memory location, hence not equal.

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

Bottom of Form

Top of Form

Question 33: **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 = new String(new char[] {'M', 'a', 'n', 'g', 'o'});
6. switch (fruit) {
7. default:
8. System.out.println("ANY FRUIT WILL DO");
9. case "Apple":
10. System.out.println("APPLE");
11. case "Mango":
12. System.out.println("MANGO");
13. case "Banana":
14. System.out.println("BANANA");
15. break;
16. }
17. }
18. }

* 

**ANY FRUIT WILL DO**

* 

**MANGO  
BANANA**

**(Correct)**

* 

**MANGO**

* 

**ANY FRUIT WILL DO  
APPLE  
MANGO  
BANANA**

**Explanation**

fruit refers to String object "Mango". Matching case is available, MANGO is printed on to the console.

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

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

Bottom of Form

Top of Form

Question 34: **Correct**

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

1. //TestBaseDerived.java
2. package com.udayan.oca;
4. class Base {
5. protected void m1() {
6. System.out.println("Base: m1()");
7. }
8. }
10. class Derived extends Base {
11. void m1() {
12. System.out.println("Derived: m1()");
13. }
14. }
16. public class TestBaseDerived {
17. public static void main(String[] args) {
18. Base b = new Derived();
19. b.m1();
20. }
21. }

* 

**Base: m1()**

* 

**Derived: m1()**

* 

**Base: m1()  
Derived: m1()**

* 

**None of the other options**

**(Correct)**

**Explanation**

Derived class overrides method m1() of Base class. Access modifier of method m1() in Base class is protected, so overriding method can use protected or public.

But overriding method in this case used default modifier and hence there is compilation error.

Bottom of Form

Top of Form

Question 35: **Correct**

Which one of these top level classes cannot be sub-classed?

* 

**class Dog {}**

* 

**abstract class Cat {}**

* 

**private class Car {}**

* 

**final class Electronics {}**

**(Correct)**

**Explanation**

class Dog {}: can be sub-classed within the same package.

abstract class Cat {}: can be sub-classed within the same package.

final class Electronics {}: a class with final modifier cannot be sub-classed.

private class Car {}: a top level class cannot be declared with private modifier.

Bottom of Form

Top of Form

Question 36: **Correct**

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

1. //Test.java
2. package com.udayan.oca;
4. import java.io.FileNotFoundException;
6. public class Test {
7. public static void main(String[] args) {
8. try {
9. System.out.println(1);
10. } catch (NullPointerException ex) {
11. System.out.println("ONE");
12. } catch (FileNotFoundException ex) {
13. System.out.println("TWO");
14. }
15. System.out.println("THREE");
16. }
17. }

* 

**Compilation error**

**(Correct)**

* 

**TWO  
THREE**

* 

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

* 

**ONE  
THREE**

* 

**THREE**

**Explanation**

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

catch(FileNotFoundException ex) {} causes  compilation error in this case as System.out.println(1); will never throw FileNotFoundException.

NOTE: Java allows to catch Exception type. catch(Exception ex) {} will never cause compilation error.

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. 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(7);
11. list.add(50);
12. list.add(17);
13. list.add(10);
14. list.add(5);
16. list.removeIf(a -> a % 10 == 0);
18. System.out.println(list);
19. }
20. }

* 

**[100, 7, 50, 17, 10, 5]**

* 

**[100, 50, 10]**

* 

**[7, 17, 5]**

**(Correct)**

* 

**Runtime Exception**

* 

**Compilation error**

**Explanation**

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

Predicate's test method returns true for all the Integers divisible by 10.

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. 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("papaya");
19. System.out.println(fruits);
20. }
21. }

* 

**[apple, orange, mango, banana]**

* 

**An exception is thrown at runtime**

* 

**[apple, 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("papaya");

fruits list doesn't have "papaya", so the list remain unchanged. In the console, you get: [apple, orange, mango, banana, grape].

Bottom of Form

Top of Form

Question 39: **Correct**

Which of the following statement declares a constant field in Java?

* 

**const int x = 10;**

* 

**int x = 10;**

* 

**final static int x = 10;**

**(Correct)**

* 

**static int x = 10;**

**Explanation**

Fields declared with final are constant fields.

Bottom of Form

Top of Form

Question 40: **Correct**

Consider below Lambda expression:

Predicate predicate = s -> true;

Which of the lambda expression can successfully replace the lambda expression in above statement?

* 

**s -> {return true}**

* 

**s -> {return true;}**

**(Correct)**

* 

**s -> {true;}**

* 

**s -> {true}**

**Explanation**

In the lambda expression's body, if used, all 3 [return, {}, ;] must be used together.

Bottom of Form

Top of Form

Question 41: **Correct**

Consider below code of Test.java file:

1. package com.udayan.oca;
3. import java.util.ArrayList;
4. import java.util.List;
6. interface Sellable {}
7. abstract class Animal {}
8. class Mammal extends Animal{}
9. class Rabbit extends Mammal implements Sellable{}
11. public class Test {
12. {
13. List<Animal> list = new ArrayList<>();
14. list.add(new Rabbit());
15. }
16. {
17. List<Animal> list = new ArrayList<>();
18. list.add(new Mammal());
19. }
20. {
21. List<Mammal> list = new ArrayList<>();
22. list.add(new Rabbit());
23. }
24. {
25. List<Sellable> list = new ArrayList<>();
26. list.add(new Mammal());
27. }
28. {
29. List<Sellable> list = new ArrayList<>();
30. list.add(new Rabbit());
31. }
32. }

Which of the following statement is true?

* 

**Three initializer blocks cause compilation error.**

* 

**Two initializer blocks cause compilation error.**

* 

**Four initializer blocks cause compilation error.**

* 

**Five initializer blocks cause compilation error.**

* 

**Only one initializer block causes compilation error.**

**(Correct)**

**Explanation**

Even though code seems to be checking the knowledge of ArrayList but it actually checks the knowledge of Polymorphism.

List<Sellable> list = new ArrayList<>(); is valid statement and list can accept any object passing instanceof check for Sellable type.

Rabbit implements Sellable hence new Rabbit() can be added to list.

But as Mammal doesn't implement Sellable hence new Mammal() can't be added to list.

Other initializer blocks can be verified on similar lines. So there is only one initializer block, which causes compilation error.

Bottom of Form

Top of Form

Question 42: **Correct**

Given code:

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. StringBuilder sb = new StringBuilder(5);
6. sb.append("0123456789");
7. sb.delete(8, 1000);
8. System.out.println(sb);
9. }
10. }

What is the result?

* 

**Compilation error**

* 

**89**

* 

**01234567**

**(Correct)**

* 

**An exception is thrown at runtime**

**Explanation**

`new StringBuilder(5);` creates a StringBuilder instance, whose internal char array's length is 5 but the internal char array's length is adjusted when characters are added/removed from the StringBuilder instance. `sb.append("0123456789");` successfully appends "0123456789" to the StringBuilder's instance referred by sb.

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(8, 1000);` is equivalent to `sb.delete(8, sb.length());` and this deletes characters at 8th index (8) and 9th index (9). So remaining characters are: "01234567".

StringBuilder class overrides toString() method, which prints the text stored in StringBuilder instance. Hence, `System.out.println(sb);` prints 01234567 on to the console.

Bottom of Form

Top of Form

Question 43: **Correct**

For the class Apple, which option, if used to replace /\*INSERT\*/, will print GREEN on to the console?

1. package com.udayan.oca;
3. public class Apple {
4. public String color;
6. public Apple(String color) {
7. /\*INSERT\*/
8. }
10. public static void main(String [] args) {
11. Apple apple = new Apple("GREEN");
12. System.out.println(apple.color);
13. }
14. }

* 

**this.color = GREEN;**

* 

**color = GREEN;**

* 

**this.color = color;**

**(Correct)**

* 

**color = color;**

**Explanation**

Instance variable color is shadowed by the parameter variable color of parameterized constructor. So, color = color will have no effect, because short hand notation within constructor body will always refer to LOCAL variable. To refer to instance variable, this reference is needed. Hence 'this.color = color;' is correct.

'color = GREEN;' and 'this.color = GREEN;' cause compilation error as GREEN is not within double quotes("").

NOTE: 'color = "GREEN";' will only assign 'GREEN' to local variable and not instance variable but 'this.color = "GREEN";' will assign 'GREEN' to instance variable.

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. int start = 1;
6. int sum = 0;
7. do {
8. if(start % 2 == 0) {
9. continue;
10. }
11. sum += start;
12. } while(++start <= 10);
13. System.out.println(sum);
14. }
15. }

* 

**Compilation error**

* 

**55**

* 

**25**

**(Correct)**

* 

**24**

**Explanation**

When start is divisible by 2 [2, 4, 6, 8, 10], continue; statement takes the control to boolean expression and hence sum += start; is not executed.

Hence result is the sum of numbers 1,3,5,7,9.

Bottom of Form

Top of Form

Question 45: **Correct**

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

1. package com.udayan.oca;
3. class Super {
4. public Super(int i) {
5. System.out.println(100);
6. }
7. }
9. class Sub extends Super {
10. public Sub() {
11. System.out.println(200);
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. new Sub();
18. }
19. }

* 

**200**

* 

**100  
200**

* 

**200  
100**

* 

**Compilation Error**

**(Correct)**

**Explanation**

super(); is added by the compiler as the first statement in both the constructors.

Class Super extends from Object class and Object class has no-argument constructor, hence no issues with the constructor of Super class.

But no-arg constructor is not available in Super class, hence calling super(); from Sub class constructor gives compilation error.

Bottom of Form

Top of Form

Question 46: **Correct**

For the given code:

1. package com.udayan.oca;
3. interface I01 {
4. void m1();
5. }
7. public class Implementer extends Object implements I01{
8. protected void m1() {
10. }
11. }

* 

**Method m1() in Implementer class is not implemented correctly.**

**(Correct)**

* 

**Implementer class declaration is not correct.**

* 

**interface I01 gives compilation error as method m1 is not public.**

* 

**None of the other options.**

**Explanation**

void m1(); in interface I01 is equivalent to public abstract void m1(); So method m1() is implicitly public and abstract.

In java,  a class can extend from only one class but can implement multiple interfaces. Correct keywords are: extends and implements.

so class declaration is correct.

As method m1() is implicitly public in I01, hence overriding method in Implementer class should also be public. But it is protected and hence compiler complains.

Bottom of Form

Top of Form

Question 47: **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. System.out.println(process(arr, 3, 8)); //Line 5
7. }
9. /\*INSERT\*/
10. }

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?

Select 3 options.

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

**(Correct)**

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

**(Correct)**

**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).

As process(arr, 3, 8) is passed as an argument of System.out.println method, hence process method's return type can be anything apart from void. println method is overloaded to accept all primitive types, char [], String type and Object type. int[] are String [] are of Object type.

In the given options, method specifying int as return type cannot return null as null can't be assigned to primitive type. int process(...) would cause compilation error.

Bottom of Form

Top of Form

Question 48: **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 Exam.java file:

1. D:.
2. └───WORK
3. └───QUIZ
4. └───SEC07
5. ├───classes
6. └───src
7. └───com
8. └───udayan
9. └───test
10. Exam.java

You are currently at Sec07 folder.  
D:\WORK\Quiz\Sec07>

Which of the following javac command, typed from above location, will generate Exam.class file structure under classes directory?

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

* 

**javac classes\ src\com\udayan\test\Exam.java**

* 

**javac -d classes\ src\com\udayan\test\Exam.java**

**(Correct)**

* 

**Not possible by javac command**

* 

**javac -d classes\ Exam.java**

**Explanation**

Use -d option with javac command. As you are typing javac command from within Sec07 directory, hence path of java file relative to Sec07 directory needs to be given.

So, correct command is: javac -d classes\ src\com\udayan\test\Exam.java

Bottom of Form

Top of Form

Question 49: **Incorrect**

Consider 2 files:

1. //Counter.java
2. package com.udayan.oca;
4. public class Counter {
5. public int count = 0;
7. public Counter(int start) {
8. count = start;
9. }
11. public int getCount() {
12. return count;
13. }
15. public void increase(int val) {
16. count = count + val;
17. }
19. public String toString() {
20. return this.count + "";
21. }
22. }
23. //Test.java
24. package com.udayan.oca.test;
26. import java.util.Arrays;
28. import com.udayan.oca.Counter;
30. public class Test {
31. public static void main(String[] args) {
32. Counter[] arr = new Counter[] { new Counter(-1000), new Counter(539), new Counter(0) };
34. /\* INSERT \*/
36. System.out.println(Arrays.toString(arr));
37. }
38. }

Currently on executing Test class, output is: [-1000, 539, 0].

And below blocks:

1.

for(Counter ctr : arr) {

    ctr.count = 100;

}

2.

for (Counter ctr : arr) {

    int x = ctr.getCount();

    x = 100;

}

3.

for (Counter ctr : arr) {

    ctr.getCount() = 100;

}

4.

for(Counter ctr : arr) {

    ctr.increase(100 - ctr.count);

}

5.

for (Counter ctr : arr) {

    ctr.increase(100 - ctr.getCount());

}

6.

for(Counter ctr : arr) {

    ctr.increase(-ctr.getCount() + 100);

}

7.

for(Counter ctr : arr) {

    ctr.increase(-ctr.count + 100);

}

How many blocks can replace /\*INSERT\*/ such that output is: [100, 100, 100]?

* 

**Only Two blocks**

* 

**Only Five blocks**

**(Correct)**

* 

**Only Four blocks**

* 

**Only Six blocks**

**(Incorrect)**

* 

**Only One block**

* 

**All Seven blocks**

* 

**Only Three blocks**

**Explanation**

There are 2 ways to change the value of count variable of Counter class:

1. As access modifier of count variable is public, hence it can easily be accessed from other classes using the instance of Counter class, such as `new Counter().count` or `obj.count` (where obj is reference variable of Counter type, referring to Counter variable's instance)

2. By invoking the increase(int) method of Counter class.

Now let's check all the blocks one by one:

1.

for(Counter ctr : arr) {

    ctr.count = 100;

}

✓ It will assign 100 to count variables of three instances of Counter class.

2.

for (Counter ctr : arr) {

    int x = ctr.getCount();

    x = 100;

}

✗ x is local variable and is copy of ctr.count. Hence, assigning 100 to x will not affect the value of ctr.count.

3.

for (Counter ctr : arr) {

    ctr.getCount() = 100;

}

✗ ctr.getCount() returns int value and not a variable, hence cannot be used on left side of assignment operator. It causes compilation error.

4.

for(Counter ctr : arr) {

    ctr.increase(100 - ctr.count);

}

✓ You must have noticed that value of count variable of 3 array elements are: -1000, 539, 0. How will you change all 3 values to 100 using same expression? It is by adding 100 and subtracting current value. For example,

-1000 + 100 -(-1000) = 100

or

539 + 100 - 539 = 100

or

0 + 100 - 0 = 100

And same this is done by executing `ctr.increase(100 - ctr.count);` statement.

5.

for (Counter ctr : arr) {

    ctr.increase(100 - ctr.getCount());

}

✓ Same as block no. 4. Only difference is ctr.getCount() is used instead of ctr.count.

6.

for(Counter ctr : arr) {

    ctr.increase(-ctr.getCount() + 100);

}

✓ Same as block no. 5.

7.

for(Counter ctr : arr) {

    ctr.increase(-ctr.count + 100);

}

✓ Same as block no. 4.

Hence, out of given 7 blocks, 5 will give you expected output.

Bottom of Form

Top of Form

Question 50: **Correct**

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

1. package com.udayan.oca;
3. public class Test {
4. public static void main(String[] args) {
5. String str = "Good"; //Line 3
6. change(str); //Line 4
7. System.out.println(str); //Line 5
8. }
10. private static void change(String s) {
11. s.concat("\_Morning"); //Line 9
12. }
13. }

* 

**\_Morning**

* 

**None of the other options**

* 

**Good\_Morning**

* 

**Good**

**(Correct)**

**Explanation**

When change(String) method is called, both variable s and str refers to same String object.

Line 9 doesn't modify the passed object instead creates a new String object "Good\_Morning".

But this newly created object is not referred and hence is a candidate for GC.

When control goes back to calling method main(String[]), str still refers to "Good".

Line 5 prints "Good" on to the console.

Bottom of Form

Top of Form

Question 51: **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("SpaceStation");
6. sb.delete(5, 6).insert(5, " S").toString().toUpperCase();
7. System.out.println(sb);
8. }
9. }

* 

**SPACE STATION**

* 

**Space Station**

**(Correct)**

* 

**Space Sation**

* 

**SPACE SATION**

**Explanation**

sb - > "SpaceStation"

sb.delete(5, 6) -> "Spacetation"

sb.insert(5, " S") -> "Space Station"

sb.toString() -> Creates a new String object "Space Station"

"Space Station".toUpperCase() -> Creates another String object "SPACE STATION" but the String object is not referred and used.

Method invocation on sb modifies the same object, so after insert(5, " S") method invocation sb refers to "Space Station" and this is printed to the Console.

Bottom of Form

Top of Form

Question 52: **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("2018-06-06");
9. date.minusDays(10);
10. System.out.println(date);
11. }
12. }

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

* 

**2018-06-06**

**(Correct)**

* 

**2018-05-26**

* 

**2018-05-27**

* 

**2018-06-25**

* 

**2018-06-26**

**Explanation**

date --> {2018-06-06}.

date.minusDays(10); => as LocalDate is immutable, hence a new LocalDate object is created {2018-05-27} but no variable refers to it. date still refers to {2018-06-06}.

2018-06-06 is displayed on to the console.

Bottom of Form

Top of Form

Question 53: **Correct**

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

1. package com.udayan.oca;
3. class Rectangle {
4. private int height;
5. private int width;
7. public Rectangle(int height, int width) {
8. this.height = height;
9. this.width = width;
10. }
12. public int getHeight() {
13. return height;
14. }
16. public int getWidth() {
17. return width;
18. }
19. }
21. public class Test {
22. public static void main(String[] args) {
23. private int i = 100;
24. private int j = 200;
25. Rectangle rect = new Rectangle(i, j);
26. System.out.println(rect.getHeight() + ", " + rect.getWidth());
27. }
28. }

* 

**0, 0**

* 

**Compilation Error**

**(Correct)**

* 

**100, 200**

* 

**200, 100**

**Explanation**

i and j cannot be declared private as i and j are local variables.

Only final modifier can be used with local variables.

Bottom of Form

Top of Form

Question 54: **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 = {5, 10, 15};
6. int [] arr2 = {'A', 'B'};
7. arr1 = arr2;
8. System.out.println(arr1.length + arr2.length);
9. }
10. }

* 

**6**

* 

**4**

**(Correct)**

* 

**0**

* 

**An exception is thrown at runtime**

* 

**Compilation error**

* 

**5**

**Explanation**

Initially arr1 refers to an int array object of 3 elements.

And arr2 refers to an int array object of 2 elements [char type is compatible with int type]

When the statement `arr1 = arr2;` executes, variable arr1 copies the content of arr2, which is the address of array object containing 2 elements. Hence, arr1 also starts referring to same array object. arr1.length = 2 and arr2.length = 2.

Therefore, output is: 4

Bottom of Form

Top of Form

Question 55: **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 int i1 = 1;
7. final Integer i2 = 1;
8. final String s1 = ":ONE";
10. String str1 = i1 + s1;
11. String str2 = i2 + s1;
13. System.out.println(str1 == "1:ONE");
14. System.out.println(str2 == "1:ONE");
15. }
16. }

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

* 

**true**

**false**

**(Correct)**

* 

**true**

**true**

* 

**false**

**false**

* 

**false**

**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.

For the statement, String str1 = i1 + s1;, i1 is a final variable of int type and s1 is a final variable of String type. Hence, `i1 + s1` is a constant expression which is computed at compile-time and results in String literal "1:ONE".

This means during compilation, Java compiler translates the statement

String str1 = i1 + s1;

to

String str1 = "1:ONE";

As "1:ONE" is a String literal, hence at runtime it will be referred by String Pool.

On the other hand, for the statement, String str2 = i2 + s1;, `i2 + s1` is not a constant expression because i2 is neither of primitive type nor of String type, hence it is computed at run-time and returns a non-pool String object "1:ONE".

As, str1 refers to String Pool object "1:ONE", hence `str1 == "1:ONE"` returns true, whereas str2 refers to non-Pool String object "1:ONE" and hence `str2 == "1:ONE"` returns false.

Bottom of Form

Top of Form

Question 56: **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. private int y;
8. public String toString() {
9. return "Point(" + x + ", " + y + ")";
10. }
11. }
13. public class Test {
14. public static void main(String[] args) {
15. Point p1 = new Point();
16. p1.x = 100;
17. p1.y = 200;
19. Point p2 = new Point();
20. p2.x = 100;
21. p2.y = 200;
23. System.out.println(p1);
24. }
25. }

* 

**Point(100, 100)**

* 

**Point(100, 200)**

* 

**Compilation error**

**(Correct)**

* 

**Point(100, 0)**

* 

**Point(200, 0)**

* 

**Point(200, 200)**

* 

**Point(0, 200)**

**Explanation**

Variable y is private so it cannot be accessed outside the boundary of Point class.

p1.y and p2.y used inside Test class, cause the compilation error.

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. class M { }
4. class N extends M { }
5. class O extends N { }
6. class P extends O { }
8. public class Test {
9. public static void main(String args []) {
10. M obj = new O();
11. if(obj instanceof M)
12. System.out.print("M");
13. if(obj instanceof N)
14. System.out.print("N");
15. if(obj instanceof O)
16. System.out.print("O");
17. if(obj instanceof P)
18. System.out.print("P");
19. }
20. }

* 

**NOP**

* 

**MNO**

**(Correct)**

* 

**MOP**

* 

**MNP**

**Explanation**

M

^

N

^

O [obj refers to instance of O class]

^

P

obj instanceof M -> true

obj instanceof N -> true

obj instanceof O -> true

but

obj instanceof P -> false

Bottom of Form

Top of Form

Question 58: **Correct**

Consider below code:

1. package com.udayan.oca;
3. public class Test {
4. static Double d1;
5. static 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**

* 

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

**(Correct)**

**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 also static variable so d1.intValue(); is executed and as d1 is null hence d1.intValue() throws a NullPointerException and as a result an instance of  java.lang.ExceptionInInitializerError is thrown.

Bottom of Form

Top of Form

Question 59: **Correct**

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

1. package com.udayan.oca;
3. public class Bonus {
4. public static void main(String[] args) {
5. int $ = 80000;
6. String msg = ($ >= 50000) ? "Good bonus" : "Average bonus";
7. System.out.println(msg);
8. }
9. }

* 

**Compilation error**

* 

**Average bonus**

* 

**Good bonus**

**(Correct)**

**Explanation**

$ is valid identifier. $ = 80000

This is an example of ternary operator. First operand ($ >= 50000) is a boolean expression which is true, as 80000 >= 50000 is true.

msg will refer to "Good bonus".

Bottom of Form

Top of Form

Question 60: **Correct**

Does below code compile successfully?

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

* 

**Yes**

**(Correct)**

* 

**No**

**Explanation**

In java, it is allowed to put multiple statements on one line. E.g. below code is legal:

1. public class Test {
2. public static void main(String [] args) {
3. String symbol = "!";System.out.print("Hello ");System.out.print("World");System.out.println(symbol);
4. }
5. }

Above code is similar to:

1. public class Test {
2. public static void main(String [] args) {
3. String symbol = "!";
4. System.out.print("Hello ");
5. System.out.print("World");
6. System.out.println(symbol);
7. }
8. }

Empty statements (just the semicolon) are also allowed in java, therefore below code is also legal:

1. public class Test {
2. public static void main(String [] args) {
3. System.out.println("Hello");
4. ;
5. ;
6. ;
7. ;
8. ;
9. ;
10. ;
11. ;
12. }
13. }

As shown above, java statements (including empty statements) can be placed on one line, therefore below code is legal:

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

Bottom of Form

Top of Form

Question 61: **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 = false;
6. while(flag) {
7. System.out.println("Good Morning!");
8. }
9. }
10. }

* 

**Infinite loop**

* 

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

* 

**Compilation error**

**(Correct)**

* 

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

**Explanation**

final boolean flag = false; statement makes flag a compile time constant.

Compiler knows the value of flag, which is false at compile time and hence it gives "Unreachable Code" error.

Bottom of Form

Top of Form

Question 62: **Incorrect**

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

1. //TestStudent.java
2. package com.udayan.oca;
4. class Student {
5. String name;
6. int age;
7. boolean result;
8. double height;
9. }
11. public class TestStudent {
12. public static void main(String[] args) {
13. Student stud = new Student();
14. System.out.println(stud.name + stud.height + stud.result + stud.age);
15. }
16. }

* 

**null0false0**

* 

**null0.0true0**

* 

**null0.0ffalse0**

**(Incorrect)**

* 

**null0.0false0**

**(Correct)**

**Explanation**

name, height, result and age are instance variables of Student class. And instance variables are initialized to their respective default values.

name is initialized to null, age to 0, result to false and height to 0.0.

Statement System.out.println(stud.name + stud.height + stud.result + stud.age); prints null0.0false0

Bottom of Form

Top of Form

Question 63: **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 change(int num) {
5. num++;
6. System.out.println(num);
7. }
9. public static void main(String[] args) {
10. int i1 = 1;
11. Test.change(i1);
12. System.out.println(i1);
13. }
14. }

* 

**2  
2**

* 

**None of the other options**

* 

**2  
1**

**(Correct)**

* 

**Compilation Error**

**Explanation**

There are no compilation errors and main(String[]) method is invoked on executing Test class.

i1 = 1.

`Test.change(i1)` is executed next, contents of i1 (which is 1) is copied to the variable 'num' and method change(int) starts executing.

`num++;` increments the value of num by 1, num = 2. There are no changes to the value of variable 'i1' of main(String[]) method, which still contains 1.

`System.out.println(num);` prints 2 on to the console.

change(int) method finishes its execution and control goes back to the main(String[]) method.

`System.out.println(i1);` prints 1 on to the console.

Bottom of Form

Top of Form

Question 64: **Correct**

Consider below code:

1. //Test.java
2. import java.time.LocalDate;
3. import java.time.format.DateTimeFormatter;
5. public class Test {
6. public static void main(String [] args) {
7. LocalDate date = LocalDate.of(1987, 9, 1);
8. String str = date.format(DateTimeFormatter.ISO\_DATE\_TIME);
9. System.out.println("Date is: " + str);
10. }
11. }

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

* 

**Runtime exception**

**(Correct)**

* 

**Date is: 1987-01-09**

* 

**Given code executes successfully but output does not match with the given options**

* 

**Date is: 01-09-1987**

* 

**Date is: 1987-09-01**

**Explanation**

LocalDate object doesn't contain time part but ISO\_DATE\_TIME looks for time portion and throws exception at runtime.

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 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. for(int i=0; i<=2; i++){}
6. System.out.println(i);
7. }
8. }

* 

**3**

* 

**Compilation error**

**(Correct)**

* 

**2**

* 

**0**

**Explanation**

Variable i is declared inside for loop, hence it is not accessible beyond loop's body.

System.out.println(i); causes compilation error.

Bottom of Form

Top of Form

Question 66: **Incorrect**

Which of the following correctly imports Animal class from com.masaimara package?

* 

**Import com.masaimara.Animal;**

* 

**import com.masaimara;**

* 

**Import com.masaimara.\*;**

**(Incorrect)**

* 

**import com.masaimara.\*;**

**(Correct)**

**Explanation**

There are 2 ways to import Animal class from com.masaimara package:

import com.masaimara.\*;

import com.masaimara.Animal;

NOTE: all small case letters in import keyword.

Out of the given 4 options, only one statement is correct:

import com.masaimara.\*;

Bottom of Form

Top of Form

Question 67: **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 = 20;
6. int var = --a \* a++ + a-- - --a;
7. System.out.println("a = " + a);
8. System.out.println("var = " + var);
9. }
10. }

* 

**Compilation error**

* 

**a = 363  
var = 363**

* 

**a = 18  
var = 363**

**(Correct)**

* 

**a = 25  
var = 363**

**Explanation**

int var = --a \* a++ + a-- - --a;

int var = --a \* (a++) + (a--) - --a;

int var = (--a) \* (a++) + (a--) - (--a);

int var = ((--a) \* (a++)) + (a--) - (--a);

int var = (((--a) \* (a++)) + (a--)) - (--a);

int var = ((19 \* (a++)) + (a--)) - (--a); //a = 19

int var = ((19 \* 19) + (a--)) - (--a); //a = 20

int var = (361 + 20) - (--a); //a = 19

int var = 381 - (--a); //a = 19

int var = 381 - 18; //a = 18

int var = 363 // a = 18

So,

a = 18

var = 363

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. m(1);
6. }
8. private static void m(Object obj) {
9. System.out.println("Object version");
10. }
12. private static void m(Number obj) {
13. System.out.println("Number version");
14. }
16. private static void m(Double obj) {
17. System.out.println("Double version");
18. }
19. }

* 

**Compilation error**

* 

**Object version**

* 

**Double version**

* 

**Number version**

**(Correct)**

**Explanation**

There are 3 overloaded method m. Note all the numeric wrapper classes (Byte, Short, Integer, Long, Float and Double) extend from Number and Number extends from Object.

Compiler either does implicit casting or Wrapping but not both. 1 is int literal, Java compiler can't implicit cast it to double and then box it to Double rather it boxes i to Integer and as Number is the immediate super class of Integer so Number version refers to Integer object.

Number version is printed on to the console.

Bottom of Form

Top of Form

Question 69: **Correct**

Consider below code:

1. //Test.java
2. package com.udayan.oca;
4. import java.util.ArrayList;
5. import java.util.List;
7. public class Test {
8. public static void main(String[] args) {
9. List<String> list = new ArrayList<>();
10. list.add(0, "Array");
11. list.set(0, "List");
13. System.out.println(list);
14. }
15. }

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

* 

**[List, Array]**

* 

**[Array]**

* 

**[Array, List]**

* 

**[List]**

**(Correct)**

* 

**An exception is thrown at runtime**

**Explanation**

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

list.set(0, "List"); means replace the current element at index 0 with the passed element "List". So after this operation, list --> [List]. In the console, [List] is printed.

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. System.out.println("Hello" + 1 + 2 + 3 + 4);
6. }
7. }

* 

**Hello10**

* 

**Hello 10**

* 

**Hello19**

* 

**Hello1234**

**(Correct)**

**Explanation**

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

"Hello" + 1 + 2 + 3 + 4

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

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

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

[Let us solve it now, + operator with String behaves as concatenation operator.]

= (("Hello1" + 2) + 3) + 4

= ("Hello12" + 3) + 4

= "Hello123" + 4

= "Hello1234"

Bottom of Form