**Exception Handling - 2, Working with Arrays and Collections - 1 - Results**

Return to review

Chart

Pie chart with 4 slices.

End of interactive chart.

Attempt 3

All knowledge areas

All questions

Top of Form

Question 1: **Correct**

**Question ID: UKOCP12735**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Scanner;
5. public class Test {
6. public static void main(String[] args) {
7. try(Scanner scan = new Scanner(System.in)) {
8. String s = scan.nextLine();
9. System.out.println(s);
10. scan = null;
11. }
12. }
13. }

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

* 

**Normal Termination**

* 

**Exception is thrown at runtime**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP12735:**

Resources used in try-with-resources statement are implicitly final, which means they can't be reassigned.

scan = null; will fail to compile as we are trying to assign null to variable scan.

Bottom of Form

Top of Form

Question 2: **Correct**

**Question ID: UKOCP66550**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Scanner;
5. public class Test {
6. public static void main(String[] args) {
7. System.out.print("Enter some text: ");
8. try(Scanner scan = new Scanner(System.in)) {
9. String s = scan.nextLine();
10. System.out.println(s);
11. scan.close();
12. scan.nextLine();
13. }
14. }
15. }

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

User input is: HELLO

* 

**Compilation error**

* 

**Runtime Exception**

**(Correct)**

* 

**On execution program terminates successfully after printing 'HELLO' on to the console**

**Explanation**

**UKOCP66550:**

Even though Scanner is created in try-with-resources block, calling close() method explicitly doesn't cause any problem.

Scanner class allows to invoke close() method multiple times.

But once Scanner object is closed, other search operations should not be invoked. If invoked on closed scanner, IllegalStateException is thrown.

Bottom of Form

Top of Form

Question 3: **Correct**

**Question ID: UKOCP34504**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Scanner;
5. public class Test {
6. public static void main(String[] args) {
7. System.out.print("Enter some text: ");
8. try(Scanner scan = new Scanner(System.in)) {
9. String s = scan.nextLine();
10. System.out.println(s);
11. scan.close();
12. scan.close();
13. }
14. }
15. }

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

User input is: HELLO

* 

**Compilation error**

* 

**Runtime Exception**

* 

**On execution program terminates successfully after printing 'HELLO' on to the console**

**(Correct)**

**Explanation**

**UKOCP34504:**

Even though Scanner is created in try-with-resources block, calling close() method explicitly doesn't cause any problem.

Scanner class allows to invoke close() method multiple times. In this case, it will be called 3 times: twice because of scan.close() and once because of try-with-resources statement.

'HELLO' is printed on to the console and program terminates successfully.

Bottom of Form

Top of Form

Question 4: **Correct**

**Question ID: UKOCP30148**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.FileNotFoundException;
4. import java.io.FileReader;
6. public class Test {
7. public static void main(String[] args) {
8. try(FileReader fr = new FileReader("C:/temp.txt")) {
10. } catch (FileNotFoundException e) {
11. e.printStackTrace();
12. }
13. }
14. }

Does above code compile successfully?

* 

**YES**

* 

**NO**

**(Correct)**

**Explanation**

**UKOCP30148:**

close() method of FileReader class throws IOException, which is a checked exception and hence handle or declare rule applies in this case.

As main method neither declares to throw IOException nor a catch block is available for IOException, hence code doesn't compile.

Bottom of Form

Top of Form

Question 5: **Correct**

**Question ID: UKOCP85294**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.PrintWriter;
5. public class Test {
6. public static void main(String[] args) {
7. try(PrintWriter writer = new PrintWriter(System.out)) {
8. writer.println("Hello");
9. } catch(Exception ex) {
10. writer.close();
11. }
12. }
13. }

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

* 

**Hello**

* 

**Compilation error**

**(Correct)**

* 

**Program ends abruptly**

**Explanation**

**UKOCP85294:**

Scope of writer variable is with-in the boundary of try-with-resources block. It is not accessible inside catch block and hence 'writer.close()' causes compilation failure.

NOTE: PrintWriter constructor and println method don't throw any exception but it is OK to catch Exception type. Compiler allows to catch Exception type even if code within try block doesn't throw any exception.

Bottom of Form

Top of Form

Question 6: **Correct**

**Question ID: UKOCP86141**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.PrintWriter;
5. public class Test {
6. public static void main(String[] args) {
7. try(PrintWriter writer = null) {
8. System.out.println("HELLO");
9. }
10. }
11. }

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

* 

**HELLO**

**(Correct)**

* 

**Compilation error**

* 

**NullPointerException is thrown at runtime**

**Explanation**

**UKOCP86141:**

For null resources, close() method is not called and hence NullPointerException is not thrown at runtime.

HELLO is printed on to the console.

Bottom of Form

Top of Form

Question 7: **Correct**

**Question ID: UKOCP74940**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.PrintWriter;
5. public class Test {
6. public static void main(String[] args) {
7. try(PrintWriter writer;) {
8. writer = new PrintWriter(System.out);
9. writer.println("HELLO");
10. }
11. }
12. }

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

* 

**HELLO**

* 

**Compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP74940:**

Resources used in try-with-resources statement must be initialized.

'try(PrintWriter writer;)' causes compilation error as writer is not initialized in this statement.

Bottom of Form

Top of Form

Question 8: **Correct**

**Question ID: UKOCP22022**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class MyResource implements AutoCloseable {
4. public void execute() {
5. System.out.println("Executing");
6. }
8. @Override
9. public void close() {
10. System.out.println("Closing");
11. }
12. }
14. public class Test {
15. public static void main(String[] args) {
16. try(MyResource resource = new MyResource()) {
17. resource.execute();
18. }
19. }
20. }

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

* 

**Compilation Error**

* 

**Executing**

* 

**Executing**

**Closing**

**(Correct)**

* 

**Runtime Exception**

**Explanation**

**UKOCP22022:**

close() method in AutoCloseable interface has below declaration:

void close() throws Exception;

MyResource class correctly overrides close() method.

try-with-resources statement internally invokes resource.close() method after executing resource.execute().

Output is:

Executing

Closing

Bottom of Form

Top of Form

Question 9: **Correct**

**Question ID: UKOCP82566**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class MyResource implements AutoCloseable {
4. public void execute() {
5. System.out.println("Executing");
6. }
8. @Override
9. public void close() throws Exception {
10. System.out.println("Closing");
11. }
12. }
14. public class Test {
15. public static void main(String[] args) {
16. try(MyResource resource = new MyResource()) {
17. resource.execute();
18. }
19. }
20. }

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

* 

**Compilation Error**

**(Correct)**

* 

**Executing**

* 

**Executing**

**Closing**

* 

**Runtime Exception**

**Explanation**

**UKOCP82566:**

close() method in AutoCloseable interface has below declaration:

void close() throws Exception;

MyResource class correctly overrides close() method.

try-with-resources statement internally invokes resource.close() method after executing resource.execute().

Overriding close method declares to throw Exception (checked exception) and hence handle or declare rule must be followed.

As main method neither declares to throw Exception nor provides catch block for Exception type, hence try-with-resources statement causes compilation error.

Bottom of Form

Top of Form

Question 10: **Correct**

**Question ID: UKOCP21173**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class MyResource implements AutoCloseable {
4. @Override
5. public void close() {
6. System.out.println("Closing");
7. }
8. }
10. public class Test {
11. public static void main(String[] args) {
12. try(AutoCloseable resource = new MyResource()) {
14. }
15. }
16. }

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

* 

**Compilation error in MyResource class**

* 

**Compilation error in Test class**

**(Correct)**

* 

**Closing**

**Explanation**

**UKOCP21173:**

close() method in AutoCloseable interface has below declaration:

void close() throws Exception;

MyResource class correctly overrides close() method.

try-with-resources statement internally invokes resource.close() method.

resource is of AutoCloseable type, so compiler checks the close() method declaration of AutoCloseable interface.

close() method in AutoCloseable interface declares to throw Exception (checked exception) and hence handle or declare rule must be followed.

As main method neither declares to throw Exception nor provides catch block for Exception type, hence try-with-resources statement causes compilation error.

Bottom of Form

Top of Form

Question 11: **Correct**

**Question ID: UKOCP49504**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.IOException;
4. import java.sql.SQLException;
6. class MyResource implements AutoCloseable {
7. @Override
8. public void close() throws IOException{
9. throw new IOException("IOException");
10. }
12. public void execute() throws SQLException {
13. throw new SQLException("SQLException");
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. try(MyResource resource = new MyResource()) {
20. resource.execute();
21. } catch(Exception e) {
22. System.out.println(e.getMessage());
23. }
24. }
25. }

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

* 

**IOException**

* 

**SQLException**

**(Correct)**

* 

**Compilation error**

**Explanation**

**UKOCP49504:**

execute() method throws an instance of SQLException.

Just before finding the matching handler, Java runtime executes close() method. This method throws an instance of IOException but it gets suppressed and an instance of SQLException is thrown.

e.getMessage() prints SQLException on to the console.

NOTE: e.getSuppressed() returns Throwable [] and this helps to get all the suppressed exceptions.

Bottom of Form

Top of Form

Question 12: **Correct**

**Question ID: UKOCP61623**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.IOException;
4. import java.sql.SQLException;
6. class MyResource implements AutoCloseable {
7. @Override
8. public void close() throws IOException{
9. throw new IOException("IOException");
10. }
12. public void execute() throws SQLException {
13. throw new SQLException("SQLException");
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. try(MyResource resource = new MyResource()) {
20. resource.execute();
21. } catch(Exception e) {
22. System.out.println(e.getSuppressed()[0].getMessage());
23. }
24. }
25. }

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

* 

**IOException**

**(Correct)**

* 

**SQLException**

* 

**Compilation error**

**Explanation**

**UKOCP61623:**

execute() method throws an instance of SQLException.

Just before finding the matching handler, Java runtime executes close() method. This method throws an instance of IOException but it gets suppressed and an instance of SQLException is thrown.

e.getSuppressed() returns Throwable [] whose 1st element is the instance of suppressed exception, IOException.

'e.getSuppressed()[0].getMessage()' prints IOException on to the console.

Bottom of Form

Top of Form

Question 13: **Correct**

**Question ID: UKOCP60777**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.io.IOException;
4. import java.sql.SQLException;
6. class MyResource implements AutoCloseable {
7. @Override
8. public void close() throws IOException{
10. }
12. public void execute() throws SQLException {
13. throw new SQLException("SQLException");
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. try(MyResource resource = new MyResource()) {
20. resource.execute();
21. } catch(Exception e) {
22. System.out.println(e.getSuppressed().length);
23. }
24. }
25. }

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

* 

**1**

* 

**0**

**(Correct)**

* 

**NullPointerException is thrown**

**Explanation**

**UKOCP60777:**

execute() method throws an instance of SQLException.

Just before finding the matching handler, Java runtime executes close() method. This method executes successfully.

An instance of SQLException is thrown. No exceptions was suppressed so 'e.getSuppressed()' returns Throwable [] of size 0.

'e.getSuppressed().length' prints 0 on to the console.

Bottom of Form

Top of Form

Question 14: **Correct**

**Question ID: UKOCP23357**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Resource1 implements AutoCloseable {
4. @Override
5. public void close() {
6. System.out.println("Resource1");
7. }
8. }
10. class Resource2 implements AutoCloseable {
11. @Override
12. public void close() {
13. System.out.println("Resource2");
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. try(Resource1 r1 = new Resource1(); Resource2 r2 = new Resource2()) {
20. System.out.println("Test");
21. }
22. }
23. }

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

* 

**Test**

**Resource1**

**Resource2**

* 

**Test**

**Resource2**

**Resource1**

**(Correct)**

* 

**Compilation Error**

**Explanation**

**UKOCP23357:**

Resources are closed in the reverse order of their declaration. So r2 is closed first and then r1.

Bottom of Form

Top of Form

Question 15: **Correct**

**Question ID: UKOCP83414**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Resource1 {
4. public void close() {
5. System.out.println("Resource1");
6. }
7. }
9. class Resource2 {
10. public void close() {
11. System.out.println("Resource2");
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. try(Resource1 r1 = new Resource1(); Resource2 r2 = new Resource2()) {
18. System.out.println("Test");
19. }
20. }
21. }

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

* 

**Test**

**Resource1**

**Resource2**

* 

**Test**

**Resource2**

**Resource1**

* 

**Compilation Error**

**(Correct)**

**Explanation**

**UKOCP83414:**

Classes used in try-with-resources statement must implement java.lang.AutoCloseable or its sub interfaces such as java.io.Closeable.

As Resource1 and Resource2 don't implement AutoCloseable interface, hence try-with-resources statement causes compilation error.

Bottom of Form

Top of Form

Question 16: **Correct**

**Question ID: UKOCP78068**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Resource implements AutoCloseable {
4. public void close() {
5. System.out.println("CLOSE");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. try(Resource r = null) {
12. System.out.println("HELLO");
13. }
14. }
15. }

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

* 

**HELLO**

**(Correct)**

* 

**HELLO**

**CLOSE**

* 

**Compilation error**

* 

**NullPointerException is thrown at runtime**

**Explanation**

**UKOCP78068:**

For null resources, close() method is not called and hence NullPointerException is not thrown at runtime.

HELLO is printed on to the console.

Bottom of Form

Top of Form

Question 17: **Correct**

**Question ID: UKOCP22028**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Resource implements AutoCloseable {
4. public void close() {
5. System.out.println("CLOSE");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. try(Resource r = null) {
12. r = new Resource();
13. System.out.println("HELLO");
14. }
15. }
16. }

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

* 

**HELLO**

* 

**HELLO**

**CLOSE**

* 

**Compilation error**

**(Correct)**

* 

**NullPointerException is thrown at runtime**

**Explanation**

**UKOCP22028:**

Variable r is implicitly final and hence can't be re-initialized.

'r = new Resource();' causes compilation error.

Bottom of Form

Top of Form

Question 18: **Correct**

**Question ID: UKOCP62956**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Resource1 implements AutoCloseable {
4. public void m1() throws Exception {
5. System.out.print("A");
6. throw new Exception("B");
7. }
9. public void close() {
10. System.out.print("C");
11. }
12. }
14. class Resource2 implements AutoCloseable {
15. public void m2() {
16. System.out.print("D");
17. }
19. public void close() throws Exception {
20. System.out.print("E");
21. }
22. }
24. public class Test {
25. public static void main(String[] args) {
26. try (Resource1 r1 = new Resource1();
27. Resource2 r2 = new Resource2()) {
28. r1.m1();
29. r2.m2();
30. } catch (Exception e) {
31. System.out.print(e.getMessage());
32. }
33. }
34. }

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

* 

**Compilation error**

* 

**ABEC**

* 

**ABCE**

* 

**ACEB**

* 

**AECB**

**(Correct)**

**Explanation**

**UKOCP62956:**

AutoCloseable interface has abstract method: void close() throws Exception;

Both Resource1 and Resource2 implement the close() method correctly and main method specified handler for Exception type, hence no compilation error.

Resources are always closes, even in case of exceptions. And in case of multiple resources, these are closed in the reverse order of their declaration. So r2 is closed first and then r1. Output will have 'EC' together.

r1.m1(); prints 'A' on to the console. An exception (with message 'B') is thrown so close methods are invoked.

After close() methods of r2 and r1 are invoked successfully, output will be: 'AEC'.

Exception is caught in main method and e.getMessage() returns 'B'.

So the overall output will be; 'AECB'.

Bottom of Form

Top of Form

Question 19: **Correct**

**Question ID: UKOCP10052**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
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. }

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

* 

**Compilation error**

* 

**An exception is thrown at runtime**

* 

**0**

* 

**4**

**(Correct)**

* 

**6**

* 

**5**

**Explanation**

**UKOCP10052:**

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 20: **Correct**

**Question ID: UKOCP38841**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int[] arr1 = { 10, 100, 1000 }; //Line n1
6. char[] arr2 = { 'x', 'y', 'z' }; //Line n2
7. arr1 = arr2; // Line n3
8. for (int i = 0; i < arr1.length; i++) {
9. System.out.print(arr1[i] + " "); //Line n4
10. }
11. }
12. }

ASCII code of 'x' is 120, 'y' is 121 and z is 122.

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

* 

**10 100 1000**

* 

**x y z**

* 

**120 121 122**

* 

**0 0 0**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP38841:**

Line n1 creates an int array object of 3 elements: 10, 100, 1000 and arr1 refers to it.

Line n2 creates an char array object of 3 elements: 'x', 'y', 'z'.

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

Bottom of Form

Top of Form

Question 21: **Correct**

**Question ID: UKOCP51809**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. String arr1 [], arr2, arr3 = null; //Line n1
6. arr1 = new String[2];
7. arr1[0] = "A";
8. arr1[1] = "B";
9. arr2 = arr3 = arr1; //Line n2
10. System.out.println(String.join("-", arr2)); //Line n3
11. }
12. }

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

* 

**Line n1 causes compilation error**

* 

**Line n2 causes compilation error**

**(Correct)**

* 

**Line n3 causes compilation error**

* 

**It executes successfully and prints A-B on to the console**

* 

**It executes successfully and prints B-A on to the console**

* 

**It executes successfully and prints A on to the console**

* 

**It executes successfully and prints B on to the console**

**Explanation**

**UKOCP51809:**

arr1 is of String[] type, where as arr2 and arr3 are of String type. As all three arr1, arr2 and arr3 are of reference type, hence null can be assigned to all these variables. Line n1 compiles successfully.

Statement at Line n2: arr2 = arr3 = arr1;

=> arr2 = (arr3 = arr1); //assignment operator is right to left associative.

arr3 is of String type and arr1 is of String [] type, hence (arr3 = arr1) causes compilation error.

Though you had to select one correct option, hence no need to look further but I am providing explanation for Line n3 as well.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:

1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,

String.join(".", "A", "B", "C"); returns "A.B.C"

String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"

String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, "A", "B"); throws NullPointerException

String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

String str = null; String.join("-", str); returns "null"

String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join​(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,

String.join(".", List.of("A", "B", "C")); returns "A.B.C"

String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

List<String> list = new ArrayList<>(); list.add("A"); list.add(null); String.join("::", list); returns "A::null"

Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Based on above points, `String.join("-", arr2)` compiles successfully as arr2 is of String type.

Bottom of Form

Top of Form

Question 22: **Correct**

**Question ID: UKOCP62971**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
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. }

Which of the following is true for the above code?

* 

**Above code causes compilation error**

* 

**0 is printed on to the console**

* 

**null is printed on to the console**

* 

**An exception is thrown at runtime**

**(Correct)**

**Explanation**

**UKOCP62971:**

Given code compiles successfully.

arr refers to an array object of size 0, this 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 23: **Correct**

**Question ID: UKOCP33156**

Consider below code of Test.java file:

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

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

* 

**An exception is thrown at runtime**

* 

**Compilation Error**

* 

**It prints null**

* 

**It prints some text containing @ symbol**

**(Correct)**

**Explanation**

**UKOCP33156:**

Variable 'arr' refers to an array object of String of 7 elements and it contains the memory address of String array object.

'arr' is of reference type, therefore when `System.out.println(arr);` is executed, toString() method defined in Object class is invoked, which returns <fully qualified name of internal array class>@<hexadecimal representation of hashcode>. That is why some text containing @ symbol is printed on to the console.

Bottom of Form

Top of Form

Question 24: **Incorrect**

**Question ID: UKOCP70103**

Consider below code of Test.java file:

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

And below options:

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

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

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

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

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

* 

**Only one option**

* 

**Only two options**

* 

**Only three options**

**(Incorrect)**

* 

**All four options**

**(Correct)**

* 

**None of the other options**

**Explanation**

**UKOCP70103:**

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

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

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

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

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

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

Hence, all the 4 options are valid.

Bottom of Form

Top of Form

Question 25: **Incorrect**

**Question ID: UKOCP14416**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
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 = 0; n < arr.length; n++**

* 

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

**(Correct)**

* 

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

**(Correct)**

* 

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

**(Correct)**

* 

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

**(Incorrect)**

**Explanation**

UKOCP14416:

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 26: **Correct**

**Question ID: UKOCP71435**

Given code of Test.java file:

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

And below statements:

1. short arr [] = new short[2];

2. byte [] arr = new byte[10];

3. short [] arr; arr = new short[3];

4. short [2] arr;

5. short [3] arr;

6. int [] arr = new int[]{100, 100};

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

8. short [] arr = {};

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

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

* 

**Only one option**

* 

**Only two options**

* 

**Only three options**

**(Correct)**

* 

**Only four options**

* 

**None of the given options**

* 

**More than four options**

**Explanation**

**UKOCP71435:**

Let's check all the statements one by one:

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

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

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

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

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

6. int [] arr = new int[]{100, 100}; => ✗ 'arr' refers to an int array object of size 2 and both array elements have value 100. arr[2] will throw ArrayIndexOutOfBoundsException at runtime.

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

8. short [] arr = {}; => ✗ 'arr' refers to a short array object of 0 size. so arr[1] will throw ArrayIndexOutOfBoundsException at runtime.

9. short [] arr = new short[2]{5, 10}; => ✗ Array's size can't be specified, if you use {} to assign values to array elements.

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

Bottom of Form

Top of Form

Question 27: **Correct**

**Question ID: UKOCP35335**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. double [] arr = new int[2]; //Line n1
6. System.out.println(arr[0]); //Line n2
7. }
8. }

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

* 

**0**

* 

**0.0**

* 

**Line n1 causes compilation error**

**(Correct)**

* 

**Line n2 causes runtime exception**

**Explanation**

**UKOCP35335:**

int variable can easily be assigned to double type but double [] and int [] are not compatible. Hence, int [] object cannot be assigned to double [] type. Line n1 causes compilation error.

Bottom of Form

Top of Form

Question 28: **Correct**

**Question ID: UKOCP11388**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. String[] arr = { "A", "B", "C", "D" }; // Line n1
6. arr[0] = arr[1]; // Line n2
7. arr[1] = "E"; // Line n3
8. for (String s : arr) {
9. System.out.print(s + " ");
10. }
11. }
12. }

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

* 

**Compilation error**

* 

**An exception is thrown at runtime**

* 

**B E C D**

**(Correct)**

* 

**E E C D**

* 

**A E C D**

* 

**B B C D**

* 

**A B C D**

**Explanation**

**UKOCP11388:**

Line n1 creates a String [] object of 4 elements. arr[0] --> "A", arr[1] --> "B", arr[2] --> "C" and arr[3] --> "D".

At Line n2, arr[0] --> "B".

At Line n3, arr[1] --> "E".

So, after Line n3, arr refers to {"B", "E", "C", "D"}.

for-each loop prints the array elements referred by arr and hence the output is: B E C D

Bottom of Form

Top of Form

Question 29: **Correct**

**Question ID: UKOCP14896**

Given code of Test.java file:

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

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

* 

**true**

* 

**false**

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**Compilation error**

**Explanation**

**UKOCP14896:**

new String[1] creates a String [] object of one element and as all the elements of array are initialized to respective zeros (in case of primitive type) or null (in case of reference type), arr[0] refers to null.

isBlank() method of String class (available since Java 11) returns true if the string is empty or contains only white space codepoints, otherwise false. As isBlank() is available, hence `arr[0].isBlank()` compiles successfully.

But as method 'isBlank()' is invoked on null reference (arr[0] refers to null), so NullPointerException is thrown at runtime.

Bottom of Form

Top of Form

Question 30: **Correct**

**Question ID: UKOCP14413**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. StringBuilder[] sb = new StringBuilder[2]; //Line n1
6. sb[0] = new StringBuilder("PLAY"); //Line n2
8. boolean[] flag = new boolean[1]; //Line n3
10. if (flag[0]) { //Line n4
11. sb[1] = new StringBuilder("GROUP"); //Line n5
12. }
14. System.out.println(String.join("-", sb)); //Line n6
15. }
16. }

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

* 

**PLAY**

* 

**GROUP**

* 

**PLAY-GROUP**

* 

**PLAY-null**

**(Correct)**

* 

**-**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

**UKOCP14413:**

Line n1 declares a StringBuilder [] object of 2 elements and both the elements are initialized to default value null. So, sb --> {null, null}.

Line n2 initializes 1st array element to ["PLAY"]. So, sb --> {["PLAY"], null}.

Line n3 creates a boolean array object of 1 element and this element is initialized to default value of boolean, which is false. flag --> {false}.

At Line n4, boolean expression of if-block is 'flag[0]', which evaluates to false, control doesn't enter if block and Line n5 is not executed.

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:

1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,

String.join(".", "A", "B", "C"); returns "A.B.C"

String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"

String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, "A", "B"); throws NullPointerException

String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

String str = null; String.join("-", str); returns "null"

String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join​(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,

String.join(".", List.of("A", "B", "C")); returns "A.B.C"

String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

List<String> list = new ArrayList<>(); list.add("A"); list.add(null); String.join("::", list); returns "A::null"

Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

As StringBuilder implements CharSequence, hence `String.join("-", sb)` is valid as 'sb' refers to StringBuilder[] object.

As per above explanation, `String.join("-", sb)` will return PLAY-null and Line n6 will print PLAY-null on to the console.

Bottom of Form

Top of Form

Question 31: **Correct**

**Question ID: UKOCP45160**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int [] arr = {10, 20, 30}; //Line n1
6. int i = 0;
7. arr[i++] = arr[++i] = 40; //Line n2
8. for(var x : arr) //Line n3
9. System.out.println(x);
10. }
11. }

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

* 

**Compilation error at Line n2**

* 

**An exception is thrown by Line n2**

* 

**10**

**20**

**30**

* 

**10**

**40**

**30**

* 

**40**

**40**

**30**

* 

**10**

**40**

**40**

* 

**40**

**20**

**40**

**(Correct)**

**Explanation**

**UKOCP45160:**

At Line n1, an int [] object of three elements is created and 'arr' refers to this array object.

arr[0] = 10, arr[1] = 20 and arr[2] = 30;

Given expression at Line n2:

arr[i++] = arr[++i] = 40;

Multiple assignment operators are available, so let's group it first.

=> arr[i++] = (arr[++i] = 40); //Assignment operator is right to left associative

Above expression is valid, hence Line n2 compiles successfully.

Let's solve the expression now. Left operand is 'arr[i++]' and right operand is '(arr[++i] = 40)'. Left operand is evaluated first.

=> arr[0] = (arr[++i] = 40); //i = 1

Right hand operand is evaluated next.

=> arr[0] = (arr[2] = 40); //i = 2

=> arr[0] = 40; //i = 2, arr[2] = 40.

Hence after Line n2, arr refers to int [] object {40, 20, 40}.

Local variable Type inference was added in JDK 10.

Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,

var x = "Java"; //x infers to String

var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

For Line n3, variable 'x' infers to int type.

Given loop prints below on to the console:

40

20

40

Bottom of Form

Top of Form

Question 32: **Correct**

**Question ID: UKOCP21192**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. String[] arr = { "L", "I", "V", "E" }; //Line n1
6. int i = -2;
8. if (i++ == -1) { //Line n2
9. arr[-(--i)] = "F"; //Line n3
10. } else if (--i == -2) { //Line n4
11. arr[-++i] = "O"; //Line n5
12. }
14. System.out.println(String.join("", arr)); //Line n6
15. }
16. }

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

* 

**Compilation error**

* 

**An exception is thrown at runtime**

* 

**LIVE**

* 

**LIFE**

* 

**LIVO**

* 

**LOVE**

**(Correct)**

* 

**LIOE**

**Explanation**

**UKOCP21192:**

Line n1 creates a String [] object of 4 elements and arr refers to this array object. arr[0] = "L", arr[1] = "I", arr[2] = "V" and arr[3] = "E".

i = -2.

Boolean expression of Line n2: i++ == -1

=> (i++) == -1 //As Post-increment operator ++ has higher precedence over ==

=> -2 == -1 //i = -1, value of i is used in the expression and then incremented.

=> false and hence Line n3 is not executed.

But there is no issue with Line n3 and it compiles successfully.

Boolean expression of Line n4 is evaluated next:

--i == -2 //i = -1

=> (--i) == -2 //As Pre-decrement operator -- has higher precedence over ==

=> -2 == -2 //i = -2, value of i is decremented first and then used in the expression.

=> true and hence Line n5 is executed next.

Line n5:

arr[-++i] = "O"; //i = -2

=> arr[-(++i)] = "O"; //Unary minus '-' and pre-increment '++' operators have same precedence

=> arr[-(-1)] = "O"; //i = -1, value of i is incremented first and then used in the expression.

=> arr[1] = "O"; //2nd array element is changed to "O".

Hence after Line n5, arr refers to {"L", "O", "V", "E"}

Static overloaded method join(...) was added in JDK 1.8 and has below declarations:

1. public static String join(CharSequence delimiter, CharSequence... elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,

String.join(".", "A", "B", "C"); returns "A.B.C"

String.join("+", new String[]{"1", "2", "3"}); returns "1+2+3"

String.join("-", "HELLO"); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, "A", "B"); throws NullPointerException

String [] arr = null; String.join("-", arr); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

String str = null; String.join("-", str); returns "null"

String.join("::", new String[] {"James", null, "Gosling"}); returns "James::null::Gosling"

2. public static String join​(CharSequence delimiter, Iterable<? extends CharSequence> elements) {...}: It returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.

For example,

String.join(".", List.of("A", "B", "C")); returns "A.B.C"

String.join(".", List.of("HELLO")); returns "HELLO"

If delimiter is null or elements refer to null, then NullPointerException is thrown. e.g.,

String.join(null, List.of("HELLO")); throws NullPointerException

List<String> list = null; String.join("-", list); throws NullPointerException

But if single element is null, then "null" is considered. e.g.,

List<String> list = new ArrayList<>(); list.add("A"); list.add(null); String.join("::", list); returns "A::null"

Please note: String.join("-", null); causes compilation error as compiler is unable to tag this call to specific join(...) method. It is an ambiguous call.

Hence, `System.out.println(String.join("", arr));` prints LOVE on to the console.

Bottom of Form

Top of Form

Question 33: **Correct**

**Question ID: UKOCP47340**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. var x = new int[]{1};
6. var y = new int[]{2};
7. var z = new int[]{3};
8. System.out.println((x = y = z)[0] + y[0] + z[0]);
9. }
10. }

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

* 

**Compilation error**

* 

**An exception is thrown at runtime**

* 

**3**

* 

**6**

* 

**7**

* 

**8**

* 

**9**

**(Correct)**

**Explanation**

**UKOCP47340:**

`var x = new int[]{1};`: Right side expression creates an int[] object of one element and 1 is assigned to that element. Target-type (int[]) is specified on the right side, hence this statement compiles successfully. x infers to int[] type.

Similarly y and z also infer to int[] type.

System.out.println((x = y = z)[0] + y[0] + z[0]); //x --> {1}, y --> {2} and z --> {3}

=> System.out.println((x = y)[0] + y[0] + z[0]); //x --> {1}, y --> {3} and z --> {3}

=> System.out.println(x[0] + y[0] + z[0]); //x --> {3}, y --> {3} and z --> {3}

=> System.out.println(3 + 3 + z[0]);

=> System.out.println(6 + 3);

=> System.out.println(9);

9 is printed on to the console.

Bottom of Form

Top of Form

Question 34: **Correct**

**Question ID: UKOCP46496**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
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 will be the result of compiling and executing Test class?

* 

**An Exception is thrown at runtime**

* 

**true:true**

* 

**false:false**

* 

**null:null**

**(Correct)**

**Explanation**

**UKOCP46496:**

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 35: **Correct**

**Question ID: UKOCP78048**

Consider below code of Test.java file:

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

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

* 

**An exception is thrown at runtime**

**(Correct)**

* 

**0.0**

* 

**0**

* 

**0.00**

* 

**Compilation error**

**Explanation**

**UKOCP78048:**

`var arr = new Double[2];`: Right side expression creates an Double[] object of 2 elements. Target-type (Double[]) is specified on the right side, hence this statement compiles successfully. arr infers to Double[] type.

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

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

Bottom of Form

Top of Form

Question 36: **Correct**

**Question ID: UKOCP60755**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. int[] a1 = { 10, 15, 20 };
8. int[] a2 = { 10, 20, 30 };
9. System.out.println(Arrays.mismatch(a1, a2) + Arrays.compare(a1, a2));
10. }
11. }

What will be the result of compiling and executing above code?

* 

**1**

* 

**-1**

* 

**0**

**(Correct)**

* 

**5**

* 

**-5**

* 

**6**

* 

**-4**

**Explanation**

**UKOCP60755:**

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

For the given code, there is a mismatch at 2nd element, which is index 1. Hence, Arrays.mismatch(a1, a2) returns 1.

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

For comparing the array contents, these methods take help of static compare(x, y) method defined in respective Wrapper classes. For example, Arrays.compare(int[] a, int[] b) uses Integer.compare(int x, int y) to compare array contents.

So, for the exam, you need to have an idea of how static compare(...) methods are implemented in these wrapper classes:

For Character, Byte & Short; compare method returns x - y.

For Integer and Long; compare method returns -1 if x < y, it returns 1 if x > y and it returns 0 if x == y.

For Float and Double; logic is almost same as Integer type but logic is bit complex for finding equality of two floats or two doubles (not part of the exam).

For Boolean; compare method returns 0 if x == y, 1 if x is true and -1 if x is false.

For the given code, Arrays.compare(a1, a2) returns -1, which is the result of Integer.compare(15, 20).

Therefore,

System.out.println(Arrays.mismatch(a1, a2) + Arrays.compare(a1, a2));

=> System.out.println(1 + -1);

=> System.out.println(0);

Above statement prints 0 on to the console.

Bottom of Form

Top of Form

Question 37: **Incorrect**

**Question ID: UKOCP40704**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. char[] c1 = { 'T', 'A', 'L', 'L' };
8. char[] c2 = { 'T', 'A', 'L', 'K' };
9. System.out.println(Arrays.mismatch(c1, c2) \* Arrays.compare(c1, c2));
10. }
11. }

What will be the result of compiling and executing above code?

* 

**0**

* 

**4**

* 

**-4**

* 

**3**

**(Correct)**

* 

**-3**

* 

**2**

* 

**-2**

* 

**1**

* 

**-1**

**(Incorrect)**

**Explanation**

**UKOCP40704:**

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

For the given code, there is a mismatch at 4th element, which is index 3. Hence, Arrays.mismatch(c1, c2) returns 3.

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

For comparing the array contents, these methods take help of static compare(x, y) method defined in respective Wrapper classes. For example, Arrays.compare(char[] a, char[] b) uses Character.compare(char x, char y) to compare array contents.

So, for the exam, you need to have an idea of how static compare(...) methods are implemented in these wrapper classes:

For Character, Byte & Short; compare method returns x - y.

For Integer and Long; compare method returns -1 if x < y, it returns 1 if x > y and it returns 0 if x == y.

For Float and Double; logic is almost same as Integer type but logic is bit complex for finding equality of two floats or two doubles (not part of the exam).

For Boolean; compare method returns 0 if x == y, 1 if x is true and -1 if x is false.

For the given code, Arrays.compare(c1, c2) returns 1, which is the result of Character.compare('L', 'K').

Therefore,

System.out.println(Arrays.mismatch(c1, c2) \* Arrays.compare(c1, c2));

=> System.out.println(3 \* 1);

=> System.out.println(3);

Above statement prints 3 on to the console.

Bottom of Form

Top of Form

Question 38: **Correct**

**Question ID: UKOCP49525**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. byte[] arr1 = { 5, 10, 15, 20 };
8. byte[] arr2 = { 5, 10, 15, 17 };
9. System.out.println(Arrays.\_\_\_\_\_\_\_\_(arr1, arr2));
10. }
11. }

Given options to fill the blanks:

1. compare

2. mismatch

3. equals

How many of the above options can fill the blank space, such that 3 is printed on to the console?

* 

**Only one option**

* 

**Only two options**

**(Correct)**

* 

**All three options**

* 

**None of the given options**

**Explanation**

**UKOCP49525:**

Arrays.equals(...) method returns boolean value and hence it is not the correct option.

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

For comparing the array contents, these methods take help of static compare(x, y) method defined in respective Wrapper classes. For example, Arrays.compare(byte[] a, byte[] b) uses Byte.compare(byte x, byte y) to compare array contents.

So, for the exam, you need to have an idea of how static compare(...) methods are implemented in these wrapper classes:

For Character, Byte & Short; compare method returns x - y.

For Integer and Long; compare method returns -1 if x < y, it returns 1 if x > y and it returns 0 if x == y.

For Float and Double; logic is almost same as Integer type but logic is bit complex for finding equality of two floats or two doubles (not part of the exam).

For Boolean; compare method returns 0 if x == y, 1 if x is true and -1 if x is false.

For the given code, if compare is used to fill the blank space, then

System.out.println(Arrays.compare(arr1, arr2)); would return 3, which is the result of Byte.compare((byte)20, (byte)17).

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

For the given code, if mismatch is used to fill the blank space, then

System.out.println(Arrays.mismatch(arr1, arr2)); would return 3, as there is a mismatch at 4th element, which is index 3.

Bottom of Form

Top of Form

Question 39: **Correct**

**Question ID: UKOCP60757**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. try {
8. System.out.print(Arrays.compare(new char[] { 'A' }, null));
9. } catch (Exception ex) {
10. System.out.print('A');
11. }
13. try {
14. System.out.print(Arrays.mismatch(null, new int[] { 1 }));
15. } catch (Exception ex) {
16. System.out.print('B');
17. }
19. try {
20. System.out.print(Arrays.equals(new short[] { 10 }, null));
21. } catch (Exception ex) {
22. System.out.print('C');
23. }
24. }
25. }

What will be the result of compiling and executing above code?

* 

**ABC**

* 

**10false**

* 

**ABfalse**

* 

**1Bfalse**

**(Correct)**

* 

**A0false**

* 

**A0C**

* 

**10C**

**Explanation**

**UKOCP60757:**

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

Please note:

If a is null and b is not null, then compare method returns -1.

If b is null and a is not null, then compare method returns 1.

If both a and b are null, then compare method returns 0.

Therefore, System.out.print(Arrays.compare(new char[] { 'A' }, null)); prints 1 on to the console.

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

Therefore, System.out.print(Arrays.mismatch(null, new int[] { 1 })); throws NullPointerException and hence corresponding catch block executes and prints B on to the console.

Arrays class has following overloaded equals methods for primitive type arrays:

boolean equals​(boolean[] a, boolean[] b)

boolean equals​(byte[] a, byte[] b)

boolean equals​(short[] a, short[] b)

boolean equals​(char[] a, char[] b)

boolean equals​(int[] a, int[] b)

boolean equals​(long[] a, long[] b)

boolean equals​(float[] a, float[] b)

boolean equals​(double[] a, double[] b)

Above methods return true if both of the arrays are null or both contain the same elements in the same order, otherwise return false.

Therefore, System.out.print(Arrays.equals(new short[] { 10 }, null)); prints false on to the console.

1Bfalse is the output of given code.

Bottom of Form

Top of Form

Question 40: **Correct**

**Question ID: UKOCP35360**

Consider below statements:

1. java.util.Arrays.compare(new int [] {10}, null);

2. java.util.Arrays.compare(null, null);

3. java.util.Arrays.mismatch(new char[] {'1'}, null);

4. java.util.Arrays.equals(null, new boolean[] {true});

How many of the above statements will compile and execute successfully (without throwing any exceptions)?

* 

**None of the above statements**

* 

**Only one statement**

* 

**Only two statements**

**(Correct)**

* 

**Only three statements**

* 

**All four statements**

**Explanation**

**UKOCP35360:**

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

Please note:

If a is null and b is not null, then compare method returns -1.

If b is null and a is not null, then compare method returns 1.

If both a and b are null, then compare method returns 0.

Hence,

1st statement, java.util.Arrays.compare(new int [] {10}, null); compiles and executes successfully and returns 1.

2nd statement, java.util.Arrays.compare(null, null); causes compilation failure as it is an ambiguous call.

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

Therefore, 3rd statement, java.util.Arrays.mismatch(new char[] {'1'}, null); compiles successfully but on execution throws NullPointerException.

Arrays class has following overloaded equals methods for primitive type arrays:

boolean equals​(boolean[] a, boolean[] b)

boolean equals​(byte[] a, byte[] b)

boolean equals​(short[] a, short[] b)

boolean equals​(char[] a, char[] b)

boolean equals​(int[] a, int[] b)

boolean equals​(long[] a, long[] b)

boolean equals​(float[] a, float[] b)

boolean equals​(double[] a, double[] b)

Above methods return true if both of the arrays are null or both contain the same elements in the same order, otherwise return false.

Therefor, 4th statement, java.util.Arrays.equals(null, new boolean[] {true}); compiles and executes successfully and returns false.

Only two statements, 1st and 4th, compile and execute successfully.

Bottom of Form

Top of Form

Question 41: **Correct**

**Question ID: UKOCP36697**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. boolean[] arr1 = { true, false, true, false };
8. boolean[] arr2 = { true, true, true, false };
10. if (!Arrays.equals(arr1, arr2)) {
11. System.out.println(Arrays.compare(arr1, arr2));
12. } else {
13. System.out.println(Arrays.mismatch(arr1, arr2));
14. }
15. }
16. }

What will be the result of compiling and executing above code?

* 

**-1**

**(Correct)**

* 

**0**

* 

**1**

* 

**true**

* 

**false**

**Explanation**

**UKOCP36697:**

Arrays class has following overloaded equals methods for primitive type arrays:

boolean equals​(boolean[] a, boolean[] b)

boolean equals​(byte[] a, byte[] b)

boolean equals​(short[] a, short[] b)

boolean equals​(char[] a, char[] b)

boolean equals​(int[] a, int[] b)

boolean equals​(long[] a, long[] b)

boolean equals​(float[] a, float[] b)

boolean equals​(double[] a, double[] b)

Above methods return true if both of the arrays are null or both contain the same elements in the same order, otherwise return false.

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

For comparing the array contents, these methods take help of static compare(x, y) method defined in respective Wrapper classes. For example, Arrays.compare(boolean[] a, boolean[] b) uses Boolean.compare(boolean x, boolean y) to compare array contents.

So, for the exam, you need to have an idea of how static compare(...) methods are implemented in these wrapper classes:

For Character, Byte & Short; compare method returns x - y.

For Integer and Long; compare method returns -1 if x < y, it returns 1 if x > y and it returns 0 if x == y.

For Float and Double; logic is almost same as Integer type but logic is bit complex for finding equality of two floats or two doubles (not part of the exam).

For Boolean; compare method returns 0 if x == y, 1 if x is true and -1 if x is false.

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

Based on above statements let's check given codes:

2nd element (at index 1) of the arrays are not same, therefore Arrays.equals(arr1, arr2) returns false.

But the boolean expression of if-block starts with !, which translates the if-block as below:

if (!Arrays.equals(arr1, arr2)) => if (!false) => if (true)

Therefore code inside if-block is executed, which is

System.out.println(Arrays.compare(arr1, arr2)); This returns -1, which is the result of Boolean.compare(false, true).

Bottom of Form

Top of Form

Question 42: **Correct**

**Question ID: UKOCP34034**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. char [] arr1 = {'A'};
8. char [] arr2 = {'A', 'A', 'A', 'A'};
9. System.out.println(Arrays.compare(arr1, arr2));
10. }
11. }

What will be the result of compiling and executing above code?

* 

**0**

* 

**1**

* 

**2**

* 

**3**

* 

**-1**

* 

**-2**

* 

**-3**

**(Correct)**

**Explanation**

**UKOCP34034:**

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

For comparing the array contents, these methods take help of static compare(x, y) method defined in respective Wrapper classes. For example, Arrays.compare(char[] a, char[] b) uses Character.compare(char x, char y) to compare array contents.

So, for the exam, you need to have an idea of how static compare(...) methods are implemented in these wrapper classes:

For Character, Byte & Short; compare method returns x - y.

For Integer and Long; compare method returns -1 if x < y, it returns 1 if x > y and it returns 0 if x == y.

For Float and Double; logic is almost same as Integer type but logic is bit complex for finding equality of two floats or two doubles (not part of the exam).

For Boolean; compare method returns 0 if x == y, 1 if x is true and -1 if x is false.

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

Consider following scenarios for lexicographic comparison of arrays:

A. If 1st array is null and 2nd array is not null, return -1.

E.g., Arrays.compare(null, new char[] {'A'}) returns -1

B. If 2nd array is null and 1st array is not null, return 1.

E.g., Arrays.compare(new boolean[] {false}, null) returns 1

C. If both the arrays are null, return 0.

E.g., Arrays.compare((int []) null, (int []) null) returns 0

D. If both the arrays are of equal lengths and contain same elements in the same order, return 0.

E.g., Arrays.compare(new char[] {'A', 'C', 'E'}, new char[] {'A', 'C', 'E'}) returns 0

E. If one array is the proper prefix of the other, then return a.length - b.length, where a refers to 1st array and b refers to 2nd array.

E.g., Arrays.compare(new char[] {'A', 'C', 'E'}, new char[] {'A'}) returns 3 - 1, which is 2. Please note that 2nd array is a proper prefix of 1st array.

F. For unequal arrays, respective compare methods of wrapper classes are invoked and it returns non-zero value (positive or negative) based on the array element comparison:

E.g.,

Arrays.compare(new char[] {'A', 'C', 'E'}, new char[] {'E', 'Y'}) returns -4, which is the result of Character.compare('A', 'E').

Arrays.compare(new int[] {5, 10, 15, 20}, new int[] {5, 100, 150, 200}) returns -1, which is the result of Integer.compare(10, 100).

Arrays.compare(new byte[] {5, 10, 100, 20}, new byte[] {5, 10, 15}) returns 85, which is the result of Byte.compare((byte)100, (byte)15).

For the given code,

char [] arr1 = {'A'};

char [] arr2 = {'A', 'A', 'A', 'A'};

arr1 is a proper prefix of arr2.

Hence, Arrays.compare(arr1, arr2) = arr1.length - arr2.length = 1 - 4 = -3

Output of the given code is: -3.

Bottom of Form

Top of Form

Question 43: **Correct**

**Question ID: UKOCP81222**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. int [] array1 = {};
8. int [] array2 = {100, 200};
9. System.out.print(Arrays.compare(array1, array2));
10. System.out.print(Arrays.mismatch(array1, array2));
11. }
12. }

What will be the result of compiling and executing above code?

* 

**-20**

**(Correct)**

* 

**-10**

* 

**-1-1**

* 

**00**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

**UKOCP81222:**

Arrays class has following overloaded compare methods for primitive type arrays:

int compare​(boolean[] a, boolean[] b)

int compare​(byte[] a, byte[] b)

int compare​(short[] a, short[] b)

int compare​(char[] a, char[] b)

int compare​(int[] a, int[] b)

int compare​(long[] a, long[] b)

int compare​(float[] a, float[] b)

int compare​(double[] a, double[] b)

For comparing the array contents, these methods take help of static compare(x, y) method defined in respective Wrapper classes. For example, Arrays.compare(char[] a, char[] b) uses Character.compare(char x, char y) to compare array contents.

So, for the exam, you need to have an idea of how static compare(...) methods are implemented in these wrapper classes:

For Character, Byte & Short; compare method returns x - y.

For Integer and Long; compare method returns -1 if x < y, it returns 1 if x > y and it returns 0 if x == y.

For Float and Double; logic is almost same as Integer type but logic is bit complex for finding equality of two floats or two doubles (not part of the exam).

For Boolean; compare method returns 0 if x == y, 1 if x is true and -1 if x is false.

All of these compare methods of Arrays class compare the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

Please note: If one array is the proper prefix of the other, then compare method returns a.length - b.length, where a refers to 1st array and b refers to 2nd array.

E.g., Arrays.compare(new char[] {'A', 'C', 'E'}, new char[] {'A'}) returns 3 - 1, which is 2. Please note that 2nd array is a proper prefix of 1st array.

array1 represents a valid int [] of length 0 and array2 represents an array of length 2.

array1 is a proper prefix of array2, therefore Arrays.compare(array1, array2) = array1.length - array2.length = 0 - 2 = -2.

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

For the given code, there is a mismatch at 1st element, which is index 0. Hence, Arrays.mismatch(array1, array2) returns 0.

Bottom of Form

Top of Form

Question 44: **Correct**

**Question ID: UKOCP74911**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. char [] arr1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'i', 'j', 'k'};
8. char [] arr2 = {'f', 'g', 'a', 'i', 'k'};
9. System.out.println(Arrays.mismatch(arr1, 5, 10, arr2, 0, 5));
10. }
11. }

What will be the result of compiling and executing above code?

* 

**7**

* 

**2**

**(Correct)**

* 

**8**

* 

**false**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

**UKOCP74911:**

Arrays class has following overloaded mismatch methods for primitive type arrays:

int mismatch​(boolean[] a, boolean[] b)

int mismatch​(byte[] a, byte[] b)

int mismatch​(short[] a, short[] b)

int mismatch​(char[] a, char[] b)

int mismatch​(int[] a, int[] b)

int mismatch​(long[] a, long[] b)

int mismatch​(float[] a, float[] b)

int mismatch​(double[] a, double[] b)

Above methods find and return the index of the first mismatch between two primitive arrays, otherwise return -1 if no mismatch is found. The index will be in the range of 0 (inclusive) up to the length (inclusive) of the smaller array. It throws NullPointerException, if either array is null.

E.g.,

Arrays.mismatch(new char[] {'A', 'B'}, new char[] {'A', 'B'}) returns -1, as there is no mismatch between 2 arrays.

Arrays.mismatch(new char[] {'A', 'B'}, new char[] {'B', 'A'}) returns 0, as there is mismatch at 0th index. Please note, index of first mismatch is returned.

Arrays.mismatch(null, new int[] {1}) throws NullPointerException, as 1st array is null.

There are other overloaded mismatch methods for primitive types with ranges:

int mismatch​(boolean[] a, int aFromIndex, int aToIndex, boolean[] b, int bFromIndex, int bToIndex)

int mismatch​(byte[] a, int aFromIndex, int aToIndex, byte[] b, int bFromIndex, int bToIndex)

int mismatch​(short[] a, int aFromIndex, int aToIndex, short[] b, int bFromIndex, int bToIndex)

int mismatch​(char[] a, int aFromIndex, int aToIndex, char[] b, int bFromIndex, int bToIndex)

int mismatch​(int[] a, int aFromIndex, int aToIndex, int[] b, int bFromIndex, int bToIndex)

int mismatch​(long[] a, int aFromIndex, int aToIndex, long[] b, int bFromIndex, int bToIndex)

int mismatch​(float[] a, int aFromIndex, int aToIndex, float[] b, int bFromIndex, int bToIndex)

int mismatch​(double[] a, int aFromIndex, int aToIndex, double[] b, int bFromIndex, int bToIndex)

These methods also compares the arrays, but over the specified range. 'From' index is inclusive but 'To' index is exclusive. Please note that returned index is relative.

Range version of mismatch methods throw below exceptions:

NullPointerException: if either array is null

IllegalArgumentException: if aFromIndex > aToIndex or if bFromIndex > bToIndex

ArrayIndexOutOfBoundsException: if aFromIndex < 0 or aToIndex > a.length or if bFromIndex < 0 or bToIndex > b.length

Given code:

char [] arr1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'i', 'j', 'k'};

char [] arr2 = {'f', 'g', 'a', 'i', 'k'};

System.out.println(Arrays.mismatch(arr1, 5, 10, arr2, 0, 5));

Above mismatch method compares 5 elements from 1st array, from index 5 to index 9, which is {'f', 'g', 'i', 'j', 'k'} with the 5 elements of 2nd array, from index 0 to index 4, which is {'f', 'g', 'a', 'i', 'k'}.  As the mismatch happens at 3rd element, which is at index 2 of sub-arrays, therefore 2 is printed on to the console.

Bottom of Form

Top of Form

Question 45: **Correct**

**Question ID: UKOCP39728**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. String [] str1 = {"A", "AA", "ABCD"};
8. String [] str2 = {"A", "AA", "ABCB"};
9. System.out.print(Arrays.mismatch(str1, str2));
10. System.out.print(Arrays.compare(str1, str2));
11. }
12. }

What will be the result of compiling and executing above code?

* 

**true2**

* 

**true-2**

* 

**22**

**(Correct)**

* 

**2-2**

* 

**-2-2**

* 

**-22**

* 

**32**

* 

**23**

* 

**33**

**Explanation**

**UKOCP39728:**

Apart from mismatch methods for primitive arrays, Arrays class also has following non-primitive mismatch methods:

int mismatch​(Object[] a, Object[] a2): Finds and returns the index of the first mismatch between two Object arrays, otherwise return -1 if no mismatch is found. Array elements are compared based on the obj.equals(Object) method.

int mismatch​(T[] a, T[] a2, Comparator<? super T> cmp): Finds and returns the index of the first mismatch between two Object arrays, otherwise return -1 if no mismatch is found. Array elements are compared based on the specified comparator.

String class overrides equals(Object) method and based on the implementation of this method, for the given code, both the arrays have different element at index 2. Therefore, Arrays.mismatch(str1, str2) returns 2.

Apart from compare methods to for primitive arrays, Arrays class also has following generic compare methods:

int compare​(T[] a, T[] b): Compares two Object arrays, with Comparable elements, lexicographically.

int compare​(T[] a, T[] b, Comparator<? super T> cmp): Compares two Object arrays using a specified comparator.

String elements are Comparable, as String class implements Comparable interface.

Please note that for the given code, as both the arrays have different element at index 2. Hence, Arrays.compare(str1, str2) returns 2, which is the result of "ABCD".compareTo("ABCB").

Output of the given code is: 22.

Bottom of Form

Top of Form

Question 46: **Correct**

**Question ID: UKOCP35368**

ASCII code of 'm' is 109 and 'M' is 77.

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. StringBuilder[] array1 = { new StringBuilder("Manners"), new StringBuilder("maketh"),
8. new StringBuilder("Man") };
9. StringBuilder[] array2 = { new StringBuilder("Manners"), new StringBuilder("maketh"),
10. new StringBuilder("man") };
11. System.out.print(Arrays.mismatch(array1, array2));
12. System.out.print(Arrays.compare(array1, array2));
13. }
14. }

What will be the result of compiling and executing above code?

* 

**true2**

* 

**true-2**

* 

**true32**

* 

**true-32**

* 

**232**

* 

**2-32**

* 

**032**

* 

**0-32**

**(Correct)**

**Explanation**

**UKOCP35368:**

Apart from mismatch methods for primitive arrays, Arrays class also has following non-primitive mismatch methods:

int mismatch​(Object[] a, Object[] a2): Finds and returns the index of the first mismatch between two Object arrays, otherwise return -1 if no mismatch is found. Array elements are compared based on the obj.equals(Object) method.

int mismatch​(T[] a, T[] a2, Comparator<? super T> cmp): Finds and returns the index of the first mismatch between two Object arrays, otherwise return -1 if no mismatch is found. Array elements are compared based on the specified comparator.

Statement Arrays.mismatch(array1, array2) returns 0 as StringBuilder class doesn't override equals(Object) method and hence the implementation of equals(Object) method Object class is invoked. equals(Object) method defined in Object class uses == operator to check the equality and based on that 1st element (at index 0) of both the arrays don't match.

Apart from compare methods to for primitive arrays, Arrays class also has following generic compare methods:

int compare​(T[] a, T[] b): Compares two Object arrays, with Comparable elements, lexicographically.

int compare​(T[] a, T[] b, Comparator<? super T> cmp): Compares two Object arrays using a specified comparator.

StringBuilder elements are Comparable, as StringBuilder class implements Comparable interface. Based on the compareTo(StringBuilder) method implementation, first two array elements of array1 and array2 are same.

As both the arrays have different StringBuilder objects at index 2 hence, Arrays.compare(array1, array2) returns -32, which is the result of "Man".compareTo("man").

It simply returns 'M' - 'm' value, which is 77 - 109 = -32.

Output of the given code is: 0-32.

Bottom of Form

Top of Form

Question 47: **Correct**

**Question ID: UKOCP71889**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.Arrays;
5. public class Test {
6. public static void main(String[] args) {
7. String [] array1 = {"OCP", "11", null};
8. String [] array2 = {"OCP", "11"};
9. String [] array3 = {null, "OCP", "11"};
11. System.out.print(Arrays.compare(array1, array2));
12. System.out.print(Arrays.compare(array2, array3));
13. System.out.print(Arrays.compare(array3, array2));
14. }
15. }

What will be the result of compiling and executing above code?

* 

**200**

* 

**201**

* 

**11-1**

**(Correct)**

* 

**1-11**

* 

**-1-1-1**

* 

**21-1**

* 

**-11-1**

* 

**-111**

**Explanation**

**UKOCP71889:**

Apart from compare methods to for primitive arrays, Arrays class also has following generic compare method:

int compare​(T[] a, T[] b): Compares two Object arrays, with Comparable elements, lexicographically.

Above method compares the arrays lexicographically and return below values:

0:  if the first and second array are equal and contain the same elements in the same order

<0: if the first array is lexicographically less than the second array

>0: if the first array is lexicographically greater than the second array

Please note:

1. A null array is less than a non-null array:

    A. If 1st array is null and 2nd array is not null, return -1.

       E.g., Arrays.compare(null, new String[] {"JAVA"}); returns -1.

    B. If 2nd array is null and 1st array is not null, return 1.

       E.g., Arrays.compare(new String[] {"JAVA"}, null); returns 1.

    C. If both the arrays are null, return 0.

       E.g., Arrays.compare((String []) null, null); returns 0.

2. While comparing, a null element is less than a non-null element:

    A. If element from 1st array is null, and corresponding element from 2nd array is not null, return -1

       E.g., Arrays.compare(new String[] {null}, new String[] {"JAVA"}); returns -1.

    B. If element from 2nd array is null, and corresponding element from 1st array is not null, return 1.

       E.g., Arrays.compare(new String[] {"JAVA"}, new String[] {null}); returns 1.

    C. If element from 1st array is null, and corresponding element from 2nd array is also null, return 0.

       Arrays.compare(new String[] {null}, new String[] {null}); returns 0.

3. If one array is the proper prefix of the other, then return a.length - b.length, where a refers to 1st array and b refers to 2nd array.

E.g., Arrays.compare(new String[] {"A", "T", "O", "M", "I", "C"}, new String[] {"A", "T", "O", "M"}) returns 6 - 4, which is 2. Please note that 2nd array is a proper prefix of 1st array.

For the given code,

String elements are Comparable, as String class implements Comparable interface.

Let's check given 3 compare statements:

Arrays.compare(array1, array2): array2 is a proper prefix of array1, hence Arrays.compare(array1, array2) = array1.length - array2.length = 3 - 2 = 1.

Arrays.compare(array2, array3): array2[0] = "OCP" and array3[0] = null. Based on above "2.B.", it returns 1.

Arrays.compare(array3, array2): array3[0] = null and array2[0] = "OCP". Based on above "2.A.", it returns -1.

Hence output of given code is: 11-1

Bottom of Form

Top of Form

Question 48: **Correct**

**Question ID: UKOCP11399**

Which of the following is not a valid array declaration?

* 

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

* 

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

* 

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

* 

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

**(Correct)**

**Explanation**

**UKOCP11399:**

`int [] arr1 = new int[8];` Creates one-dimensional array object to store 8 elements and arr1 refers to it. This statement compiles without any error.

`int [][] arr2 = new int[8][8];` Creates two-dimensional array object to store 8 \* 8 = 64 elements. This statement also compiles fine.

`int [] arr3 [] = new int[8][];` Creates two-dimensional array object, whose 1st dimension is 8 but 2nd dimension is not yet defined. On the left side array symbols can be used before the reference variable or after the reference variable or can be mixed, hence `int [][] arr3`, `int [] arr3 []` and `int arr3[][]` all are valid. This statement compiles successfully.

`int arr4[][] = new int[][8];`: 1st array dimension must be specified at the time of declaration. new int[][8]; causes compilation error as 1st dimension is not specified.

Bottom of Form

Top of Form

Question 49: **Correct**

**Question ID: UKOCP68241**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
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. }

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

* 

**ABC**

**DEF**

**GHI**

* 

**BBB**

**EEE**

**HHH**

**(Correct)**

* 

**AAA**

**DDD**

**GGG**

* 

**CCC**

**FFF**

**III**

**Explanation**

**UKOCP68241:**

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 50: **Correct**

**Question ID: UKOCP89186**

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

* 

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

* 

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

* 

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

**(Correct)**

* 

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

**Explanation**

**UKOCP89186:**

`char [] arr1 [] = new char[5][];`: Creates two-dimensional array object, whose 1st dimension is 5 but 2nd dimension is not yet defined. On the left side array symbols can be used before the reference variable or after the reference variable or can be mixed, hence `char [][] arr1`, `char [] arr1 []` and `char arr1[][]` all are valid. This statement compiles successfully.

`byte [] val = new byte[10];`: Creates one-dimensional byte array object and val refers to it. All the array elements are initialized to 0. This statement compiles without any error.

`int [] arr2 = {1, 2, 3, 4, 5};`: This syntax creates one-dimensional int array object of 5 elements and initializes these 5 elements as well. It initializes element at 0th index to 1, element at 1st index to 2, element at 2nd index to 3, element at 3rd index to 4 and element at 4th index to 5. 'arr2' refers to this one-dimensional array object. This statement also compiles fine.

`int [] arr3 = new int[3]{10, 20, 30};`: You can't specify size at the time of initializing with data, hence `new int[3]{10, 20, 30};` causes compilation error.

Correct syntax is: `int [] arr3 = new int[]{10, 20, 30};` OR `int [] arr3 = {10, 20, 30};`

Bottom of Form

Top of Form

Question 51: **Correct**

**Question ID: UKOCP50981**

Consider below code of Test.java file:

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

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

* 

**6**

* 

**64**

* 

**642**

**(Correct)**

* 

**7**

* 

**75**

* 

**753**

* 

**7531**

* 

**7654321**

**Explanation**

**UKOCP50981:**

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

switch expression can accept following:

char or Character,

byte or Byte,

short or Short,

int or Integer,

An enum only from Java 6,

A String expression only from Java 7.

In this case, switch expression [switch (arr[i][j])] is of String type.

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

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

1st iteration of outer loop, i=0.

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

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

-2nd iteration of inner loop

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

-3rd iteration of inner loop

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

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

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

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

Therefore, the output is: 642.

Bottom of Form

Top of Form

Question 52: **Correct**

**Question ID: UKOCP61619**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. var var = 3; //Line n1
6. String [][] arr = new String[--var][var++]; //Line n2
7. arr[1][1] = "X"; //Line n3
8. arr[1][2] = "Y"; //Line n4
9. for(String [] arr1 : arr) {
10. for(String s : arr1) {
11. if(s != null)
12. System.out.print(s);
13. }
14. }
15. }
16. }

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

* 

**It causes compilation error at single statement**

* 

**It causes compilation error at multiple statements**

* 

**It throws an exception at runtime**

**(Correct)**

* 

**It prints XY on to the console and program terminates successfully**

* 

**It prints XY on to the console and program terminates abruptly**

**Explanation**

**UKOCP61619:**

Local variable Type inference was added in JDK 10.

Reserved type name var is allowed in JDK 10 onwards for local variable declarations with initializers, enhanced for-loop indexes, and index variables declared in traditional for loops. For example,

var x = "Java"; //x infers to String

var m = 10; //m infers to int

The identifier var is not a keyword, hence var can still be used as variable name, method name or package name but it cannot be used as a class or interface name.

Hence, there is no issue with Line n1, variable 'var' is of int type and it stores value 3.

Line n2:

String [][] arr = new String[--var][var++]; //var = 3

Access array element operator [] is left to right associative.

=> String [][] arr = new String[2][var++]; //var = 2, var is decremented first and then used in the expression.

=> String [][] arr = new String[2][2]; //var = 3, value of var is used first and then it is incremented by 1

Hence, arr refers to 2-dimensional String array object {{null, null}, {null, null}}.

At Line n3, arr[1][1] = "X"; assigns "X" to element at index [1][1], therefore arr --> {{null, null}, {null, "X"}}

At Line n4, arr[1][2] = "Y"; causes ArrayIndexOutOfBoundsException as 2nd index 2 is out of range.

As Line n4 throws Exception at runtime, hence for loop will not be executed.

Bottom of Form

Top of Form

Question 53: **Correct**

**Question ID: UKOCP22056**

How many of the below code statements, written inside main method will compile successfully?

1. var arr1 = new int[]{10};

2. var arr2 = new String[][] {};

3. var arr3 = new char[][] {{}};

4. var arr4 = {10, 20, 30};

5. var arr5 = new String[][] {new String[]{"LOOK"}, new String[] {"UP"}};

6. var [] arr6 = new int[] {2, 3, 4};

* 

**None of the statements compile successfully**

* 

**Only 1 statement compiles successfully**

* 

**2 statements compile successfully**

* 

**3 statements compile successfully**

* 

**4 statements compile successfully**

**(Correct)**

* 

**5 statements compile successfully**

* 

**All 6 statements compile successfully**

**Explanation**

**UKOCP22056:**

`var arr1 = new int[]{10};`: Right side expression creates an int[] object of one element and 10 is assigned to that element. Target-type (int[]) is specified on the right side, hence this statement compiles successfully. arr1 infers to int[] type.

`var arr2 = new String[][] {};`: Right side expression creates a String[][] object, whose first dimension is 0 (similar to `var arr2 = new String[0][];`), so it's a valid array object. Target-type (String[][]) is specified on the right side, hence this statement compiles successfully. arr2 infers to String[][] type.

`var arr3 = new char[][] {{}};`: Right side expression creates a char[][] object, whose first dimension is 1 (similar to `var arr3 = new char[1][];`), so it's a valid array object. Target-type (char[][]) is specified on the right side, hence this statement compiles successfully. arr3 infers to char[][] type.

`var arr4 = {10, 20, 30};`: Explicit target-type is needed for the array initializer, if you use var type. Over here as explicit target-type is missing, hence this statement causes compilation error.

`var arr5 = new String[][] {new String[]{"LOOK"}, new String[] {"UP"}};`: Right side expression creates String[][] object whose first dimension is 2 and second dimension is 1 (same as `var arr5 = new String[2][1]; arr5[0][0] = "LOOK"; arr5[1][0] = "UP";`). Target-type (String[][]) is specified on the right side, hence this statement compiles successfully. arr5 infers to String[][] type.

`var [] arr6 = new int[] {2, 3, 4};`: var is not allowed as an element type of an array, hence `var [] arr6` causes compilation error.

Hence, out of 6 statements, 4 statements compile successfully.

Bottom of Form

Top of Form

Question 54: **Correct**

**Question ID: UKOCP81203**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. var arr = new int[x][y]; //Line n1
6. arr[1][4] = 100;
7. arr[6][6] = 200;
8. arr[3][6] = 300;
9. }
10. }

And below combination of x and y values:

1. x = 6, y = 6

2. x = 2, y = 5

3. x = 4, y = 7

4. x = 7, y = 7

5. x = 8, y = 8

6. x = 0, y = 0

7. x = -1, y = -1

How many of above x,y pair(s) can replace x and y at Line n1 such that Test.java file compiles successfully?

* 

**All 7 pairs**

**(Correct)**

* 

**6 pairs**

* 

**5 pairs**

* 

**4 pairs**

* 

**3 pairs**

* 

**2 pairs**

* 

**1 pair**

**Explanation**

**UKOCP81203:**

Given question expects you to solve the compilation error and not care about runtime error. For array indexes, any int values can be used, hence all the 7 pairs are allowed in this case.

If question were expecting to compile and execute the program successfully, then any combination greater than the max indexes values would have worked. For example, in the given code, as max 1st dimension value = 6 and max 2nd dimension value = 6, so any int value > 6 can be used for x and any int value > 6 can be used for y.

Out of the given seven options, only two options (x = 7, y = 7) and (x = 8, y = 8) would have worked.

Also note that the statement at Line n1 after replacing the values of x and y compiles successfully. E.g.

`var arr = new int[1][1];`: Right side expression creates an int[][] object. arr infers to int[][] type.

Bottom of Form

Top of Form

Question 55: **Correct**

**Question ID: UKOCP14883**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class T {
4. @Override
5. public String toString() {
6. return "T";
7. }
8. }
10. class Printer<T> {
11. private T t;
12. Printer(T t){
13. this.t = t;
14. }
15. @Override
16. public String toString(){
17. return t.toString();
18. }
19. }
21. public class Test {
22. public static void main(String[] args) {
23. Printer<T> obj = new Printer<>(new T());
24. System.out.println(obj);
26. }
27. }

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

* 

**T**

**(Correct)**

* 

**Compilation error in Printer<T> class**

* 

**Compilation error in Test class**

* 

**Compilation error in T class**

**Explanation**

**UKOCP14883:**

T is a valid identifier in Java, hence can be used as class name. toString() method has been correctly overridden by class T. No issues with class T.

Type parameter should not be a Java keyword and naming convention for Type parameter is to use uppercase single character. class Printer<T> correctly uses type parameter, T.

When using Generic types in the code, you need to specify the type argument. In Test class, 'Printer<T> obj' => T refers to class T and not type parameter, T.

System.out.println(obj); => Prints 'T' on to the console.

Bottom of Form

Top of Form

Question 56: **Correct**

**Question ID: UKOCP69119**

1. Given code of Test.java file:
2. package com.udayankhattry.ocp;
4. class Printer<String> {
5. private String t;
6. Printer(String t){
7. this.t = t;
8. }
9. }
11. public class Test {
12. public static void main(String[] args) {
13. Printer<Integer> obj = new Printer<>(100);
14. System.out.println(obj);
15. }
16. }

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

* 

**100**

* 

**Some text containing @ symbol**

**(Correct)**

* 

**Compilation error in Printer class**

* 

**Compilation error in Test class**

**Explanation**

**UKOCP69119:**

Type parameter should not be a Java keyword & a valid Java identifier. Naming convention for Type parameter is to use uppercase single character.

In class Printer<String>, 'String' is a valid Java identifier and hence a valid type parameter even though it doesn't follow the naming convention of uppercase single character.

Printer<Integer> obj = new Printer<>(100); => Type argument is Integer and it correctly creates an instance of Printer class passing Integer object 100.

Printer class doesn't override toString() method and hence 'System.out.println(obj);' prints some text containing @ symbol.

Bottom of Form

Top of Form

Question 57: **Correct**

**Question ID: UKOCP27708**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Printer<String> {
4. private String t;
5. Printer(String t){
6. this.t = t;
7. }
8. public String toString() {
9. return null;
10. }
11. }
13. public class Test {
14. public static void main(String[] args) {
15. Printer<Integer> obj = new Printer<>(100);
16. System.out.println(obj);
17. }
18. }

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

* 

**100**

* 

**null**

* 

**Compilation error in Printer class**

**(Correct)**

* 

**Compilation error in Test class**

**Explanation**

**UKOCP27708:**

Type parameter should not be a Java keyword & a valid Java identifier. Naming convention for Type parameter is to use uppercase single character.

In class Printer<String>, 'String' is a valid Java identifier and hence a valid type parameter even though it doesn't follow the naming convention of uppercase single character.

But within Printer<String> class, 'String' is considered as type parameter and not java.lang.String class. Return value of toString() method is java.lang.String class and not type parameter 'String'. So toString() method caused compilation error in Printer class.

To resolve the compilation error, you can use below code:

1. public java.lang.String toString() {
2. return null;
3. }

Bottom of Form

Top of Form

Question 58: **Correct**

**Question ID: UKOCP10040**

Does below code compile successfully?

1. class GenericPrinter<T> {}
2. abstract class AbstractGenericPrinter<X,Y,T> extends GenericPrinter<T>{}

* 

**Yes**

**(Correct)**

* 

**No**

**Explanation**

**UKOCP10040:**

'GenericPrinter<T>' is generic type and is defined correctly.

'AbstractGenericPrinter<X,Y,T>' is also a generic type and extends another generic type 'GenericPrinter<T>'.

NOTE: If a class extends from generic type, then it must pass type arguments to its super class. Third type parameter, 'T' in 'AbstractGenericPrinter<X,Y,T>' correctly passed type argument to super class, 'GenericPrinter<T>'.

Below codes will not compile:

abstract class AbstractGenericPrinter<X,Y> extends GenericPrinter<T>{} => No way to pass type argument to the type parameter, T of super class.

abstract class AbstractGenericPrinter extends GenericPrinter<T>{} => No way to pass type argument to the type parameter, T of super class.

But below codes will compile successfully:

abstract class AbstractGenericPrinter<X,Y> extends GenericPrinter<String>{} => Type argument, 'String' is passed to the type parameter, T of super class.

abstract class AbstractGenericPrinter extends GenericPrinter<String>{} => Type argument, 'String' is passed to the type parameter, T of super class.

Bottom of Form

Top of Form

Question 59: **Correct**

**Question ID: UKOCP36672**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class A{}
4. interface M{}
5. interface N{}
7. class B extends A {}
8. class C extends A implements M {}
9. class D extends A implements M, N {}
11. class Generic<T extends A & M & N> {}
13. public class Test {
14. public static void main(String[] args) {
15. /\*INSERT\*/
16. }
17. }

Which of the following statements, if used to replace /\*INSERT\*/, will not cause any compilation error?

* 

**Generic<A> obj = new Generic<>();**

* 

**Generic<M> obj = new Generic<>();**

* 

**Generic<N> obj = new Generic<>();**

* 

**Generic<D> obj = new Generic<>();**

**(Correct)**

* 

**All options will work**

**Explanation**

**UKOCP36672:**

T is with multiple bounds, so the type argument must be a subtype of all bounds.

Bottom of Form

Top of Form

Question 60: **Correct**

**Question ID: UKOCP33167**

Does below code compile successfully?

1. class A{}
2. interface M{}
3. interface N{}
5. class B extends A {}
6. class C extends A implements M {}
7. class D extends A implements M, N {}
9. class Generic<T extends M & N & A> {}

* 

**Yes**

* 

**No**

**(Correct)**

**Explanation**

**UKOCP33167:**

If multiple bounds are available and one of the bounds is a class, then it must be specified first.

class Generic<T extends M & N & A> {} => A is specified at last and hence compilation error.

Bottom of Form

Top of Form

Question 61: **Correct**

**Question ID: UKOCP39706**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Printer<T extends String> {}
5. public class Test {
6. public static void main(String[] args) {
7. Printer<String> printer = new Printer<>();
8. System.out.println(printer);
9. }
10. }

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

* 

**Some text containing @ symbol**

**(Correct)**

* 

**Compilation error for Printer class**

* 

**Compilation error for Test class**

**Explanation**

**UKOCP39706:**

Even though String is a final class but T extends String is a valid syntax. As no class extends from java.lang.String class so parameterized type will always be Printer<String>.

Generic class Printer doesn't override toString() method, hence Object version is invoked.

Bottom of Form

Top of Form

Question 62: **Correct**

**Question ID: UKOCP66089**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Printer<T implements Cloneable> {}
5. public class Test {
6. public static void main(String[] args) {
7. Printer<String []> printer = new Printer<>();
8. System.out.println(printer);
9. }
10. }

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

* 

**Some text containing @ symbol**

* 

**Compilation error for Printer class**

**(Correct)**

* 

**Compilation error for Test class**

**Explanation**

**UKOCP66089:**

For bounds, extends keyword is used for both class and interface.

Correct declaration of Printer class should be:

class Printer<T extends Cloneable> {}

Bottom of Form

Top of Form

Question 63: **Correct**

**Question ID: UKOCP71425**

Consider below codes:

1. package com.udayankhattry.ocp;
3. class A<T extends String> {
5. }
7. class B<T super String> {
9. }

Which of the following statement is correct?

* 

**Both class A and B compile successfully**

* 

**Only class A compiles successfully**

**(Correct)**

* 

**Only class B compiles successfully**

**Explanation**

**UKOCP71425:**

super is used with wildcard (?) only.

Bottom of Form

Top of Form

Question 64: **Correct**

**Question ID: UKOCP68264**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static <T> T get(T t) {
5. return t;
6. }
8. public static void main(String[] args) {
9. String str = get("HELLO");
10. System.out.println(str);
11. }
12. }

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

* 

**HELLO**

**(Correct)**

* 

**Compilation error in 'get' method**

* 

**Compilation error in 'main' method**

* 

**Runtime Exception**

**Explanation**

**UKOCP68264:**

Return type of generic method 'get' is T, which is correctly defined before the return type of the method.

get("HELLO"); passed String so return value should be String only. String result is stored in str variable and same is printed using System.out.println statement.

Bottom of Form

Top of Form

Question 65: **Correct**

**Question ID: UKOCP68265**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Animal {}
5. class Dog extends Animal {}
7. class Cat extends Animal {}
9. class A<T> {
10. T t;
11. void set(T t) {
12. this.t = t;
13. }
15. T get() {
16. return t;
17. }
18. }
20. public class Test {
21. public static <T> void print1(A<? extends Animal> obj) {
22. obj.set(new Dog()); //Line n1
23. System.out.println(obj.get().getClass());
24. }
26. public static <T> void print2(A<? super Dog> obj) {
27. obj.set(new Dog()); //Line n2
28. System.out.println(obj.get().getClass());
29. }
31. public static void main(String[] args) {
32. A<Dog> obj = new A<>();
33. print1(obj); //Line n3
34. print2(obj); //Line n4
35. }
36. }

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

* 

**class com.udayankhattry.ocp.Dog**

**class com.udayankhattry.ocp.Dog**

* 

**null**

**class com.udayankhattry.ocp.Dog**

* 

**class com.udayankhattry.ocp.Dog**

**null**

* 

**Compilation error**

**(Correct)**

* 

**Runtime Exception**

**Explanation**

**UKOCP68265:**

print1(A<? extends Animal> obj) => print1 method can accept arguments of A<Animal> or A<Dog> or A<Cat> types at runtime.

Suppose you have passed 'new A<Cat>()' as the argument of print1 method. Line n1 will not work in this case.

As compiler is not sure about the data that would come at runtime, hence it doesn't allow Line n1. Line n1 causes compilation failure.

print2(A<? super Dog> obj) => print2 method can accept arguments of A<Dog> or A<Animal> or A<Object> types at runtime.

All 3 arguments works with Line n2, hence no issues with Line n2.

Bottom of Form

Top of Form

Question 66: **Incorrect**

**Question ID: UKOCP13554**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. private <T extends Number> static void print(T t) {
5. System.out.println(t.intValue());
6. }
8. public static void main(String[] args) {
9. print(Double.valueOf(5.5));
10. }
11. }

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

* 

**5**

* 

**6**

* 

**Compilation error**

**(Correct)**

* 

**Runtime Exception**

**(Incorrect)**

**Explanation**

**UKOCP13554:**

A generic method is defined in non-generic class.

Type parameter for the method should be defined just before the return type of the method. In this case, '<T extends Number>' is not appearing just before void and hence compilation error.

Bottom of Form

Top of Form

Question 67: **Correct**

**Question ID: UKOCP25541**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. private static <T extends Number> void print(T t) {
5. System.out.println(t.intValue());
6. }
8. public static void main(String[] args) {
9. /\*INSERT\*/
10. }
11. }

Which of the following statements, if used to replace /\*INSERT\*/, will not cause any compilation error?

Select 2 options.

* 

**print(Integer.valueOf(1));**

**(Correct)**

* 

**print(Number.valueOf(0));**

* 

**print(new Object());**

* 

**print(Character.valueOf('a'));**

* 

**print(Double.valueOf(5.5));**

**(Correct)**

**Explanation**

**UKOCP25541:**

Let's check all the options one by one:

print(Integer.valueOf(1));

✓ Integer.valueOf(int) returns Integer instance and Integer class extends Number class.

print(Number.valueOf(0));

✗ Number is an abstract class and it doesn't have valueOf method.

print(new Object());

✗ Object class doesn't extend Number.

print(Character.valueOf('a'));

✗ Character class doesn't extend Number.

print(Double.valueOf(5.5));

✓ Double.valueOf(double) returns Double instance and Double class extends Number class.

Bottom of Form

Top of Form

Question 68: **Correct**

**Question ID: UKOCP44284**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.\*;
5. public class Test {
6. public static void main(String[] args) {
7. List<? extends String> list = new ArrayList<>(Arrays.asList("A", "E", "I", "O")); //Line n1
8. list.add("U"); //Line n2
9. list.forEach(System.out::print);
10. }
11. }

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

* 

**AEIO**

* 

**AEIOU**

* 

**Line n1 causes compilation error**

* 

**Line n2 causes compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP44284:**

Line n1 is a valid syntax but as upper-bounded wildcard is used, hence add operation is not supported. Line n2 causes compilation failure.

Bottom of Form

Top of Form

Question 69: **Correct**

**Question ID: UKOCP47794**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<? super String> list = new ArrayList<>();
9. list.add("A");
10. list.add("B");
11. for(String str : list) {
12. System.out.print(str);
13. }
14. }
15. }

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

* 

**AB**

* 

**Compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP47794:**

For 'List<? super String>' type of read objects is 'Object' and type of write objects are String and its subclasses (no subclass of String as String is final).

'for(String str : list)' causes compilation failure. Correct syntax should be: 'for(Object str : list)'

Bottom of Form

Top of Form

Question 70: **Correct**

**Question ID: UKOCP70594**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. private static final <X extends Integer, Y extends Integer> void add(X x, Y y) {
5. System.out.println(x + y);
6. }
8. public static void main(String[] args) {
9. add(10, 20);
10. }
11. }

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

* 

**Compilation error**

* 

**Runtime Exception**

* 

**30**

**(Correct)**

* 

**1020**

**Explanation**

**UKOCP70594:**

If a generic method is defined in a non-generic class then type parameters must appear before the return type of the method.

Integer is also a final class so parameters X and Y can only be of Integer type.

add(10, 20); => Auto-boxing converts int literals to Integer objects. 30 is printed on to the console.

Bottom of Form

Top of Form

Question 71: **Correct**

**Question ID: UKOCP49972**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test<T> {
4. T [] obj;
6. public Test() {
7. obj = new T[100];
8. }
10. public T [] get() {
11. return obj;
12. }
14. public static void main(String[] args) {
15. Test<String> test = new Test<>();
16. String [] arr = test.get();
17. System.out.println(arr.length);
18. }
19. }

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

* 

**100**

* 

**Compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP49972:**

Instantiation of a type parameter 'new T()' or an array of type parameter 'new T[5]' are not allowed.

'obj = new T[100];' causes compilation failure.

Bottom of Form

Top of Form

Question 72: **Correct**

**Question ID: UKOCP57597**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test<T> {
4. static T obj;
5. }

Does above code compile successfully?

* 

**Yes**

* 

**No**

**(Correct)**

**Explanation**

**UKOCP57597:**

static declaration of type parameter is not allowed, only instance declaration is possible.

'static T obj;' causes compilation failure.

Bottom of Form

Top of Form

Question 73: **Correct**

**Question ID: UKOCP44289**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
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**

* 

**false**

* 

**Compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP44289:**

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 polymorhic, 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 74: **Correct**

**Question ID: UKOCP78072**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. import java.util.ArrayList;
4. import java.util.List;
6. public class Test {
7. public static void main(String[] args) {
8. List<String> list1 = new ArrayList<>();
9. list1.add("A");
10. list1.add("B");
12. List<? extends Object> list2 = list1;
13. list2.remove("A"); //Line n1
14. list2.add("C"); //Line n2
16. System.out.println(list2);
17. }
18. }

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

* 

**ABC**

* 

**BC**

* 

**Runtime exception**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP78072:**

list1 is of List<String> type and contains 2 elements "A" and "B".

list2 is of List<? extends Object> type, which means any List whose type extends from Object. As String extends Object, hence 'List<? extends Object> list2 = list1;' works.

list2.remove("A"); => remove is non-generic method. remove(Object) will be invoked and it will successfully remove "A" from list2.

list2.add("C"); => add is a generic method. add(? extends Object) would be invoked. This means it can take an instance of any UnknownType (extending from Object class).

Compiler can never be sure whether passed argument is a subtype of UnknownType (extending from Object class). Line n2 causes compilation failure.

NOTE: Compiler works with reference types and not instances.

Simple way to remember is that as upper-bounded wildcard is used, hence add operation is not supported. Line n2 causes compilation failure.

Bottom of Form

Top of Form

Question 75: **Correct**

**Question ID: UKOCP83407**

Given code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test<T> {
4. private T t;
6. public T get() {
7. return t;
8. }
10. public void set(T t) {
11. this.t = t;
12. }
14. public static void main(String args[]) {
15. Test obj = new Test();
16. obj.set("OCP");
17. obj.set(85);
18. obj.set('%');
20. System.out.println(obj.get());
21. }
22. }

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

* 

**Compilation error**

* 

**Runtime exception**

* 

**OCP85%**

* 

**%**

**(Correct)**

* 

**Output contains some text containing @ symbol**

**Explanation**

**UKOCP83407:**

Test<T> is generic type and Test is raw type. When raw type is used then T is Object, which means set method will have signature: set(Object t).

Test obj = new Test(); => Test object is created and obj refers to it.

obj.set("OCP"); => Instance variable t refers to "OCP".

obj.set(85); => Instance variable t refers to Integer object, 85. Auto-boxing converts int literal to Integer object.

obj.set('%'); => Instance variable t refers to Character object, %. Auto-boxing converts char literal to Character object.

obj.get() => this returns Character object as as Character class overrides toString() method, % is printed on to the console.

Bottom of Form