**Controlling Program Flow, Object-Oriented Approach - 1 - Results**

Return to review

Chart

Pie chart with 4 slices.

End of interactive chart.

Attempt 1

All knowledge areas

All questions

Top of Form

Question 1: **Correct**

**Question ID: UKOCP51029**

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 num = 10;
6. if(num++ == num++) {
7. System.out.println("EQUAL " + num);
8. } else {
9. System.out.println("NOT EQUAL " + num);
10. }
11. }
12. }

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

* 

**EQUAL 12**

* 

**EQUAL 11**

* 

**NOT EQUAL 12**

**(Correct)**

* 

**NOT EQUAL 11**

**Explanation**

**UKOCP51029:**

Given boolean expression:

(num++ == num++) //num=10

(10 == num++) //Left side operand is evaluated first, value 10 is used in the expression and variable num is incremented by 1, so num=11

(10 == 11) //Right side operand is evaluated next, value 11 is used in the expression and variable num is incremented by 1, so num = 12

Above expression evaluates to false, hence else block is executed and NOT EQUAL 12 is printed on to the console.

Bottom of Form

Top of Form

Question 2: **Correct**

**Question ID: UKOCP19327**

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 grade = 85;
6. if(grade >= 60)
7. System.out.println("Congratulations");
8. System.out.println("You passed");
9. else
10. System.out.println("You failed");
12. }
13. }

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

* 

**Congratulations**

* 

**You failed**

* 

**Compilation error**

**(Correct)**

* 

**Congratulations**

**You passed**

**Explanation**

**UKOCP19327:**

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

Above code can be written as below:

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

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

Bottom of Form

Top of Form

Question 3: **Correct**

**Question ID: UKOCP33613**

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 grade = 60;
6. if(grade = 60)
7. System.out.println("You passed...");
8. else
9. System.out.println("You failed...");
10. }
11. }

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

* 

**You passed...**

* 

**You failed...**

* 

**Compilation error**

**(Correct)**

* 

**Produces no output**

**Explanation**

**UKOCP33613:**

Following are allowed in boolean expression of if statement:

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

2. A boolean variable. e.g. flag

3. A boolean literal: true or false

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

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

Bottom of Form

Top of Form

Question 4: **Correct**

**Question ID: UKOCP27356**

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

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int val = 25;
6. if(val++ < 26) {
7. System.out.println(val++);
8. }
9. }
10. }

* 

**25**

* 

**26**

**(Correct)**

* 

**27**

* 

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

**Explanation**

**UKOCP27356:**

Initially val = 25.

'if(val++ < 26)' means 'if(25 < 26)', value of val (25) is used in the boolean expression and then value of val is incremented by 1, so val = 26.

25 < 26 is true, control goes inside if-block and executes System.out.println(val++); This prints current value of val to the console, which is 26 and after that increments the value of val by 1, so val becomes 27.

Bottom of Form

Top of Form

Question 5: **Correct**

**Question ID: UKOCP31855**

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 score = 30; // Line n1
6. var grade = 'F'; // Line n2
7. if (50 <= score < 60) // Line n3
8. grade = 'D';
9. else if (60 <= score < 70) // Line n4
10. grade = 'C';
11. else if (70 <= score < 80) // Line n5
12. grade = 'B';
13. else if (score >= 80)
14. grade = 'A';
15. System.out.println(grade);
16. }
17. }

What is the result of compiling and executing Test class?

* 

**Compilation error**

**(Correct)**

* 

**A**

* 

**B**

* 

**C**

* 

**D**

* 

**F**

**Explanation**

**UKOCP31855:**

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.

At Line n1, score infers to int type.

At Line n2, grade infers to char type.

Both Line n1 and Line n2 compile successfully.

Let's check the boolean expression of Line n3:

50 <= score < 60

As multiple operators are available, so let's group the operators first on the basis of precedence and associativity.

Relational operators (<, >, <= and >=) are at same level and left to right associative, hence given expression can be grouped as:

(50 <= score) < 60

< is a binary operator with two operands: (50 <= score) on the left is of boolean type and 60 on the right is of int type. But < operator is not defined for boolean, int type and hence Line n3 causes compilation error. Line n4 and Line n5 cause compilation error for the same reason.

Bottom of Form

Top of Form

Question 6: **Correct**

**Question ID: UKOCP74585**

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 = 10; //Line n1
6. if (false)
7. System.out.println(x); //Line n2
8. System.out.println("HELLO"); //Line n3
9. }
10. }

What is the result of compiling and executing Test class?

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

* 

**Compilation error at Line n3**

* 

**HELLO**

**(Correct)**

* 

**10**

**HELLO**

**Explanation**

**UKOCP74585:**

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.

At Line n1, variable 'x' infers to int type.

Even though compiler is aware that Line n2 will never execute, but it doesn't tag it as unreachable code. Reason for this odd behavior is explained in the Java Language specification:

https://docs.oracle.com/javase/specs/jls/se11/html/jls-14.html#jls-14.21

Following statement results in a compile-time error:

while (false) { x=3; }

because the statement x=3; is not reachable; but the superficially similar case:

if (false) { x=3; }

does not result in a compile-time error. An optimizing compiler may realize that the statement x=3; will never be executed and may choose to omit the code for that statement from the generated class file, but the statement x=3; is not regarded as "unreachable" in the technical sense specified here.

The rationale for this differing treatment is to allow programmers to define "flag" variables such as:

static final boolean DEBUG = false;

and then write code such as:

if (DEBUG) { x=3; }

The idea is that it should be possible to change the value of DEBUG from false to true or from true to false and then compile the code correctly with no other changes to the program text.

Line n2 is not executed but Line n3 executes successfully and prints HELLO on to the console.

Bottom of Form

Top of Form

Question 7: **Correct**

**Question ID: UKOCP30528**

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

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int x = 2;
6. switch (x) {
7. default:
8. System.out.println("Still no idea what x is");
9. case 1:
10. System.out.println("x is equal to 1");
11. break;
12. case 2:
13. System.out.println("x is equal to 2");
14. break;
15. case 3:
16. System.out.println("x is equal to 3");
17. break;
18. }
19. }
20. }

* 

**x is equal to 2**

**(Correct)**

* 

**Compilation error**

* 

**Still no idea what x is**

**x is equal to 1**

* 

**Produces no output**

**Explanation**

**UKOCP30528:**

Even though default block is available at the top but matching case is present. So control goes inside matching case and prints "x is equal to 2" on to the console. After that break; statement takes the control out of the switch-case block.

Bottom of Form

Top of Form

Question 8: **Correct**

**Question ID: UKOCP50130**

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

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int a = 5;
6. int x = 10;
7. switch(x) {
8. case 10:
9. a \*= 2;
10. case 20:
11. a \*= 3;
12. case 30:
13. a \*= 4;
14. }
15. System.out.println(a);
16. }
17. }

* 

**5**

* 

**10**

* 

**30**

* 

**120**

**(Correct)**

**Explanation**

**UKOCP50130:**

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

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

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

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

Bottom of Form

Top of Form

Question 9: **Correct**

**Question ID: UKOCP52310**

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

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int score = 60;
6. switch (score) {
7. default:
8. System.out.println("Not a valid score");
9. case score < 70:
10. System.out.println("Failed");
11. break;
12. case score >= 70:
13. System.out.println("Passed");
14. break;
15. }
16. }
17. }

* 

**Compilation error**

**(Correct)**

* 

**Failed**

* 

**Not a valid score**

**Failed**

* 

**Passed**

**Explanation**

**UKOCP52310:**

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

switch expression can accept following:

char or Character,

byte or Byte,

short or Short,

int or Integer,

An enum only from Java 6,

A String expression only from Java 7.

In this case, switch expression [switch (score)] is of int type.

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

Bottom of Form

Top of Form

Question 10: **Correct**

**Question ID: UKOCP29530**

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

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

* 

**var is 100**

* 

**var is 200**

* 

**In default**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP29530:**

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

switch expression can accept following:

char or Character

byte or Byte

short or Short

int or Integer

An enum only from Java 6

A String expression only from Java 7

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

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

Bottom of Form

Top of Form

Question 11: **Incorrect**

**Question ID: UKOCP20722**

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

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

* 

**MANGO**

* 

**ANY FRUIT WILL DO**

**(Correct)**

* 

**MANGO**

**BANANA**

**(Incorrect)**

* 

**MANGO**

**ANY FRUIT WILL DO**

* 

**MANGO**

**BANANA**

**ANY FRUIT WILL DO**

**Explanation**

**UKOCP20722:**

"mango" is different from "Mango", so there is no matching case available. default block is executed and as it is the last block inside switch hence after printing "ANY FRUIT WILL DO" control goes out of switch block, main method ends and program terminates successfully.

Bottom of Form

Top of Form

Question 12: **Incorrect**

**Question ID: UKOCP26028**

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

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. String sport = "swimming";
6. switch (sport) {
7. default:
8. System.out.println("RUNNING");
9. case "Tennis":
10. System.out.println("TENNIS");
11. case "Swimming":
12. System.out.println("SWIMMING");
13. case "Football":
14. System.out.println("FOOTBALL");
15. break;
16. }
17. }
18. }

* 

**RUNNING**

* 

**SWIMMING**

* 

**SWIMMING**

**FOOTBALL**

**(Incorrect)**

* 

**RUNNING**

**TENNIS**

**SWIMMING**

**FOOTBALL**

**(Correct)**

**Explanation**

**UKOCP26028:**

"swimming" is different from "Swimming", so there is no matching case available. default block is executed, "RUNNING" is printed on to the console. No break statement inside default, hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends. So in this case, it prints TENNIS, SWIMMING, FOOTBALL one after another and break; statement takes control out of switch block. main method ends and program terminates successfully.

Bottom of Form

Top of Form

Question 13: **Correct**

**Question ID: UKOCP66959**

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

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. String furniture = new String(new char[] {'S', 'o', 'f', 'a'});
6. switch (furniture) {
7. default:
8. System.out.println("CHAIR");
9. case "Recliner":
10. System.out.println("RECLINER");
11. case "Sofa":
12. System.out.println("SOFA");
13. case "Bed":
14. System.out.println("BED");
15. break;
16. }
17. }
18. }

* 

**CHAIR**

* 

**SOFA**

* 

**SOFA**

**BED**

**(Correct)**

* 

**CHAIR**

**RECLINER**

**SOFA**

**BED**

**Explanation**

**UKOCP66959:**

'furniture' refers to String object "Sofa". Matching case is available, SOFA is printed on to the console. No break statement inside 'case "Sofa":', hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends. So in this case, it prints BED and after that break; statement takes control out of switch block. main method ends and program terminates successfully.

Bottom of Form

Top of Form

Question 14: **Correct**

**Question ID: UKOCP37160**

Given code:

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

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

Select 4 options.

* 

**byte var = 10;**

**(Correct)**

* 

**long var = 10;**

* 

**Short var = 10;**

**(Correct)**

* 

**Integer var = 10;**

**(Correct)**

* 

**char var = 10;**

**(Correct)**

* 

**double var = 10;**

**Explanation**

**UKOCP37160:**

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums. In this case long and double are invalid values to be passed in switch expression. char uses 16 bits (2 Bytes) and its range is 0 to 65535 (no signed bit reserved) so it can easily store value 10.

Please note that 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.

Bottom of Form

Top of Form

Question 15: **Incorrect**

**Question ID: UKOCP71071**

Given code:

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

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

Select 3 options.

* 

**char var = '7';**

**(Incorrect)**

* 

**char var = 7;**

**(Correct)**

* 

**Integer var = 7;**

**(Correct)**

* 

**Character var = '7';**

* 

**Character var = 7;**

**(Correct)**

**Explanation**

**UKOCP71071:**

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums. In this case, all are valid values but only 3 executes "case 7:".

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

Please also note that 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.

Bottom of Form

Top of Form

Question 16: **Correct**

**Question ID: UKOCP38015**

Given:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. /\*INSERT\*/
6. switch(day) {
7. case '3':
8. System.out.println("BUY 2 GET 1 FREE");
9. break;
10. default:
11. System.out.println("SORRY!!! NO SALE");
12. }
13. }
14. }

For the class Test, which of the following options, if used to replace /\*INSERT\*/, will print "BUY 2 GET 1 FREE" on to the console?

* 

**int day = 3;**

* 

**Integer day = 3;**

* 

**int day = '3';**

**(Correct)**

* 

**None of the other options**

**Explanation**

**UKOCP38015:**

Even though char is compatible with int type but '3' is not equal to 3.

int day = 3; => SORRY!!! NO SALE

Integer day = 3; => day is of Integer type and case contains char '3'. char '3' cannot be compared with Integer and hence compilation error. case '3' can easily be compared with int value but not with Integer type.

int day = '3'; => BUY 2 GET 1 FREE

HINT: There is no need to remember. case '3' value means you are trying to equate or compare day (Integer value) with '3' (char). If assignment operation works then method invocation, switch expression parameter etc. will also work. Integer day = 3; is possible but Integer day = '3'; causes compilation error as char cannot be converted to Integer.

Bottom of Form

Top of Form

Question 17: **Incorrect**

**Question ID: UKOCP63457**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. switch ("HELLO") {
6. case "HELLO":
7. System.out.print(1);
8. default:
9. System.out.print(2);
10. case "null":
11. System.out.print(3);
12. }
13. }
14. }

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

* 

**Compilation error**

**(Incorrect)**

* 

**1**

* 

**2**

* 

**12**

* 

**23**

* 

**123**

**(Correct)**

**Explanation**

**UKOCP63457:**

case values must evaluate to the same/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.

Compatible literal value or constant can be used as the switch expression.

In this case, switch expression "HELLO" is of String type.

All the 3 case values are of String type, so no issues in the given code.

1st case is the matching case, 1 is printed on to the console. No break statement inside 'case "HELLO":', hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends.

So in this case, it prints 2 and after that 3 and finally control goes out of the switch block.

main method ends and program terminates successfully after printing 123 on to the console.

Bottom of Form

Top of Form

Question 18: **Correct**

**Question ID: UKOCP20713**

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

What is the result of compiling and executing Test class?

* 

**ONE**

**TWO**

**THREE**

* 

**TWO**

**THREE**

* 

**THREE**

* 

**None of the other options**

**(Correct)**

**Explanation**

**UKOCP20713:**

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums. switch(b) causes compilation failure as b is of Boolean type.

Bottom of Form

Top of Form

Question 19: **Correct**

**Question ID: UKOCP64304**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. final boolean flag = false; //Line n1
6. while(flag) {
7. System.out.println("Good Morning!"); //Line n2
8. }
9. }
10. }

Which of the following statements is correct for above code?

* 

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

* 

**Compilation error**

**(Correct)**

* 

**Infinite loop**

* 

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

**Explanation**

**UKOCP64304:**

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

Compiler knows the value of flag, which is false at compile time and hence it gives "Unreachable Code" error for Line n2.

Bottom of Form

Top of Form

Question 20: **Incorrect**

**Question ID: UKOCP83067**

Consider below code of Test.java file:

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

Which of the following statements is correct for above code?

* 

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

**(Correct)**

* 

**Compilation error**

**(Incorrect)**

* 

**Infinite loop**

* 

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

**Explanation**

**UKOCP83067:**

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

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

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

Bottom of Form

Top of Form

Question 21: **Correct**

**Question ID: UKOCP42992**

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

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

* 

**Compilation error**

* 

**5**

**6**

**7**

**8**

**9**

* 

**It will go in an infinite loop**

**(Correct)**

* 

**Produces no output**

**Explanation**

**UKOCP42992:**

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

Above syntax can be written as follows:

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

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

Bottom of Form

Top of Form

Question 22: **Incorrect**

**Question ID: UKOCP83060**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. do {
6. System.out.print(100);
7. } while (true); //Line n1
9. System.out.println(200); //Line n2
10. }
11. }

Which of the following statements is correct for above code?

* 

**Compiles successfully and on execution prints 200**

* 

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

**(Incorrect)**

* 

**Unreachable code compilation error**

**(Correct)**

* 

**Compiles successfully and on execution prints 100200**

**Explanation**

**UKOCP83060:**

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

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

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

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

1. public class Test {
2. public static void main(String[] args) {
3. boolean flag = true;
4. do {
5. System.out.println(100);
6. } while (flag);
8. System.out.println(200);
9. }
10. }

Above program prints 100 in infinite loop and 200 never gets printed.

Bottom of Form

Top of Form

Question 23: **Correct**

**Question ID: UKOCP49133**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. do {
6. System.out.print("SLOW-");
7. } while (false);
9. System.out.println("DOWN");
10. }
11. }

Which of the following statements is correct for above code?

* 

**Compiles successfully and on execution prints DOWN**

* 

**Compiles successfully and on execution prints SLOW- in infinite loop**

* 

**Unreachable code compilation error**

* 

**Compiles successfully and on execution prints SLOW-DOWN**

**(Correct)**

**Explanation**

**UKOCP49133:**

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

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

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

Bottom of Form

Top of Form

Question 24: **Incorrect**

**Question ID: UKOCP28697**

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 start = 1;
6. int sum = 0;
7. do {
8. if(start % 2 == 0) {
9. continue;
10. }
11. sum += start; //Line n1
12. } while(++start <= 10);
13. System.out.println(sum);
14. }
15. }

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

* 

**25**

**(Correct)**

* 

**55**

* 

**Compilation error**

* 

**24**

**(Incorrect)**

**Explanation**

**UKOCP28697:**

When start is divisible by 2 [2, 4, 6, 8, 10], continue; statement takes the control to boolean expression and hence Line n1 is not executed.

Line n1 is executed only in the case of odd numbers which are less than or equals to 10. Therefore, result is the sum of numbers 1,3,5,7,9; which is 25.

Bottom of Form

Top of Form

Question 25: **Correct**

**Question ID: UKOCP76772**

Consider below code of Test.java file:

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

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

* 

**0**

* 

**2**

* 

**3**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

**(Correct)**

**Explanation**

**UKOCP76772:**

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.

At Line n1, initialization expression of for loop is: `var var = 0;`. Java compiler automatically infers 'var' to be of int type as right side expression is int literal.

Variable 'var' is declared inside for loop, hence it is not accessible beyond loop's body. Line n2 causes compilation error.

For more information on local variable type inference, please check the URL: http://openjdk.java.net/jeps/286

Bottom of Form

Top of Form

Question 26: **Incorrect**

**Question ID: UKOCP52302**

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 i = 4; //Line n1
6. for(i = 0; i <= 2; i++){} //Line n2
7. System.out.println(i); //Line n3
8. }
9. }

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

* 

**0**

* 

**2**

* 

**3**

**(Correct)**

* 

**4**

**(Incorrect)**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

**Explanation**

**UKOCP52302:**

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

At Line n1, 'i' infers to int type, there is no issue at Line n1.

At Line n2, variable 'i' is reinitialized to 0 and this loop executes 3 times, for i = 0, i = 1 and i = 2. For i = 3, control goes out of the for loop.

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

Line n3 prints 3 to the console.

Bottom of Form

Top of Form

Question 27: **Correct**

**Question ID: UKOCP71086**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. for:
6. for (int i = 2; i <= 100; i = i + 2) {
7. for(int j = 1; j <= 10; j++) {
8. System.out.print(i \* j + "\t");
9. }
10. System.out.println();
11. if(i == 10) {
12. break for;
13. }
14. }
15. }
16. }

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

* 

**Total 5 non-blank rows will be there in the output**

* 

**Total 6 non-blank rows will be there in the output**

* 

**Total 50 non-blank rows will be there in the output**

* 

**Total 51 non-blank rows will be there in the output**

* 

**Total 100 non-blank rows will be there in the output**

* 

**Total 101 non-blank rows will be there in the output**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP71086:**

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

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

Bottom of Form

Top of Form

Question 28: **Correct**

**Question ID: UKOCP25185**

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

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

* 

**Prints 5 in an infinite loop**

* 

**Prints 5 once**

**(Correct)**

* 

**Compilation error**

* 

**5**

**3**

**2**

**1**

**Explanation**

**UKOCP25185:**

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

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

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

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

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

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

main method ends and program terminates successfully.

So, 5 gets printed only once.

Bottom of Form

Top of Form

Question 29: **Correct**

**Question ID: UKOCP43823**

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 [][] countries = {{"SINGAPORE", "AUSTRALIA"}, {"CHINA", "RUSSIA"}};
6. /\*INSERT\*/
7. }
8. }

For the class Test, which options, if used to replace /\*INSERT\*/, will print "SINGAPORE AUSTRALIA CHINA RUSSIA " on to the console?

Choose 2 options.

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

**(Correct)**

* 
  1. for(int i = 1; i <= countries.length; i++)
  2. for(int j = 1; j <= countries[i].length ; j++)
  3. System.out.print(countries[i][j] + " ");
* 
  1. for(int i = 1; i < countries.length; i++)
  2. for(int j = 1; j < countries[i].length ; j++)
  3. System.out.print(countries[i][j] + " ");
* 
  1. for(String [] arr : countries)
  2. for(String country : arr)
  3. System.out.print(country + " ");

**(Correct)**

**Explanation**

**UKOCP43823:**

Easy question on iterating through 2-dimensional array. Starting index should be 0 and not 1. As for loops contain one statement, hence curly brackets can be ignored.

for-each loop's syntax is correct.

Bottom of Form

Top of Form

Question 30: **Correct**

**Question ID: UKOCP39335**

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 = {2, 1, 0};
6. for(int i : arr) {
7. System.out.println(arr[i]);
8. }
9. }
10. }

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

* 

**2**

**1**

**0**

* 

**0**

**1**

**2**

**(Correct)**

* 

**Compilation error**

* 

**An Exception is thrown at runtime**

**Explanation**

**UKOCP39335:**

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

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

Bottom of Form

Top of Form

Question 31: **Incorrect**

**Question ID: UKOCP86564**

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

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

* 

**3**

**2**

**1**

**(Incorrect)**

* 

**1**

**2**

**3**

* 

**Compilation error**

* 

**An Exception is thrown at runtime**

**(Correct)**

**Explanation**

**UKOCP86564:**

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

When loop executes 1st time, i stores the first array element, which is 3 but System.out.println statement prints arr[3] and this causes java runtime to throw the instance of ArrayIndexOutOfBoundsException.

Bottom of Form

Top of Form

Question 32: **Correct**

**Question ID: UKOCP19749**

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 = 0; //Line n1
6. var: for (; var < 3; var++) { //Line n2
7. if (var % 2 == 0) {
8. continue var; //Line n3
9. }
10. var++; //Line n4
11. }
12. System.out.println(var);
13. }
14. }

Which of the following statements is true?

* 

**Line n1 causes compilation error**

* 

**Line n2 causes compilation error**

* 

**Line n3 causes compilation error**

* 

**Line n4 causes compilation error**

* 

**Code compiles successfully**

**(Correct)**

**Explanation**

**UKOCP19749:**

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, package name or loop's label but it cannot be used as a class or interface name.

At Line n1, variable 'var' infers to int type. Line n1 compiles successfully.

At Line n2, 'var' is used as for loop's label, no issues at Line n2 as well.

At Line n3, continue is used with label. break and continue can be used with labels, so no issue at line n3 as well.

As var infers to int type, hence at Line n4, `var++;` is a valid statement.

Given code compiles successfully.

Bottom of Form

Top of Form

Question 33: **Incorrect**

**Question ID: UKOCP64314**

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: while (true) { //Line n1
6. i: for (int i = 0; i <= 2; i++) {
7. if(i == 2) {
8. /\*INSERT\*/
9. }
10. }
11. }
12. System.out.println("THINK DIFFERENT"); //Line n2
13. }
14. }

Which of the following options, if used to replace /\*INSERT\*/, will compile successfully and on execution will print THINK DIFFERENT on to the console?

* 

**continue;**

* 

**continue i;**

* 

**continue var;**

* 

**break;**

**(Incorrect)**

* 

**break i;**

* 

**break var;**

**(Correct)**

**Explanation**

**UKOCP64314:**

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, package name or loop's label but it cannot be used as a class or interface name.

At Line n1, 'var' is used as while loop's label, so no issues at Line n1.

while(true) is and infinite loop and compiler is aware of that, so unless and until you provide the logic to break out of the loop, compiler considers Line n2 as unreachable code, which causes compilation failure.

continue statements don't break out of the loop so even if you use `continue;`, `continue i;` or `continue var;` to replace /\*INSERT\*/, unreachable code error will still be there.

`break;` and `break i;` statements are same and will break out to the inner for loop, Line n2 will still be unreachable.

`break var;` will successfully break out of the outer while loop.

On execution, control will come out of the while loop, when i equals to 2 and THINK DIFFERENT will be printed on to the console.

Bottom of Form

Top of Form

Question 34: **Incorrect**

**Question ID: UKOCP22901**

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 sum = 0;
6. for(var i = 0; i < 7; i++) { //Line n1
7. if(i == 4)
8. break;
9. else
10. continue;
11. sum += i; //Line n2
12. }
13. System.out.println(sum); //Line n3
14. }
15. }

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

* 

**10**

* 

**21**

* 

**17**

* 

**6**

* 

**4**

* 

**0**

**(Incorrect)**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

**(Correct)**

* 

**Compilation error at Line n3**

**Explanation**

**UKOCP22901:**

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, package name or loop's label but it cannot be used as a class or interface name.

At Line n1, variable 'i' infers to int type, so no issues at Line n1.

if-else block uses break; and continue; statements. break; will exit the loop and will take the control to Line n3 on the other hand continue; will take the control to Line n1. In both the cases Line n2 will never be executed.

As Compiler knows about it, hence it tags Line n2 as unreachable, which causes compilation error.

Bottom of Form

Top of Form

Question 35: **Incorrect**

**Question ID: UKOCP63462**

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 elements = 0;
6. Object [] arr = {"A", "E", "I", new Object(), "O", "U"}; //Line n1
7. for(var obj : arr) { //Line n2
8. if(obj instanceof String) {
9. continue;
10. } else {
11. break;
12. }
13. elements++; //Line n3
14. }
15. System.out.println(elements); //Line n4
16. }
17. }

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

* 

**0**

**(Incorrect)**

* 

**1**

* 

**3**

* 

**5**

* 

**6**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

* 

**Compilation error at Line n3**

**(Correct)**

* 

**Compilation error at Line n4**

**Explanation**

**UKOCP63462:**

Line n1 correctly declares an Object array. Line n1 doesn't cause any compilation error.

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, package name or loop's label but it cannot be used as a class or interface name.

At Line n2, variable 'obj' infers to Object type, so no issues at Line n2.

if-else block uses break; and continue; statements. break; will exit the loop and will take the control to Line n4 on the other hand continue; will take the control to Line n2. In both the cases Line n3 will never be executed.

As Compiler knows about it, hence it tags Line n3 as unreachable, which causes compilation error.

Bottom of Form

Top of Form

Question 36: **Incorrect**

**Question ID: UKOCP88746**

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 = { {"%", "$$"}, {"\*\*\*", "@@@@", "#####"}};
6. for(String [] str : arr) {
7. for(String s : str) {
8. System.out.println(s);
9. if(s.length() == 4) //Line n1
10. break; //Line n2
11. }
12. break; //Line n3
13. }
14. }
15. }

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

* 

**%**

* 

**%**

**$$**

**(Correct)**

* 

**%**

**$$**

**\*\*\***

**(Incorrect)**

* 

**%**

**$$**

**\*\*\***

**@@@@**

* 

**%**

**$$**

**\*\*\***

**@@@@**

**#####**

**Explanation**

**UKOCP88746:**

Variable 'arr' refers to a two-dimensional array. for-each loops are used to iterate the given array.

In 1st iteration of outer loop, str refers to one-dimensional String array {"%", "$$"}.

In 1st iteration of inner loop, s refers to "%" and "%" will be printed on to the console. Boolean expression of Line n1 evaluates to false so Line n2 is not executed.

In 2nd iteration of inner loop, s refers to "$$" and "$$" will be printed on to the console. Boolean expression of Line n1 evaluates to false so Line n2 is not executed.

Iteration of inner for-each loop is over and control executes Line n3. break; statement at Line n3 terminates the outer loop and program ends successfully.

So, output is:

%

$$

Bottom of Form

Top of Form

Question 37: **Incorrect**

**Question ID: UKOCP14052**

Consider below code of the Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. int ctr = 100;
6. one: for (var i = 0; i < 10; i++) {
7. two: for (var j = 0; j < 7; j++) {
8. three: while (true) {
9. ctr++;
10. if (i > j) {
11. break one;
12. } else if (i == j) {
13. break two;
14. } else {
15. break three;
16. }
17. }
18. }
19. }
20. System.out.println(ctr);
21. }
22. }

What is the result?

* 

**Compilation error**

**(Incorrect)**

* 

**100**

* 

**101**

* 

**102**

**(Correct)**

* 

**103**

* 

**104**

* 

**105**

* 

**106**

**Explanation**

**UKOCP14052:**

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 the variable name, method name, package name, or loop's label but it cannot be used as a class or interface name.

For the 1st loop variable, 'i' infers to int type, so no issues for 1st loop, and for the 2nd loop variable 'j' infers to int type, so no issues for 2nd loop as well.

Let's check the iteration:

1st iteration of loop one: i = 0

    1st iteration of loop two: j = 0

        1st iteration of loop three: ctr = 101. As `i == j` evaluates to true, hence `break two;` gets executed, which takes the control out of loop two and hence to the increment expression (i++) of loop one.

2nd iteration of loop one; i = 1

    1st iteration of loop two: j = 0

        1st iteration of loop three; ctr = 102. As `i > j` evaluates to true, hence `break one;` gets executed, which takes the control out of the loop one.

`System.out.println(ctr);` prints 102 onto the console.

Bottom of Form

Top of Form

Question 38: **Incorrect**

**Question ID: UKOCP22906**

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 i = 1;
6. var j = 5;
7. var k = 0;
8. A: while(true) {
9. i++;
10. B: while(true) {
11. j--;
12. C: while(true) {
13. k += i + j;
14. if(i == j)
15. break A;
16. else if (i > j)
17. continue A;
18. else
19. continue B;
20. }
21. }
22. }
23. System.out.println(k);
24. }
25. }

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

* 

**Compilation error**

**(Incorrect)**

* 

**Program never terminates as above code causes infinite loop**

* 

**6**

* 

**11**

* 

**15**

**(Correct)**

* 

**None of the other options**

**Explanation**

**UKOCP22906:**

No syntax error in the given code.

Initially, i = 1, j = 5 and k = 0.

1st iteration of A: i = 2.

    1st iteration of B: j = 4.

        1st iteration of C: k = k + i + j = 0 + 2 + 4 = 6. `i == j` evaluates to false and `i > j` also evaluates to false, hence else block gets executed. `continue B` takes the control to the loop B.

    2nd iteration of B: j = 3.

        1st iteration of C: k = k + i + j = 6 + 2 + 3 = 11. `i == j` evaluates to false and `i > j` also evaluates to false, hence else block gets executed. `continue B` takes the control to the loop B.

    3rd iteration of B: j = 2.

        1st iteration of C: k = k + i + j = 11 + 2 + 2 = 15. `i == j` evaluates to true, control breaks out of the loop A.

`System.out.println(k);` prints 15 on to the console.

Bottom of Form

Top of Form

Question 39: **Incorrect**

**Question ID: UKOCP64311**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. outer: for(var i = 0; i < 3; System.out.print(i)) {
6. i++;
7. inner: for(var j = 0; j < 3; System.out.print(j)) {
8. if(i > ++j) {
9. break outer;
10. }
11. }
12. }
13. }
14. }

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

* 

**Compilation error**

* 

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

* 

**Program terminates successfully after printing 1 on to the console**

* 

**Program terminates successfully after printing 12 on to the console**

**(Incorrect)**

* 

**Program terminates successfully after printing 123 on to the console**

* 

**Program terminates successfully after printing 1231 on to the console**

**(Correct)**

* 

**Program terminates successfully after printing 121 on to the console**

* 

**Program terminates successfully after printing 0120 on to the console**

**Explanation**

**UKOCP64311:**

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, package name or loop's label but it cannot be used as a class or interface name.

For the 1st loop variable 'i' infers to int type, so no issues for 1st loop and for the 2nd loop variable 'j' infers to int type, so no issues for 2nd loop as well.

Basic/Regular for loop has following form:

for ( [ForInit] ; [Expression] ; [ForUpdate] ) {...}

[ForInit] can be local variable initialization or the following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for both the loops, `System.out.print(...)` is used as [ForUpdate] expression, which is a MethodInvocation expression and hence a valid statement.

Given code compiles successfully.

Let's check the iterations:

1st iteration of outer: i = 0. i < 3 evaluates to true.

    i = 1.

    1st iteration of inner: j = 0. j < 3 evaluates to true as j = 0. Boolean expression `i > ++j` = `1 > 1` evaluates to false. j = 1.

    2nd iteration of inner: `System.out.print(j)` prints 1 to the console. j < 3 evaluates to true as j = 1. Boolean expression `i > ++j` = `1 > 2` evaluates to false. j = 2.

    3rd iteration of inner: `System.out.print(j)` prints 2 to the console. j < 3 evaluates to true as j = 2. Boolean expression `i > ++j` = `1 > 3` evaluates to false. j = 3.

    4th iteration of inner: `System.out.print(j)` prints 3 to the console. j < 3 evaluates to false as j = 3. Control goes out of inner for loop and to the [ForUpdate] expression of outer loop.

2nd iteration of outer: `System.out.print(i)` prints 1 to the console. i < 3 evaluates to true as i = 1.

    i = 2.

    1st iteration of inner: j = 0. j < 3 evaluates to true as j = 0. Boolean expression `i > ++j` = `2 > 1` evaluates to true. j = 1. ` break outer;` takes the control out of the outer for loop.

Program terminates successfully after printing 1231 on to the console.

Bottom of Form

Top of Form

Question 40: **Correct**

**Question ID: UKOCP64795**

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 i = 0;
6. for(System.out.print(i++); i < 2; System.out.print(i++)) {
7. System.out.print(i);
8. }
9. }
10. }

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

* 

**112**

* 

**012**

* 

**011**

**(Correct)**

* 

**12**

* 

**01**

* 

**Compilation error**

**Explanation**

**UKOCP64795:**

Basic/Regular for loop has following form:

for ( [ForInit] ; [Expression] ; [ForUpdate] ) {...}

[ForInit] can be local variable initialization or the following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for [ForInit] and [ForUpdate], `System.out.print(i++);` is used, which is a method invocation statement and hence a valid statement. Given code compiles fine.

Let's check the iterations:

1st iteration: [ForInit] expression is executed, 0 is printed on to the console. i = 1. i < 2 evaluates to true, control goes inside the loop's body and execute `System.out.print(i);` statement. 1 is printed on to the console.

2nd iteration: [ForUpdate] expression is executed, 1 is printed on to the console. i = 2. 2 < 2 evaluates to false, control goes out of the for loop. main method ends and program terminates successfully after printing 011 on to the console.

Bottom of Form

Top of Form

Question 41: **Incorrect**

**Question ID: UKOCP25180**

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 i = 0;
6. String res = null;
7. for(String [] s = {"A", "B", "C", "D"};;res = String.join(".", s)) { //Line n1
8. if(i++ == 0)
9. continue;
10. else
11. break;
12. }
13. System.out.println(res); //Line n2
14. }
15. }

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

* 

**Compilation error at Line n1**

**(Incorrect)**

* 

**Compilation error at Line n2**

* 

**A.B.C.D**

**(Correct)**

* 

**A.B**

* 

**A.**

* 

**A**

* 

**null**

**Explanation**

**UKOCP25180:**

Basic/Regular for loop has following form:

for ( [ForInit] ; [Expression] ; [ForUpdate] ) {...}

[ForInit] can be local variable initialization or the following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

In the given code, for [ForInit], `String [] s = {"A", "B", "C", "D"};` is used, which is a local variable initialization statement and hence a valid statement. For [ForUpdate], `res = String.join(".", s)` is used, which is a assignment statement and hence valid as well. Line n1 compiles fine.

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 compiler is aware that given infinite loop has an exit point, hence it doesn't mark Line n2 as unreachable. Line n2 doesn't cause any compilation error as well.

Let's check the iterations:

Initially, i = 0 and res = null.

1st iteration: s refers to a String array of 4 elements: {"A", "B", "C", "D"}. Boolean expression is missing, hence it is considered as true. Control enters loop's body.

`i++ == 0` evaluates to true and i is incremented by 1, i = 1. continue; statement takes the control to [ForUpdate] expression.

2nd iteration: res = String.join(".", s) is evaluated as res = "A.B.C.D". Boolean expression is missing, hence it is considered as true. Control enters loop's body.

`i++ == 0` evaluates to true and i is incremented by 1, i = 2. break; statement takes the control out of the for loop.

Line n2 is executed and prints A.B.C.D on to the console.

Bottom of Form

Top of Form

Question 42: **Correct**

**Question ID: UKOCP18892**

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 res = ""; //Line n1
6. String [] arr = {"1", "2", "3"};
7. for(var s : arr) { //Line n2
8. res += String.join(".", s); //Line n3
9. }
10. System.out.println(res); //Line n4
11. }
12. }

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

* 

**1.2.3**

* 

**123**

**(Correct)**

* 

**3**

* 

**Compilation error**

* 

**An exception is thrown at runtime**

**Explanation**

**UKOCP18892:**

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.

Given code compiles successfully.

At Line n1, variable 'res' infers to String and at line n2, variable 's' infers to String 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.

Let's check the iterations:

1st iteration: s refers to "1". `String.join(".", s)` returns "1" and res = "" + "1" = "1".

2nd iteration: s refers to "2". `String.join(".", s)` returns "2" and res = "1" + "2" = "12".

3rd iteration: s refers to "3". `String.join(".", s)` returns "3" and res = "12" + "3" = "123".

Loop finishes its execution and Line n4 prints 123 on to the console.

Bottom of Form

Top of Form

Question 43: **Correct**

**Question ID: UKOCP64791**

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 res = ""; //Line n1
6. String [] arr = {"Dog", null, "Friendly"};
7. for(String s : arr) { //Line n2
8. res += String.join("-", s); //Line n3
9. }
10. System.out.println(res); //Line n4
11. }
12. }

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

* 

**DogFriendly**

* 

**Dog-Friendly**

* 

**DognullFriendly**

**(Correct)**

* 

**Dog-null-Friendly**

* 

**An exception is thrown at runtime**

* 

**Compilation error**

**Explanation**

**UKOCP64791:**

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.

Given code compiles successfully.

At Line n1, variable 'res' infers to String.

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.

Let's check the iterations:

1st iteration: s refers to "Dog". `String.join(".", s)` returns "Dog" and res = "" + "Dog" = "Dog".

2nd iteration: s refers to null. `String.join(".", s)` returns "null" and res = "Dog" + "null" = "Dognull".

3rd iteration: s refers to "Friendly". `String.join(".", s)` returns "Friendly" and res = "Dognull" + "Friendly" = "DognullFriendly".

Loop finishes its execution and Line n4 prints DognullFriendly on to the console.

Bottom of Form

Top of Form

Question 44: **Incorrect**

**Question ID: UKOCP50116**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. for(var i = 5; i >= 1; i--) { //Line n1
6. System.out.println("\*".repeat(i)); //Line n2
7. }
8. }
9. }

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

* 

**\***

**\*\***

**\*\*\***

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

**(Incorrect)**

* 

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

**\*\*\***

**\*\***

**\***

* 

**\***

**\*\***

**\*\*\***

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

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

* 

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

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

**\*\*\***

**\*\***

**\***

**(Correct)**

* 

**Compilation error**

* 

**Line n2 causes runtime error**

**Explanation**

**UKOCP50116:**

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.

At Line n1, variable 'i' infers to int type, so given loop executes 5 times from values of i from 5 to 1.

Instance method 'repeat()' has been added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`

It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,

"A".repeat(3); returns "AAA".

1st iteration: System.out.println("\*".repeat(5)); => prints \*\*\*\*\* on to the console.

2nd iteration: System.out.println("\*".repeat(4)); => prints \*\*\*\* on to the console.

3rd iteration: System.out.println("\*".repeat(3)); => prints \*\*\* on to the console.

4th iteration: System.out.println("\*".repeat(2)); => prints \*\* on to the console.

5th iteration: System.out.println("\*".repeat(1)); => prints \* on to the console.

Hence, output is:

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

Bottom of Form

Top of Form

Question 45: **Correct**

**Question ID: UKOCP86560**

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 res = "";
6. loop: for(var i = 1; i <= 5; i++) { //Line n1
7. switch(i) {
8. case 1:
9. res += "UP ";
10. break;
11. case 2:
12. res += "TO ";
13. break;
14. case 3:
15. break;
16. case 4:
17. res += "DATE";
18. break loop;
19. }
20. }
21. System.out.println(String.join("-", res.split(" "))); //Line n2
22. }
23. }

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

* 

**UP**

* 

**TO**

* 

**DATE**

* 

**UP-TO**

* 

**TO-DATE**

* 

**UP-TO-DATE**

**(Correct)**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

**Explanation**

**UKOCP86560:**

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.

At Line n1, variable 'i' infers to int type, so given loop executes 5 times from values of i from 5 to 1.

To know more about join and split methods of String class, please check the URLs:

https://udayankhattry.com/join-string/

https://udayankhattry.com/split-string/

Let's solve the given code:

Initially, res = ""

1st iteration: i = 1. code of 'case 1:' is executed, res = "" + "UP " = "UP ". break; statement breaks out of the switch-case block and not the loop.

2nd iteration: i = 2. code of 'case 2:' is executed, res = "UP " + "TO " = "UP TO ". break; statement breaks out of the switch-case block.

3rd iteration: i = 3. code of 'case 3:' is executed. break; statement breaks out of the switch-case block.

4th iteration: i = 4. code of 'case 4:' is executed. res = "UP TO " + "DATE" = "UP TO DATE". break loop; statement breaks out of the for loop.

Line n2 is executed:

String.join("-", res.split(" "))

= String.join("-", "UP TO DATE".split(" "))

= String.join("-", ["UP","TO","DATE"]) //Splits on the basis of single space " ".

= "UP-TO-DATE"

Line n2 prints UP-TO-DATE on to the console.

Bottom of Form

Top of Form

Question 46: **Correct**

**Question ID: UKOCP30546**

Consider below code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String[] args) {
5. for(int x = 10, y = 11, z = 12; y > x && z > y; y++, z -= 2) {
6. System.out.println(x + y + z);
7. }
8. }
9. }

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

* 

**32**

* 

**33**

**(Correct)**

* 

**34**

* 

**33**

**32**

* 

**Compilation error**

**Explanation**

**UKOCP30546:**

Basic/Regular for loop has following form:

for ( [ForInit] ; [Expression] ; [ForUpdate] ) {...}

[ForInit] can be local variable initialization or the following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

[ForUpdate] can be following expressions:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

The [Expression] must have type boolean or Boolean, or a compile-time error occurs. If [Expression] is left blank, it evaluates to true.

All the expressions can be left blank; for(;;) is a valid for loop and it is an infinite loop as [Expression] is blank and evaluates to true.

Multiple comma separated statements are allowed for [ForInit] and [ForUpdate] expressions, where as [Expression] must be single expression which results in boolean or Boolean.

In the given for loop:

[ForInit] = int x = 10, y = 11, z = 12: It is allowed. 3 variables are declared and initialized. x = 10, y = 11 & z = 12.

[Expression] = y > x && z > y = (y > x) && (z > y) [Relational operator has higher precedence than logical AND]. This expression is valid and results in boolean value.

[ForUpdate] = y++, z -= 2. It is allowed. y is incremented by 1 and z is decremented by 2.

Let's check the loop's iteration:

1st iteration: x = 10, y = 11, z = 12. (y > x) && (z > y) = (11 > 10) && (12 > 11) = true && true = true. Loop's body is executed and prints x + y + z = 10 + 11 + 12 = 33 on to the console.

2nd iteration: [ForUpdate] is executed. y = 12, z = 10. (y > x) && (z > y) = (12 > 10) && (10 > 12) = true && false = false.

Control goes out of for loop and program terminates successfully.

Loop's body executes once and prints 33 on to the console.

Bottom of Form

Top of Form

Question 47: **Correct**

**Question ID: UKOCP27378**

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 flag = false;
6. do {
7. if(flag = !flag) { //Line n1
8. System.out.print(1); //Line n2
9. continue; //Line n3
10. }
11. System.out.print(2); //Line n4
12. } while(flag); //Line n5
13. }
14. }

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

* 

**1**

* 

**2**

* 

**12**

**(Correct)**

* 

**21**

* 

**212**

* 

**121**

* 

**112**

* 

**221**

* 

**Compilation Error**

**Explanation**

**UKOCP27378:**

Body of do-while loop is executed first and then condition is checked for the next iteration.

Initially, flag = false;

1st iteration: Boolean expression of if-block `flag = !flag` = `flag = !false` = `flag = true`: it assigns true to variable 'flag' and evaluates to true as well. Line n2 is executed and 1 is printed on to the console. Line n3 takes the control to the boolean expression of Line n5.

2nd iteration: As flag is true, boolean expression at Line n5 evaluates to true and control enters the loop's body. Boolean expression of if-block `flag = !flag` = `flag = !true` = `flag = false`: it assigns false to variable 'flag' and evaluates to false as well. Line n2 and Line n3 are not executed. Line n4 is executed, which prints 2 on to the console. Control goes to the boolean expression of Line n5. As flag is false, boolean expression at Line n5 evaluates to false and control exits the loop.

Program terminates successfully after printing 12 on to the console.

Bottom of Form

Top of Form

Question 48: **Correct**

**Question ID: UKOCP29555**

Consider below code of TestStudent.java file:

1. package com.udayankhattry.ocp;
3. class Student {
4. String name;
5. int age;
6. boolean result;
7. double height;
8. }
10. public class TestStudent {
11. public static void main(String[] args) {
12. Student stud = new Student(); //Line n1
13. System.out.println(stud.name + stud.height + stud.result + stud.age); //Line n2
14. }
15. }

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

* 

**null0.0false0**

**(Correct)**

* 

**null0false0**

* 

**null0.0ffalse0**

* 

**null0.0true0**

* 

**Compilation error**

**Explanation**

**UKOCP29555:**

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

For the Student object created at Line n1, 'name' is initialized to null, 'age' to 0, 'result' to false and 'height' to 0.0.

Hence, Line n2 prints null0.0false0

Bottom of Form

Top of Form

Question 49: **Incorrect**

**Question ID: UKOCP72387**

Consider below code of Test.java file:

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

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

* 

**>null**

**>0.0**

**>0.0**

* 

**>**

**>0.0**

**>0.0**

**(Correct)**

* 

**>**

**>0.0**

**>0.0f**

**(Incorrect)**

* 

**>null**

**>0.0**

**>0.0f**

**Explanation**

**UKOCP72387:**

Primitive type instance variables are initialized to respective zeros (byte: 0, short: 0, int: 0, long: 0L, float: 0.0f, double: 0.0, boolean: false, char: \u0000). When printed on the console; byte, short, int & long prints 0, float & double print 0.0, boolean prints false and char prints nothing or non-printable character (whitespace).

Reference type instance variables are initialized to null.

Bottom of Form

Top of Form

Question 50: **Correct**

**Question ID: UKOCP29554**

Consider below code of Counter.java file:

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

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

* 

**1**

**(Correct)**

* 

**2**

* 

**3**

* 

**4**

**Explanation**

**UKOCP29554:**

new Counter(); is invoked only once, hence only one Counter object is created in the memory. 'c1', 'c2', 'c3' and 'counter' are reference variables of Counter type and not Counter objects.

Bottom of Form

Top of Form

Question 51: **Incorrect**

**Question ID: UKOCP34906**

Consider below code of TestStyle.java file:

1. package com.udayankhattry.ocp;
3. class Style {
4. String pattern = "\*";
5. }
7. public class TestStyle {
8. public static void main(String[] args) {
9. var style = new Style(); //Line n1
10. System.out.println(style.pattern.repeat(5).length()); //Line n2
11. }
12. }

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

* 

**0**

* 

**1**

* 

**5**

**(Correct)**

* 

**null**

* 

**An exception is thrown at runtime**

* 

**Compilation error at Line n1**

**(Incorrect)**

* 

**Compilation error at Line n2**

**Explanation**

**UKOCP34906:**

Style class has an instance variable 'pattern', which is initialized to "\*" for all the instances.

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.

At Line n1, variable 'style' infers to Style type. An instance of Style class is created, pattern instance variable is initialized to "\*" and variable 'style' refers to this instance.

Instance method 'repeat()' was added to String class in Java 11 and it has the signature: `public String repeat(int count) {}`

It returns the new String object whose value is the concatenation of this String repeated 'count' times. For example,

"A".repeat(3); returns "AAA".

Expression at Line n2: style.pattern.repeat(5).length()

=> "\*".repeat(5).length() //style.pattern refers to String object "\*".

=> "\*\*\*\*\*".length() //"\*".repeat(5) returns "\*\*\*\*\*".

=> 5 //"\*\*\*\*\*".length() returns 5.

Line n2 prints 5 on to the console.

Bottom of Form

Top of Form

Question 52: **Incorrect**

**Question ID: UKOCP25197**

Consider below code of Test.java file:

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

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

* 

**true**

* 

**false**

**(Incorrect)**

* 

**Compilation error**

**(Correct)**

* 

**An exception is thrown at runtime**

**Explanation**

**UKOCP25197:**

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

As the name suggests, Local variable Type inference is applicable only for local variables and not for instance or class variables. Hence, `static var arr = new Boolean[1];` causes compilation error.

Bottom of Form

Top of Form

Question 53: **Correct**

**Question ID: UKOCP51000**

How can you force JVM to run Garbage Collector?

* 

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

* 

**By calling: System.gc();**

* 

**By setting the reference variable to null**

* 

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

**(Correct)**

**Explanation**

**UKOCP51000:**

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

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

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

Bottom of Form

Top of Form

Question 54: **Incorrect**

**Question ID: UKOCP88727**

Given code:

1. package com.udayankhattry.ocp;
3. public class Pen {
4. public static void main(String[] args) {
5. Pen p1 = new Pen(); //Line n1
6. Pen p2 = new Pen(); //Line n2
7. p1 = p2; //Line n3
8. p1 = null; //Line n4
9. }
10. }

When is the Pen object, created at Line n1, will be eligible for Garbage Collection?

* 

**After Line n2**

* 

**After Line n3**

**(Correct)**

* 

**After Line n4**

**(Incorrect)**

* 

**At the end of main method**

**Explanation**

**UKOCP88727:**

At Line n3, p1 starts referring to the object referred by p2(Created at Line n2).

So, after Line n3, object created at Line n1 becomes unreachable and thus eligible for Garbage Collection.

Bottom of Form

Top of Form

Question 55: **Correct**

**Question ID: UKOCP88728**

Consider below code of TestNewsPaper.java file:

1. package com.udayankhattry.ocp;
3. class NewsPaper {
5. }
7. public class TestNewsPaper {
8. public static void main(String[] args) {
9. new NewsPaper(); //Line n1
10. NewsPaper p = new NewsPaper(); // Line n2
11. change(p); //Line n3
12. System.out.println("About to end."); //Line n4
13. }
15. public static void change(NewsPaper np) { //Line n5
16. np = new NewsPaper(); //Line n6
17. }
18. }

On executing TestNewsPaper class, how many objects of NewsPaper class will be eligible for Garbage Collection at Line n4?

* 

**0**

* 

**1**

* 

**2**

**(Correct)**

* 

**3**

**Explanation**

**UKOCP88728:**

Object created at Line n1 becomes eligible for Garbage collection after Line n1 only, as there are no references to it. So We have one object marked for GC.

Line n2 creates another NewsPaper object.

Line n3 invokes change(NewsPaper) method. Parameter variable 'np' refers to the same NewsPaper object created at Line n2.

Line n6 creates a new NewsPaper object and variable 'np' refers to it.

Object created at Line n6 becomes unreachable after change(NewsPaper) method pops out of the STACK, and this happens after Line n3.

So at Line n4, we have two NewsPaper objects eligible for Garbage collection: Created at Line n1 and Created at Line n6.

Bottom of Form

Top of Form

Question 56: **Incorrect**

**Question ID: UKOCP39360**

Given code:

1. package com.udayankhattry.ocp;
3. class Outer {
4. private String name = "NOW OR NEVER";
5. //Insert inner class definition here
6. }
8. public class Test {
9. public static void main(String [] args) {
10. new Outer().new Inner().printName();
11. }
12. }

Which of the following Inner class definitions inserted in the Outer class, will print NOW OR NEVER in the output on executing Test class?

Select 2 options.

* 
  1. class Inner {
  2. public void printName() {
  3. System.out.println(this.name);
  4. }
  5. }

**(Incorrect)**

* 
  1. class Inner {
  2. public void printName() {
  3. System.out.println(name);
  4. }
  5. }

**(Correct)**

* 
  1. inner class Inner {
  2. public void printName() {
  3. System.out.println(name);
  4. }
  5. }
* 
  1. abstract class Inner {
  2. public void printName() {
  3. System.out.println(name);
  4. }
  5. }
* 
  1. class Inner {
  2. public void printName() {
  3. System.out.println(Outer.this.name);
  4. }
  5. }

**(Correct)**

**Explanation**

**UKOCP39360:**

Let's check all the options one by one:

1. class Inner {
2. public void printName() {
3. System.out.println(this.name);
4. }
5. }

✗ Keyword 'this' inside Inner class refers to currently executing instance of Inner class and not the Outer class. As, there is no instance variable 'name' inside Inner class, hence `this.name` causes compilation error.

1. class Inner {
2. public void printName() {
3. System.out.println(name);
4. }
5. }

✓ To access Outer class variable from within inner class you can use these 2 statements: System.out.println(name); OR System.out.println(Outer.this.name);

1. inner class Inner {
2. public void printName() {
3. System.out.println(name);
4. }
5. }

✗ There is no keyword with the name 'inner' in java.

1. abstract class Inner {
2. public void printName() {
3. System.out.println(name);
4. }
5. }

✗ As new Inner() is used in main method, hence cannot declare class Inner as abstract in this case. But note that abstract or final can be used with regular inner classes.

1. class Inner {
2. public void printName() {
3. System.out.println(Outer.this.name);
4. }
5. }

✓ To access Outer class variable from within inner class you can use these 2 statements: System.out.println(name); OR System.out.println(Outer.this.name);

Bottom of Form

Top of Form

Question 57: **Incorrect**

**Question ID: UKOCP42527**

Given code:

1. package com.udayankhattry.ocp;
3. public class Test {
4. class A {
5. void m() {
6. System.out.println("OVER AND OUT");
7. }
8. }
10. public static void main(String [] args) {
11. //Insert statement here
12. }
13. }

Which statements when inserted in the main(String []) method will print OVER AND OUT in the output?

Select 3 options.

* 
  1. A a1 = new Test().new A();
  2. a1.m();

**(Correct)**

* 
  1. Test.A a2 = new Test().new A();
  2. a2.m();

**(Correct)**

* 
  1. A a3 = this.new A();
  2. a3.m();
* 
  1. var a4 = new A();
  2. a4.m();
* 
  1. var a5 = new Test().new A();
  2. a5.m();

**(Correct)**

* 
  1. Test.A a6 = new A();
  2. a6.m();

**(Incorrect)**

**Explanation**

**UKOCP42527:**

There are 2 parts: 1st one is referring the name of inner class, A and 2nd one is creating the instance of inner class, A.

main method is inside Test class only, so inner class's name can be referred by 3 ways: A or Test.A or var.

As, A is Regular inner class, so instance of outer class is needed for creating the instance of inner class.

Let's check all the options one by one:

1. A a1 = new Test().new A();
2. a1.m();

✓ Inner class name is referred by A, hence left part is correct. `new Test().new A();` correctly creates an instance of inner class A.

1. Test.A a2 = new Test().new A();
2. a2.m();

✓ Inner class name is referred by Test.A, hence left part is correct. `new Test().new A();` correctly creates an instance of inner class A.

1. A a3 = this.new A();
2. a3.m();

✗ Keyword 'this' is not allowed inside static main method.

1. var a4 = new A();
2. a4.m();

✗ Inner class name is correctly referred by var, hence left part is correct. `new A();` causes compilation error as outer class's instance is needed to create the instance of inner class.

1. var a5 = new Test().new A();
2. a5.m();

✓ Inner class name is correctly referred by var, hence left part is correct. `new Test().new A();` correctly creates an instance of inner class A.

1. Test.A a6 = new A();
2. a6.m();

✗ Inner class name is referred by Test.A, hence left part is correct. `new A();` causes compilation error as outer class's instance is needed to create the instance of inner class.

Bottom of Form

Top of Form

Question 58: **Incorrect**

**Question ID: UKOCP43864**

Given code:

1. package com.udayankhattry.ocp;
3. class A {
4. private String str = "WINNERS NEVER QUIT";
5. public class B {
6. public B(String s) {
7. if(s != null)
8. str = s;
9. }
10. public void m1() {
11. System.out.println(str);
12. }
13. }
14. }
16. public class Test {
17. public static void main(String[] args) {
18. //Insert statement here
19. }
20. }

Which statement when inserted in the main(String []) method will print WINNERS NEVER QUIT (all characters in upper-case) in the output?

* 

**new A().new B().m1();**

* 

**new A.B().m1();**

* 

**new A().new B("Winners never quit").m1();**

**(Incorrect)**

* 

**new A().new B(null).m1();**

**(Correct)**

**Explanation**

**UKOCP43864:**

Let's check all the options one by one:

new A().new B().m1();

✗ new B() causes compilation error, as no-argument constructor is not defined in inner class B.

new A.B().m1();

✗ new A.B() is invalid syntax for creating the instance of Regular inner classes and also no-argument constructor is not defined in inner class B.

new A().new B("Winners never quit").m1();

✗ new A().new B("Winners never quit").m1(); is a valid syntax but it will print "Winners never quit" in the output and not "WINNERS NEVER QUIT".

new A().new B(null).m1();

✓ Instance variable 'str' of class A is initialized to "WINNERS NEVER QUIT". While creating the instance of inner class B, because of the not null check, statement `str = s;` is not executed. and therefore invocation of method m() prints WINNERS NEVER QUIT on to the console.

Bottom of Form

Top of Form

Question 59: **Incorrect**

**Question ID: UKOCP22920**

Given code:

1. package com.udayankhattry.ocp;
3. class Outer {
4. class Inner {
5. public void m() {
6. System.out.println("CHANGE IS GOOD");
7. }
8. }
9. }
11. public class Test {
12. public static void main(String[] args) {
13. //Insert statement here
14. }
15. }

Which statement when inserted in the main(String []) method will print CHANGE IS GOOD on to the console?

Select 2 options.

* 
  1. Outer.Inner obj1 = new Outer().new Inner();
  2. obj1.m();

**(Correct)**

* 
  1. Inner obj2 = new Outer().new Inner();
  2. obj2.m();

**(Incorrect)**

* 
  1. var obj3 = new Outer().new Inner();
  2. obj3.m();

**(Correct)**

* 
  1. Outer.Inner obj4 = this.new Inner();
  2. obj4.m();
* 
  1. Inner obj5 = this.new Inner();
  2. obj5.m();

**Explanation**

**UKOCP22920:**

There are 2 parts: 1st one is referring the name of inner class, Inner and 2nd one is creating an instance of inner class, Inner.

Now, main method is outside Outer class only, so inner class's name can be referred by 2 ways only: Outer.Inner or var.

As, Inner is Regular inner class, so instance of outer class is needed for creating the instance of inner class. Instance of outer class, Outer can only be obtained by new Outer(). So, instance of inner class can be created by: new Outer().new Inner();

Also note, keyword 'this' is not allowed within static main method.

Bottom of Form

Top of Form

Question 60: **Incorrect**

**Question ID: UKOCP65605**

Given code:

1. package com.udayankhattry.ocp;
3. class A {
4. A() {
5. System.out.print(1);
6. }
7. class B {
8. B() {
9. System.out.print(2);
10. }
11. }
12. }
14. public class Test {
15. public static void main(String [] args) {
16. B obj = new A().new B();
17. }
18. }

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

* 

**12**

**(Incorrect)**

* 

**21**

* 

**2**

* 

**1**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP65605:**

To refer to inner class name from outside the top level class, use the syntax: outer\_class.inner\_class.

In this case, correct syntax to refer B from Test class is: A.B and not B.

Bottom of Form

Top of Form

Question 61: **Incorrect**

**Question ID: UKOCP83041**

Given code:

1. package com.udayankhattry.ocp;
3. class Foo {
4. public static void m1() { //Line n1
5. System.out.println("Foo : m1()");
6. }
7. class Bar {
8. public static void m1() { //Line n2
9. System.out.println("Bar : m1()");
10. }
11. }
12. }
14. public class Test {
15. public static void main(String [] args) {
16. var foo = new Foo(); //Line n3
17. var bar = foo.new Bar(); //Line n4
18. bar.m1();
19. }
20. }

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

* 

**Foo : m1()**

* 

**Bar : m1()**

* 

**Compilation error in class Bar**

**(Correct)**

* 

**Compilation error in class Test**

**(Incorrect)**

* 

**Runtime exception**

**Explanation**

**UKOCP83041:**

Regular inner class Bar cannot define any static methods. Method m1() is static and hence Line n2 causes compilation error.

NOTE: Regular inner class cannot define anything static, except static final variables.

There is no issue with Line n3.

There are 2 parts in creating and using the instance of inner class: 1st one is referring the name of inner class, Bar and 2nd one is creating an instance of inner class, Bar.

Now, main method is outside outer class, so inner class's name can be referred by 2 ways only: Foo.Bar or var. As, Bar is Regular inner class, so instance of outer class is needed for creating the instance of inner class. Hence, `foo.new Bar();` is the correct statement. Line n4 compiles successfully.

Bottom of Form

Top of Form

Question 62: **Incorrect**

**Question ID: UKOCP16246**

Given code:

1. package com.udayankhattry.ocp;
3. class M {
4. private int num1 = 100;
5. class N {
6. private int num2 = 200;
7. }
9. public static void main(String[] args) {
10. M outer = new M();
11. M.N inner = outer.new N();
12. System.out.println(outer.num1 + inner.num2);
13. }
14. }

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

* 

**Compilation error**

**(Incorrect)**

* 

**300**

**(Correct)**

* 

**100**

* 

**200**

**Explanation**

**UKOCP16246:**

Outer class (M) code has access to all the members of inner class (N) including private members, hence inner.num2 doesn't cause any compilation error.

outer.num1 is 100 and inner.num2 is 200, therefore 300 is printed on to the console.

Bottom of Form

Top of Form

Question 63: **Correct**

**Question ID: UKOCP64765**

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

1. package com.udayankhattry.ocp;
3. class X {
4. class Y {
5. private void m() {
6. System.out.println("INNER");
7. }
8. }
10. public void invokeInner() {
11. var obj = new Y(); //Line n1
12. obj.m(); //Line n2
13. }
14. }
16. public class Test {
17. public static void main(String[] args) {
18. new X().invokeInner();
19. }
20. }

* 

**INNER**

**(Correct)**

* 

**Compilation error at Line n1 as instance of outer class (X) is needed to create the instance of inner class (Y)**

* 

**Compilation error at Line n2 as private method m() cannot be invoked outside the body of inner class (Y)**

* 

**Exception is thrown at runtime**

**Explanation**

**UKOCP64765:**

invokeInner() is instance method of outer class, X. So, implicit 'this' reference is available for this method. this reference refers to the currently executing instance of outer class, X. So Java compiler converts `var obj = new Y();` to `var obj = this.new Y();` and hence this syntax has no issues. So Line n1 is fine.

Because of the special relationship between Outer and inner class, Outer and Inner class can very easily access each other's private members. Hence, no issues with Line n2 as well.

Given code compiles and executes fine and prints INNER to the console.

Bottom of Form

Top of Form

Question 64: **Correct**

**Question ID: UKOCP61251**

Given code:

1. package com.udayankhattry.ocp;
3. class P {
4. private int var = 100;
5. class Q {
6. String var = "Java";
7. void print() {
8. System.out.println(var);
9. }
10. }
11. }
13. public class Test {
14. public static void main(String[] args) {
15. new P().new Q().print();
16. }
17. }

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

* 

**Java**

**(Correct)**

* 

**100**

* 

**Compilation error**

* 

**Exception is thrown at runtime**

**Explanation**

**UKOCP61251:**

In this example, inner class's variable var shadows the outer class's variable var. So output is Java.

Few points to note here:

1. If inner class shadows the variable of outer class, then Java compiler prepends 'this.' to the variable. System.out.println(var); is replaced by System.out.println(this.var);

2. If inner class does not shadow the variable of outer class, then Java compiler prepends "outer\_class.this." to the variable. So, if class Q doesn't shadow the variable of class P, then System.out.println(var); would be replaced by System.out.println(P.this.var);

In the given example, if you provide System.out.println(P.this.var); inside print() method, then output would be 100.

Bottom of Form

Top of Form

Question 65: **Incorrect**

**Question ID: UKOCP84374**

Given code:

1. package com.udayankhattry.ocp;
3. class Foo {
4. static { //static initializer block
5. System.out.print("A");
6. }
7. class Bar {
8. static { //static initializer block
9. System.out.print("B");
10. }
11. }
12. }
14. public class Test {
15. public static void main(String [] args) {
16. new Foo().new Bar();
17. }
18. }

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

* 

**AB**

* 

**BA**

**(Incorrect)**

* 

**Compilation error**

**(Correct)**

* 

**Exception is thrown at runtime**

**Explanation**

**UKOCP84374:**

Regular inner class cannot define anything static, except static final variables. In this case, static initializer block inside inner class Bar is not allowed.

Bottom of Form

Top of Form

Question 66: **Correct**

**Question ID: UKOCP44706**

Given code:

1. package com.udayankhattry.ocp;
3. class Outer {
4. Outer() {
5. System.out.print(2);
6. }
7. /\*INSERT 1\*/
9. class Inner {
10. Inner() {
11. System.out.print(4);
12. }
13. /\*INSERT 2\*/
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. new Outer().new Inner();
20. }
21. }

Currently on executing Test class, 24 is printed in the output.

Which of the following pairs will correctly replace /\*INSERT 1\*/ and /\*INSERT 2\*/ so that on executing Test class, 1234 is printed in the output?

Select 2 options.

* 

**Replace /\*INSERT 1\*/ with {System.out.print(1);}**

**Replace /\*INSERT 2\*/ with {System.out.print(3);}**

**(Correct)**

* 

**Replace /\*INSERT 1\*/ with static {System.out.print(1);}**

**Replace /\*INSERT 2\*/ with {System.out.print(3);}**

**(Correct)**

* 

**Replace /\*INSERT 1\*/ with {System.out.print(1);}**

**Replace /\*INSERT 2\*/ with static {System.out.print(3);}**

* 

**Replace /\*INSERT 1\*/ with static {System.out.print(1);}**

**Replace /\*INSERT 2\*/ with static {System.out.print(3);}**

**Explanation**

**UKOCP44706:**

Regular inner class cannot define anything static, except static final variables. So, replacing /\*INSERT 2\*/ with `static {System.out.print(3);}` will cause compilation error.

If a class contains, constructor, instance initializer block and static initializer block and constructor is invoked, then the execution order is:

static initializer block, instance initializer block and then constructor.

Bottom of Form

Top of Form

Question 67: **Correct**

**Question ID: UKOCP37185**

Given code:

1. package com.udayankhattry.ocp;
3. class Outer {
4. public void print(int x) {
5. class Inner {
6. public void getX() {
7. System.out.println(++x);
8. }
9. }
10. Inner inner = new Inner();
11. inner.getX();
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. new Outer().print(100);
18. }
19. }

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

* 

**100**

* 

**101**

* 

**Compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP37185:**

class Inner is method local inner class and it is accessing parameter variable x.

Starting with JDK 8, a method local inner class can access local variables and parameters of the enclosing block that are final or effectively final.

But the statement System.out.println(++x); tries to increment the value of variable x and hence compilation error.

Bottom of Form

Top of Form

Question 68: **Incorrect**

**Question ID: UKOCP87887**

Given code:

1. package com.udayankhattry.ocp;
3. class A {
4. public void someMethod(final String name) {
5. /\*INSERT\*/ {
6. void print() {
7. System.out.println("Hello " + name);
8. }
9. }
10. new B().print();
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. new A().someMethod("World!");
18. }
19. }

Which of the following options can replace /\*INSERT\*/ such that on executing Test class, "Hello World!" is displayed in the output?

Select 2 options.

* 

**public class B**

**(Incorrect)**

* 

**protected class B**

**(Incorrect)**

* 

**class B**

**(Correct)**

* 

**private class B**

* 

**final class B**

**(Correct)**

* 

**abstract class B**

**Explanation**

**UKOCP87887:**

Method-local inner classes cannot be defined using explicit access modifiers (public, protected and private) but non-access modifiers: final and abstract can be used with method-local inner class.

In this case, abstract is also not possible as new B() is used.

Bottom of Form

Top of Form

Question 69: **Incorrect**

**Question ID: UKOCP84376**

Given Code:

1. package com.udayankhattry.ocp;
3. class Outer {
4. public static void sayHello() {}
5. static {
6. class Inner {
7. /\*INSERT\*/
8. }
9. new Inner();
10. }
11. }
13. public class TestOuter {
14. public static void main(String[] args) {
15. Outer.sayHello();
16. }
17. }

Which of the following options can replace /\*INSERT\*/ such that on executing TestOuter class, "HELLO" is printed in the output?

Choose 2 options.

* 
  1. {
  2. System.out.println("HELLO");
  3. }

**(Correct)**

* 
  1. static {
  2. System.out.println("HELLO");
  3. }

**(Incorrect)**

* 
  1. Inner() {
  2. System.out.println("HELLO");
  3. }

**(Correct)**

* 
  1. Inner(String s) {
  2. System.out.println(s);
  3. }

**Explanation**

**UKOCP84376:**

static initializer block defined inside Outer class is invoked when static method sayHello is invoked.

Method-local inner class can be defined inside methods(static and non-static) and initializer blocks(static and non-static).

But like Regular inner class,  method-local inner class cannot define anything static, except static final variables.

new Inner(); invokes the no-argument constructor of Inner class. So, System.out.println("HELLO") can either be provided inside no-argument constructor or instance initializer block.

Bottom of Form

Top of Form

Question 70: **Incorrect**

**Question ID: UKOCP14065**

Given code:

1. package com.udayankhattry.ocp;
3. class A {
4. public void print(String name) {
5. class B {
6. B() {
7. System.out.println(name); //Line n1
8. }
9. }
10. }
11. B obj = new B(); //Line n2
12. }
14. public class Test {
15. public static void main(String[] args) {
16. new A().print("OCP"); //Line n3
17. }
18. }

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

* 

**OCP**

**(Incorrect)**

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

**(Correct)**

* 

**Compilation error at Line n3**

**Explanation**

**UKOCP14065:**

Instance of method-local inner class can only be created within the boundary of enclosing initializer block or enclosing method.

B obj = new B(); is written outside the closing curly bracket of print(String) method and hence Line n2 causes compilation error.

Starting with JDK 8, a method local inner class can access local variables and parameters of the enclosing block that are final or effectively final so no issues with Line n1.

Bottom of Form

Top of Form

Question 71: **Correct**

**Question ID: UKOCP62588**

Given code:

1. package com.udayankhattry.ocp;
3. class Outer {
4. private String msg = "A";
5. public void print() {
6. final String msg = "B";
7. class Inner {
8. public void print() {
9. System.out.println(this.msg);
10. }
11. }
12. Inner obj = new Inner();
13. obj.print();
14. }
15. }
17. public class Test {
18. public static void main(String[] args) {
19. new Outer().print();
20. }
21. }

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

* 

**Compilation error**

**(Correct)**

* 

**A**

* 

**B**

* 

**Exception is thrown at runtime**

**Explanation**

**UKOCP62588:**

Keyword 'this' inside method-local inner class refers to the instance of inner class.

In this case this.msg refers to msg variable defined inside Inner class but there is no msg variable inside Inner class. Hence, this.msg causes compilation error.

System.out.println(msg); would print B (msg shadows Outer class variable) and System.out.println(Outer.this.msg); would print A.

Bottom of Form

Top of Form

Question 72: **Incorrect**

**Question ID: UKOCP20738**

Given code:

1. package com.udayankhattr.ocp;
3. class Message {
4. public void printMessage() {
5. System.out.println("Hello!");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. Message msg = new Message() {}; //Line n1
12. msg.printMessage(); //Line n2
13. }
14. }

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

* 

**Compilation error at Line n1**

**(Incorrect)**

* 

**NullPointerException is thrown by Line n2**

* 

**Hello!**

**(Correct)**

* 

**HELLO!**

**Explanation**

**UKOCP20738:**

Message msg = new Message() {}; means msg doesn't refer to an instance of Message class but to an instance of un-named sub class of Message class, which means to an instance of anonymous inner class.

In this case, anonymous inner class code doesn't override printMessage() method of super class, Message.

So at runtime, msg.printMessage() method invokes the printMessage() method defined in super class (Message) and Hello! is printed to the console.

Bottom of Form

Top of Form

Question 73: **Incorrect**

**Question ID: UKOCP54504**

Given code:

1. package com.udayankhattry.ocp;
3. class Greet {
4. public void sayHello() {
5. System.out.println("Hello!");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. Greet obj = new Greet() {
12. public void SayHello() {
13. System.out.println("HELLO!");
14. }
15. };
16. obj.sayHello();
17. }
18. }

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

* 

**Compilation error**

**(Incorrect)**

* 

**Runtime error**

* 

**Hello!**

**(Correct)**

* 

**HELLO!**

**Explanation**

**UKOCP54504:**

It is a valid anonymous inner class syntax. But anonymous inner class code doesn't override sayHello() method of Greet class, rather it defines a new method SayHello (S in upper case).

Anonymous inner class allows to define methods not available in its super class, in this case SayHello() method.

`obj.sayHello();` statement invokes the sayHello method of super class, Greet and thus "Hello!" gets printed on to the console.

Bottom of Form

Top of Form

Question 74: **Incorrect**

**Question ID: UKOCP27389**

Given code:

1. package com.udayankhattry.ocp;
3. class Logger {
4. public void log() {
5. System.out.println("GO FOR IT");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. Logger obj = new Logger() {
12. public void Log() {
13. System.out.println("LET IT BE");
14. }
15. };
16. obj.Log();
17. }
18. }

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

* 

**Compilation error**

**(Correct)**

* 

**Runtime error**

* 

**GO FOR IT**

**(Incorrect)**

* 

**LET IT BE**

**Explanation**

**UKOCP27389:**

Anonymous inner class allows to define methods not available in its super class. Anonymous inner class in this case doesn't override the method of super class but it provides a new method Log [L in upper case].

obj is of Logger type, `obj.Log();` statement tries to invoke the Log() method of Logger class, which is not available and hence it causes compilation error.

Bottom of Form

Top of Form

Question 75: **Correct**

**Question ID: UKOCP50145**

Given code:

1. package com.udayankhattry.ocp;
3. class Printer {
4. public void getType() {
5. System.out.println("LASER");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. /\*INSERT\*/ obj = new Printer() { //Line n1
12. public void GetType() { //Line n2
13. System.out.println("INKJET");
14. }
15. };
16. obj.GetType(); //Line n3
17. }
18. }

And the options below:

1. Printer

2. Object

3. var

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

* 

**None of the given options**

* 

**Only one option**

**(Correct)**

* 

**Only two options**

* 

**All three options**

**Explanation**

**UKOCP50145:**

Anonymous inner class allows to define methods not available in its super class. Anonymous inner class in this case doesn't override the method of super class but it provides a new method GetType [G in upper case]. But this GetType() method cannot be invoked on reference variable of type Printer or Object as these classes don't define GetType() method.

GetType() method can only be invoked on the reference variable of anonymous inner class type.

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.

If /\*INSERT\*/ is replaced with var, then obj will infer to anonymous inner class type and then Line n3 will print INKJET on to the console. Hence, only one option can replace /\*INSERT\*/ successfully.

Bottom of Form

Top of Form

Question 76: **Correct**

**Question ID: UKOCP10944**

Given code:

1. package com.udayankhattry.ocp;
3. class Season {
4. public void printCurrentSeason() {
5. System.out.println("SUMMER");
6. }
7. }
9. public class Test {
10. public static void main(String[] args) {
11. var season = new Season() { //Line n1
12. @Override public void PrintCurrentSeason() { //Line n2
13. System.out.println("WINTER");
14. }
15. };
16. season.PrintCurrentSeason(); //Line n3
17. }
18. }

What is the result?

* 

**Line n1 causes compilation error**

* 

**Line n2 causes compilation error**

**(Correct)**

* 

**Line n3 causes compilation error**

* 

**Above program compiles successfully and on execution prints SUMMER on to the console**

* 

**Above program compiles successfully and on execution prints WINTER on to the console**

**Explanation**

**UKOCP10944:**

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.

At Line n1, variable 'season' infers to anonymous inner class type and hence Line n1 compiles successfully.

Season class has printCurrentSeason() method but anonymous inner class does not override this method, rather it provides a new method PrintCurrentSeason() [P in upper case]. Because of @Override annotation (which is used for overriding method), Line n2 causes compilation error.

For Line n3, as variable 'season' infers to anonymous inner class type, which defines PrintCurrentSeason() method and therefore Line n3 compiles successfully.

Bottom of Form

Top of Form

Question 77: **Correct**

**Question ID: UKOCP75425**

Given code:

1. package com.udayankhattry.ocp;
3. abstract class Greetings {
4. abstract void greet();
5. }
7. public class Test {
8. public static void main(String[] args) {
9. var obj = new Greetings() {
10. @Override public void greet() {
11. System.out.println("BELIEVE IN YOURSELF");
12. }
13. };
14. obj.greet();
15. }
16. }

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

* 

**Compilation error**

* 

**NullPointerException**

* 

**BELIEVE IN YOURSELF**

**(Correct)**

* 

**Nothing is printed on to the console**

**Explanation**

**UKOCP75425:**

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.

obj refers to an anonymous inner class instance extending from Greetings class and the anonymous inner class code correctly overrides greet() method.

Code executes and prints BELIEVE IN YOURSELF on to the console.

Bottom of Form

Top of Form

Question 78: **Incorrect**

**Question ID: UKOCP20750**

Below is the code of Test.java file:

1. package com.udayankhattry.ocp;
3. interface Flyable {
4. void fly();
5. }
7. public class Test {
8. public static void main(String[] args) {
9. /\*INSERT\*/
10. }
11. }

Which of the following options can replace /\*INSERT\*/ such that there are no compilation errors?

Select 2 options.

* 

**Flyable flyable = new Flyable();**

* 

**Flyable flyable = new Flyable(){};**

* 
  1. Flyable flyable = new Flyable() {
  2. public void fly() {
  3. System.out.println("Flying high");
  4. }
  5. }

**(Incorrect)**

* 
  1. Flyable flyable = new Flyable() {
  2. public void fly() {
  3. System.out.println("Flying high");
  4. }
  5. };

**(Correct)**

* 
  1. var flyable = new Flyable() {
  2. @Override
  3. public void fly() {
  4. System.out.println("Flying high");
  5. }
  7. public void stop() {
  8. System.out.println("Stopping");
  9. }
  10. };
  11. flyable.fly();
  12. flyable.stop();

**(Correct)**

**Explanation**

**UKOCP20750:**

Let's check all the options one by one:

Flyable flyable = new Flyable();

✗ Can't instantiate an interface.

Flyable flyable = new Flyable(){};

✗ fly() method has not been implemented.

1. Flyable flyable = new Flyable() {
2. public void fly() {
3. System.out.println("Flying high");
4. }
5. }

✗ semicolon is missing at the end

1. Flyable flyable = new Flyable() {
2. public void fly() {
3. System.out.println("Flying high");
4. }
5. };

✓ Correct syntax, fly() method has been implemented successfully.

1. var flyable = new Flyable() {
2. @Override
3. public void fly() {
4. System.out.println("Flying high");
5. }
7. public void stop() {
8. System.out.println("Stopping");
9. }
10. };
11. flyable.fly();
12. flyable.stop();

✓ 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.

Variable 'flyable' infers to anonymous inner class implementing Flyable interface. The anonymous inner class correctly overrides fly() method and provides a new stop() method. Both these methods can be invoked on reference variable 'flyable' as it is of anonymous inner class type.

Bottom of Form

Top of Form

Question 79: **Incorrect**

**Question ID: UKOCP84358**

Given code:

1. package com.udayankhattry.ocp;
3. interface I1 {
4. void m1();
5. }
7. public class Test {
8. public static void main(String[] args) {
9. I1 i1 = new I1() {
10. @Override
11. public void m1() {
12. System.out.println(1234);
13. }
14. }
15. i1.m1();
16. }
17. }

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

* 

**1234**

**(Incorrect)**

* 

**No output**

* 

**Compilation error**

**(Correct)**

* 

**Runtime exception**

**Explanation**

**UKOCP84358:**

Semicolon is missing just before the statement i1.m1(); Wrong syntax of anonymous inner class.

Bottom of Form

Top of Form

Question 80: **Correct**

**Question ID: UKOCP88717**

Below is the code of Test.java file:

1. package com.udayankhattry.ocp;
3. public class Test {
4. public static void main(String [] args) {
5. System.out.println(new Object() {
6. public String toString() {
7. return "ANONYMOUS";
8. }
9. });
10. }
11. }

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

* 

**ANONYMOUS**

**(Correct)**

* 

**Some text containing @ symbol**

* 

**Compilation error**

* 

**Runtime exception**

**Explanation**

**UKOCP88717:**

System.out.println(new Object()); invokes the toString() method defined in Object class, which prints fully qualified class name, @ symbol and hexadecimal value of hash code [Similar to java.lang.Object@15db9742]

In the given code, an instance of anonymous class extending Object class is passed to System.out.println() method and the anonymous class overrides the toString() method. Thus, at runtime overriding method is invoked, which prints ANONYMOUS to the console.

Bottom of Form

Top of Form

Question 81: **Incorrect**

**Question ID: UKOCP66946**

Below is the code of TestSellable.java file:

1. package com.udayankhattry.ocp;
3. interface Sellable {
4. double getPrice();
5. }
7. public class TestSellable {
8. private static void printPrice(Sellable sellable) {
9. System.out.println(sellable.getPrice());
10. }
12. public static void main(String[] args) {
13. /\*INSERT\*/
14. }
15. }

Which of the following options can replace /\*INSERT\*/ such that there are no compilation errors?

Select 3 options.

* 

**printPrice(null);**

**(Correct)**

* 

**printPrice(new Sellable());**

**(Incorrect)**

* 
  1. printPrice(new Sellable() {
  3. });

**(Incorrect)**

* 
  1. printPrice(new Sellable() {
  2. @Override
  3. public double getPrice() {
  4. return 45.34;
  5. }
  6. });

**(Correct)**

* 
  1. var obj = new Sellable() {
  2. @Override
  3. public double getPrice() {
  4. return 20.21;
  5. }
  7. public Sellable getThisObject() {
  8. return this;
  9. }
  10. };
  11. printPrice(obj.getThisObject());

**(Correct)**

**Explanation**

**UKOCP66946:**

Instance of anonymous inner class can be assigned to static variable, instance variable, local variable, method parameter and return value.

In this question, anonymous inner class instance is assigned to method parameter.

printPrice(null);

✓ No compilation error as asked in the question but it would throw NullPointerException at runtime.

printPrice(new Sellable());

✗ Cannot create an instance of Sellable type.

printPrice(new Sellable() {});

✗ getPrice() method has not been implemented.

1. printPrice(new Sellable() {
2. @Override
3. public double getPrice() {
4. return 45.34;
5. }
6. });

✓ anonymous inner class correctly implements the getPrice() method. On execution 45.34 would be printed on to the console.

1. var obj = new Sellable() {
2. @Override
3. public double getPrice() {
4. return 20.21;
5. }
7. public Sellable getThisObject() {
8. return this;
9. }
10. };
11. printPrice(obj.getThisObject());

✓ Variable 'obj' infers to anonymous inner class implementing Sellable interface. The anonymous inner class correctly overrides getPrice() method and provides a new getThisObject() method. Both these methods can be invoked on reference variable 'obj' as it is of anonymous inner class type.

getThisObject() method returns the currently executing instance of anonymous inner class, which implements Sellable interface. Hence, instance returned by obj.getThisObject() method will always be subtype of Sellable interface. It would compile successfully and on execution would print 20.21 on to the console.

Bottom of Form

Top of Form

Question 82: **Correct**

**Question ID: UKOCP71060**

Given code:

1. package com.udayankhattry.ocp;
3. interface Logger {
4. Object get();
5. void log();
6. }
8. public class Test {
9. private static void testLogger(Logger logger) {
10. logger.log();
11. }
13. public static void main(String[] args) {
14. var obj = new Logger() { //Line n1
15. @Override
16. public Logger get() { //Line n2
17. return this;
18. }
20. @Override
21. public void log() {
22. System.out.println("WINNERS NEVER QUIT"); //Line n3
23. }
24. };
25. testLogger(obj.get()); //Line n4
26. }
27. }

What is the result?

* 

**Line n1 causes compilation error**

* 

**Line n2 causes compilation error**

* 

**Line n3 causes compilation error**

* 

**Line n4 cuases compilation error**

* 

**Test class compiles successfully and on execution prints WINNERS NEVER QUIT on to the console**

**(Correct)**

* 

**Test class compiles successfully and on execution prints nothing**

**Explanation**

**UKOCP71060:**

Variable 'obj' infers to anonymous inner class implementing Logger interface. The anonymous inner class correctly overrides get() and log() methods. Both these methods can be invoked on reference variable 'obj' as it is of anonymous inner class type.

get() method returns the currently executing instance of anonymous inner class, which implements Logger interface. Hence, instance returned by obj.get() method will always be subtype of Logger interface.

Given code compiles successfully and on execution prints WINNERS NEVER QUIT on to the console.

Bottom of Form

Top of Form

Question 83: **Correct**

**Question ID: UKOCP40345**

Can an anonymous inner class implement multiple interfaces?

* 

**Yes**

* 

**No**

**(Correct)**

**Explanation**

**UKOCP40345:**

Unlike other inner classes, an anonymous inner class can either extend from one class or can implement one interface. It cannot extend and implement at the same time and it cannot implement multiple interfaces.

Bottom of Form

Top of Form

Question 84: **Correct**

**Question ID: UKOCP38026**

Below is the code to Test.java file:

1. package com.udayankhattry.ocp;
3. enum ShapeType {
4. CIRCLE, SQUARE, RECTANGLE;
5. }
7. abstract class Shape {
8. private ShapeType type = ShapeType.SQUARE; //default ShapeType
10. Shape(ShapeType type) {
11. this.type = type;
12. }
14. public ShapeType getType() {
15. return type;
16. }
18. abstract void draw();
19. }
21. public class Test {
22. public static void main(String[] args) {
23. Shape shape = new Shape() {
24. @Override
25. void draw() {
26. System.out.println("Drawing a " + getType());
27. }
28. };
29. shape.draw();
30. }
31. }

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

* 

**Drawing a CIRCLE**

* 

**Drawing a SQUARE**

* 

**Drawing a RECTANGLE**

* 

**Compilation error**

**(Correct)**

**Explanation**

**UKOCP38026:**

At the time of creating the instance of anonymous inner class, new Shape() is used, which means it is looking for a no-argument constructor in anonymous inner class code, which would invoke the no-argument constructor of super class, Shape. But as parameterized constructor is specified in Shape class, so no-argument constructor is not provided by the compiler and hence compilation error.

To correct the compilation error, pass the enum constant while instantiating anonymous inner class.

Shape shape = new Shape(ShapeType.CIRCLE) {...}; or you can even pass null: Shape shape = new Shape(null) {...};  OR provide the no-argument constructor in the Shape class: Shape(){}

Bottom of Form

Top of Form

Question 85: **Incorrect**

**Question ID: UKOCP42523**

Below is the code of Test.java file:

1. package com.udayankhattry.ocp;
3. class A {
4. static class B {
6. }
7. }
9. public class Test {
10. /\*INSERT\*/
11. }

Which of the following options can replace /\*INSERT\*/ such that there are no compilation errors?

* 

**B obj = new B();**

**(Incorrect)**

* 

**B obj = new A.B();**

* 

**A.B obj = new A.B();**

**(Correct)**

* 

**A.B obj = new A().new B();**

**Explanation**

**UKOCP42523:**

In this case, you have to write code outside class A.

B is a static nested class and outside class A, class B's name can be referred by A.B or var.

Instance of class B can be created by 'new A.B();'.

Bottom of Form

Top of Form

Question 86: **Incorrect**

**Question ID: UKOCP64776**

Below the code of A.java file:

1. package com.udayankhattry.ocp;
3. public class A {
4. private static class B {
5. private void log() {
6. System.out.println("BE THE CHANGE");
7. }
8. }
10. public static void main(String[] args) {
11. /\*INSERT\*/
12. }
13. }

Which of the following options can replace /\*INSERT\*/ such that there on executing class A, output is: BE THE CHANGE?

Select 3 options.

* 
  1. B obj1 = new B();
  2. obj1.log();

**(Correct)**

* 
  1. B obj3 = new A().new B();
  2. obj3.log();

**(Incorrect)**

* 
  1. A.B obj2 = new A.B();
  2. obj2.log();

**(Correct)**

* 
  1. A.B obj4 = new A().new B();
  2. obj4.log();
* 
  1. var obj5 = new A.B();
  2. obj5.log();

**(Correct)**

**Explanation**

**UKOCP64776:**

static nested class can use all 4 access modifiers (public, protected, default and private) and 2 non-access modifiers (final and abstract).

static nested class can contain all type of members, static as well as non-static. This behavior is different from other inner classes as other inner classes don't allow to define anything static, except static final variables. This is the reason static nested class is not considered as inner class.

There are 2 parts in accessing static nested class: 1st one to access the static nested class's name and 2nd one to instantiate the static nested class.

Within the top-level class, a static nested class's name can be referred by 3 ways: TOP-LEVEL-CLASS.STATIC-NESTED-CLASS or STATIC-NESTED-CLASS or var.

In this case, either use A.B or B or var. Now for instantiating the static nested class, an instance of enclosing class cannot be used, which means in this case, I can't write new A().new B(). Correct way to instance static nested class is: new A.B(); but as main method is inside class A, hence I can even write new B(); as well.

Top-level class can easily access private members of inner or static nested class, so no issues in invoking log() method from within the definition of class A.

Bottom of Form

Top of Form

Question 87: **Correct**

**Question ID: UKOCP82180**

Below is the code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Outer {
4. abstract static class Animal { //Line n1
5. abstract void eat();
6. }
8. static class Dog extends Animal { //Line n2
9. void eat() { //Line n3
10. System.out.println("DOG EATS BISCUITS");
11. }
12. }
13. }
15. public class Test {
16. public static void main(String[] args) {
17. Outer.Animal animal = new Outer.Dog(); //Line n4
18. animal.eat();
19. }
20. }

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

* 

**Compilation error at Line n1**

* 

**Compilation error at Line n2**

* 

**Compilation error at Line n3**

* 

**Compilation error at Line n4**

* 

**DOG EATS BISCUITS**

**(Correct)**

**Explanation**

**UKOCP82180:**

A class can have multiple static nested classes. static nested class can use all 4 access modifiers (public, protected, default and private) and 2 non-access modifiers (final and abstract). No issues at Line n1.

static nested class can extend from a class and can implement multiple interfaces so Line n2 compiles fine. No overriding rules were broken while overriding eat() method, so no issues at Line n3.

Test class is outside the boundary of class Outer. So Animal can be referred by Outer.Animal and Dog can be referred by Outer.Dog. Polymorphism is working in this case, super class (Outer.Animal) reference variable is referring to the instance of sub class (Outer.Dog). So, no issues at Line n4 as well.

Test class compiles and executes successfully and prints DOG EATS BISCUITS on to the console.

Bottom of Form

Top of Form

Question 88: **Correct**

**Question ID: UKOCP39359**

Below is the code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Outer {
4. private static int i = 10;
5. private int j = 20;
7. static class Inner {
8. void add() {
9. System.out.println(i + j);
10. }
11. }
12. }
14. public class Test {
15. public static void main(String[] args) {
16. Outer.Inner inner = new Outer.Inner();
17. inner.add();
18. }
19. }

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

* 

**Compilation error in Test class code**

* 

**Compilation error in Inner class code**

**(Correct)**

* 

**30**

* 

**Exception is thrown at runtime**

**Explanation**

**UKOCP39359:**

static nested class cannot access non-static member of the Outer class using static reference. Hence, usage of variable j in Inner class causes compilation error.

Bottom of Form

Top of Form

Question 89: **Incorrect**

**Question ID: UKOCP86544**

Below is the code of Test.java file:

1. package com.udayankhattry.ocp;
3. class Outer {
4. static class Inner {
5. static void greetings(String s) {
6. System.out.println(s);
7. }
8. }
9. }
11. public class Test {
12. public static void main(String[] args) {
13. /\*INSERT\*/
14. }
15. }

Which of the following 2 options can replace /\*INSERT\*/ such that on executing class Test, output is: HELLO!?

* 
  1. Outer.Inner inner1 = new Outer().new Inner();
  2. inner1.greetings("HELLO!");

**(Incorrect)**

* 
  1. Outer.Inner inner2 = new Outer.Inner();
  2. inner2.greetings("HELLO!");

**(Correct)**

* 

**Outer.Inner.greetings("HELLO!");**

**(Correct)**

* 

**Inner.greetings("HELLO!");**

**Explanation**

**UKOCP86544:**

Outside of top-level class, Outer, static nested class's name can be referred by using TOP-LEVEL-CLASS.STATIC-NESTED-CLASS or var. So, in this case, correct way to refer static nested class is Outer.Inner. greetings(String) is a static method, so it can be invoked by using the class name, which is by the statement: Outer.Inner.greetings("...");

Even though it is not preferred to invoke static method in non-static manner, but you can use the instance of class to invoke its static method.

To Create the instance of static nested class, syntax is: new TOP-LEVEL-CLASS.STATIC-NESTED-CLASS(...);

in this case, new Outer.Inner();

Bottom of Form

Top of Form

Question 90: **Correct**

**Question ID: UKOCP37179**

Will below code compile successfully?

1. package com.udayankhattry.ocp;
3. class Outer {
4. interface I1 {
5. void m1();
6. }
7. }

* 

**Yes**

**(Correct)**

* 

**No**

**Explanation**

**UKOCP37179:**

interface can be nested inside a class. Class Outer is top-level class and interface I1 is implicitly static.

Static nested interface can use all 4 access modifiers(public, protected, default and private).

Bottom of Form

Top of Form

Question 91: **Correct**

**Question ID: UKOCP14074**

Will below code compile successfully?

1. package com.udayankhattry.ocp;
3. interface I1 {
4. void m1();
6. interface I2 {
7. void m2();
8. }
10. abstract class A1 {
11. public abstract void m3();
12. }
14. class A2 {
15. public void m4() {
16. System.out.println(4);
17. }
18. }
19. }

* 

**Yes**

**(Correct)**

* 

**No**

**Explanation**

**UKOCP14074:**

interface I2 is implicitly public and static (Nested interface). class A1 is implicitly public and static (Nested class). class A2 is implicitly public and static (Nested class).

You cannot explicitly specify protected and private for nested classes and nested interfaces inside an interface.

Bottom of Form