Question **1** of **55**

1. Question

```
package com.skillcertpro.ocp;

public class Test {
static Character obj;
char c = obj.charValue();

public static void main(String[] args) {
System.out.println("DREAMS COME TRUE");
}
}
```

On execution, does Test class print DREAMS COME TRUE on to the console?

- (●) **Yes**
- ( ) No

Correct

SCP57630:

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

Variable 'c' is an instance variable, so char c = obj.charValue(); is not executed. Class is loaded successfully in the memory and DREAMS COME TRUE is printed on to the console.

NOTE: `new Test();` statement inside main(String []) method will throw NullPointerException but not ExceptionInInitializerError.

2. Question

Given code:

```
package com.skillcertpro.ocp;

public class Test {
final static StringBuilder sb = new StringBuilder("A");
static int x = 2;
public static void main(String[] args) {
switch (x) {
default:
sb.append("B"); //Line n1
case 1:
case 3:
case 5:
case 7:
case 9:
sb.append("C"); //Line n2
break;
}
System.out.println(x + ":" + sb.toString()); //Line n3
}
static {
sb.append("Z");
}
{ x += 1; }
}
```

What is the result?

- Compilation error
- **2:AZBC**
- 2:ABCZ
- 2:ABC
- 3:AZC
- 3:ACZ

Correct

SCP24735:

Variables 'sb' and 'x' are of static type, so both static and instance initializer blocks can access these.

Also because of the final keyword, variable 'sb' cannot be changed but the StringBuilder object it refers to can be changed.

Given code compiles successfully.

We are not creating the instance of Test class, so instance initializer block will not be executed. Only static initializer block will be executed in this case.

If static variable declaration / initialization statements are present along with static initializer blocks, then these are invoked in top to bottom order. So, for the given code, order of execution will be:

1. final static StringBuilder sb = new StringBuilder("A"); => StringBuilder object ["A"] is created and sb refers to it.

2. static int x = 2; => Variable x of int type is created and it stores 2.

3. static { sb.append("Z"); } => 'sb' now refers to ["AZ"]

4. Code inside main[String []) method is executed. x = 2 so default case (Line n1) is executed, 'sb' now refers to ["AZB"]. Because of the fall-through logic, Line n2 is also executed, 'sb' refers to ["AZBC"]. Line n3 prints 2:AZBC.

3. Question

Given code:

```
package com.skillcertpro.ocp;

public class Test {
Integer i = 10; //Line n1
{
Integer i = 2; //Line n2
}
public static void main(String[] args) {
System.out.println(new Test().i); //Line n3
}
{ i--; } //Line n4
}
```

What is the result?

- Compilation error
- 10
- **9** ✓
- 2 ✗
- 1
- 11

Incorrect

SCP78595:

If instance variable declaration / initialization statements are present along with instance initializer blocks, then these are invoked in top to bottom order. So, for the given code, order of execution will be:

1. Integer i = 10; => Instance variable 'i' is created and it refers to Integer object 10.

2. { Integer i = 2; } => This instance initializer block creates a local variable 'i' and refres to Integer object 2. This local variable shadows the instance variable, created at Line n1.

3. { i--; } => This is equivalent to `i = i – 1;` A new Integer object 9 is created and instance variable 'i' refers to it.

Line n3 prints 9 on to the console.

4. Question

Given code:

```
package com.skillcertpro.ocp;

class Super {
{ System.out.print("A");}
Super(String str) {
System.out.print(str);
}
static {
System.out.print("1");
}
{ System.out.print("B");}
}

class Sub extends Super {
static { System.out.print("2"); }
Sub(String str) {
super(str);
System.out.print(str);
}
{ System.out.print("C"); }
static {
System.out.print("3");
}
}

public class Test {
public static void main(String[] args) {
new Sub("Z");
}
}
```

What is the result?

- 1AZB23ZC
- 1ABZ23CZ
- 123AZBZC
- **123ABZCZ**
- 1ZAB23ZC

Incorrect

SCP83920:

Order of execution is:

static initializer block of Parent (Super) class, in top to bottom order

static initializer block of Child (Sub) class, in top to bottom order

instance initializer block of Parent (Super) class, in top to bottom order

Statements inside the constructor of Parent (Super) class

instance initializer block of Child (Sub) class, in to to bottom order

Statements inside the constructor of Child (Sub) class

Based on above order of execution, output will be:

123ABZCZ

5. Question

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

- const int x = 10;
- static int x = 10;
- **final static int x = 10;**
- int x = 10;

Correct

SCP20374:

Fields declared with final are constant fields.

Question **6** of **55**

6. Question

_____ modifier is most restrictive and _____ modifier is least restrictive.

Which of the following options (in below specified order) can be filled in above blank spaces?

- public, private
- **private, public**
- default (with no access modifier specified), public
- protected, public
- default (with no access modifier specified), protected

Correct

SCP79443:

'private' is most restrictive, then comes 'default (with no access modifier specified)', after that 'protected' and finally 'public' is least restrictive.

Question **7** of **55**

7. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Base {
String msg = "INHALE"; //Line n1
}

class Derived extends Base {
Object msg = "EXHALE"; //Line n2
}

public class Test {
public static void main(String[] args) {
Base obj1 = new Base(); //Line n3
Base obj2 = new Derived(); //Line n4
Derived obj3 = (Derived) obj2; //Line n5
var var = obj1.msg + "-" + obj2.msg + "-" + obj3.msg; //Line n6
System.out.println(var); //Line n7
}
}
```

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

- Line n2 causes compilation error
- Line n5 throws Exception at runtime
- Line n6 causes compilation error
- It executes successfully and prints INHALE-EXHALE-EXHALE
- **It executes successfully and prints INHALE-INHALE-EXHALE**
- It executes successfully and prints INHALE-INHALE-INHALE

Incorrect

SCP48701:

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n2 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

At Line n3, obj1 is of Base type and refers to an instance of Base class.

At Line n4, obj2 is of Base type and refers to an instance of Derived class.

At Line n5, as obj2 refers to an instance of Derived class, hence typecasting it to Derived type doesn't cause any Exception. obj3 is of Derived type and refers to an instance of Derived class.

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.

Let's check the expression of Line n6:

obj1.msg + "-" + obj2.msg + "-" + obj3.msg;

=> (obj1.msg + "-") + obj2.msg + "-" + obj3.msg; //+ operator is left to right associative and behaves as concatenation operator as one of the operand is of String type.

=> ((obj1.msg + "-") + obj2.msg) + "-" + obj3.msg;

=> (((obj1.msg + "-") + obj2.msg) + "-") + obj3.msg;

Let's solve the expression now:

=> (((“INHALE” + “-“) + obj2.msg) + “-“) + obj3.msg; //obj1 is of Base type, hence obj1.msg refers to “INHALE”

=> ((“INHALE-” + obj2.msg) + “-“) + obj3.msg;

=> ((“INHALE-” + “INHALE”) + “-“) + obj3.msg; //obj2 is of Base type, hence obj2.msg refers to “INHALE”

=> (“INHALE-INHALE” + “-“) + obj3.msg;

=> “INHALE-INHALE-” + obj3.msg;

In above expression, left operand is of String type and right operand is of Object type, so toString() method is invoked. So, given expression is similar to:

=> “INHALE-INHALE-” + obj3.msg.toString();

=> “INHALE-INHALE-” + “EXHALE”; //As obj3.msg is of Object type and refers to an instance of String type, hence toString() method on “EXHALE” instance is invoked and this returns “EXHALE”.

=> “INHALE-INHALE-EXHALE”;

So, at Line n6, variable 'var' is of String type and refers to “INHALE-INHALE-EXHALE”.

Line n7 prints INHALE-INHALE-EXHALE on to the console.

8. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Base {
static void print() { //Line n1
System.out.println("BASE");
}
}

class Derived extends Base {
static void print() { //Line n2
System.out.println("DERIVED");
}
}

public class Test {
public static void main(String[] args) {
Base b = null;
Derived d = (Derived) b; //Line n3
d.print(); //Line n4
}
}
```

Which of the following statements is true for above code?

Incorrect

SCP55474:

print() method at Line n2 hides the method at Line n1. So, no compilation error at Line n2.

Reference variable 'b' is of type Base, so `(Derived) b` does not cause any compilation error. Moreover, at runtime it will not throw any ClassCastException as well because b is null. Had 'b' been referring to an instance of Base class [Base b = new Base();], `(Derived) b` would have thrown ClassCastException.

d.print(); doesn't cause any compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Derived.print(); is the preferred syntax.

d.print(); invokes the static print() method of Derived class and prints DERIVED on to the console.

- ○ Line n2 causes compilation error
- ○ Line n3 causes compilation error
- ◉ Line n4 causes compilation error
- ○ Code compiles successfully and on execution Line n3 throws an exception
- ○ Code compiles successfully and on execution prints BASE on to the console
- ◉ **Code compiles successfully and on execution prints DERIVED on to the console**

Question **9** of **55**

9. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Paper {
static String getType() { //Line n1
return “GENERIC”;
}
}

class RuledPaper extends Paper {
String getType() { //Line n2
return “RULED”;
}
}

public class Test {
public static void main(String[] args) {
Paper paper = new RuledPaper(); //Line n3
System.out.println(paper.getType()); //Line n4
}
}
```

Which of the following statements is true for above code?

Incorrect

SCP54140:

Instance method of subclass cannot override the static method of superclass.

Instance method at Line n2 tries to override the static method at Line n1 and hence Line n2 causes compilation error.

There is no issue with Line n3 as reference variable of superclass can refer to an instance of subclass.

At Line n4, paper.getType() doesn't cause compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Paper.getType() is the preferred syntax.

- **Compilation error in RuledPaper class**
- Compilation error in Test class
- Code compiles successfully and on execution prints GENERIC on to the console
- Code compiles successfully and on execution prints RULED on to the console

10. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class M {
public void main(String[] args) { //Line n1
System.out.println("M");
}
}

class N extends M {
public static void main(String[] args) { //Line n2
new M().main(args); //Line n3
}
}

public class Test {
public static void main(String[] args) {
N.main(args); //Line n4
}
}
```

Correct

SCP47854:

The static method of subclass cannot hide the instance method of superclass. static main(String []) method at Line n2 tries to hide the instance main(String []) method at Line n1 and hence Line n2 causes compilation error.

There is no issue with Line n3 as it is a valid syntax to invoke the instance main(String []) method of M class.

No issue with Line n4 as well as it correctly invokes static main(String []) method of N class.

Which of the following statements is true for above code?

- ○ Line n1 causes compilation error
- ● **Line n2 causes compilation error**
- ○ Line n3 causes compilation error
- ○ Line n4 causes compilation error
- ○ It executes successfully and prints M on to the console

11. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Parent {
String quote = "MONEY DOESN'T GROW ON TREES";
}

class Child extends Parent {
String quote = "LIVE LIFE KING SIZE";
}

class GrandChild extends Child {
String quote = "PLAY PLAY PLAY";
}

public class Test {
public static void main(String[] args) {
GrandChild gc = new GrandChild();
System.out.println(/*INSERT*/);
}
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print MONEY DOESN'T GROW ON TREES on to the console?

Select 2 options.

- [x] gc.quote
- [ ] (Parent)gc.quote
- [ ] **((Parent)gc).quote**
- [ ] **((Parent)(Child)gc).quote**
- [ ] (Parent)(Child)gc.quote

Incorrect

SCP81729:

As instance variables are hidden by subclasses and not overridden, therefore instance variable can be accessed by using explicit casting.

Let's check all the options one by one:

gc.quote => It refers to "PLAY PLAY PLAY" as gc is of GrandChild class.

(Parent)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Parent type. This would cause compilation error.

((Parent)gc).quote => Variable 'gc' is casted to Parent type, so this expression refers to "MONEY DOESN'T GROW ON TREES". It is one of the correct options.

((Parent)(Child)gc).quote => 'gc' is of GrandChild type, it is first casted to Child and then to Parent type and finally quote variable is accessed, so this expression refers to "MONEY DOESN'T GROW ON TREES". It is also one of the correct options.

(Parent)(Child)gc.quote => gc.quote will be evaluated first as dot (.) operator has higher precedence than cast. gc.quote refers to String, hence it cannot be casted to Child type. This would cause compilation error.

12. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class X {
void greet() {
System.out.println("Good Morning!");
}
}

class Y extends X {
void greet() {
System.out.println("Good Afternoon!");
}
}

class Z extends Y {
void greet() {
System.out.println("Good Night!");
}
}

public class Test {
public static void main(String[] args) {
X x = new Z();
x.greet(); //Line n1
((Y)x).greet(); //Line n2
((Z)x).greet(); //Line n3
}
}
```

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

- ○ Compilation error
- ○ An exception is thrown at runtime It compiles successfully and on execution prints below: Good Night! Good Afternoon! Good Morning!
- ○ It compiles successfully and on execution prints below: Good Night! Good Night! Good Night!
- ○ It compiles successfully and on execution prints below: Good Morning! Good Morning! Good Morning!

Incorrect

SCP23427:

Variable x is of X type (superclass) and refers to an instance of Z type (subclass).

At Line n1, compiler checks whether greet() method is available in class X or not. As greet() method is available in class X, hence no compilation error for Line n1.

At Line n2, x is casted to Y and compiler checks whether greet() method is available in class Y or not. As greet() method is available in class Y, hence no compilation error for Line n2.

At Line n3, x is casted to Z and compiler checks whether greet() method is available in class Z or not. As greet() method is available in class Z, hence no compilation error for Line n3.

There is no compilation error in the given code it compiles successfully.

Variable x refers to an instance of Z class and at Line n1, n2 and n3 same instance is being used. Which overriding method to invoke is decided at runtime based on the instance.

At runtime, all three statements, at Line n1, Line n2 and Line n3 would invoke the greet() method of Z class, which would print Good Night! three times on to the console.

13. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Base {
int id = 1000; //Line n1

Base() {
Base(); //Line n2
}

void Base() { //Line n3
System.out.println(++id); //Line n4
}
}

class Derived extends Base {
int id = 2000; //Line n5

Derived() {} //Line n6

void Base() { //Line n7
System.out.println(--id); //Line n8
}
}

public class Test {
public static void main(String[] args) {
Base base = new Derived(); //Line n9
}
}
```

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

- ○ 1000
- ○ 1001
- ○ 999
- ○ 2000
- ○ 1999
- ○ -1

Incorrect

SCP76394:

Method can have same name as that of the Class. Hence, void Base() is a valid method declaration in Base class.

Line n2 invokes the Base() method and not the constructor.

Subclass overrides the methods of superclass but it hides the variables of superclass.

Line n5 hides the variable created at Line n1, there is no rules related to hiding (type and access modifier can be changed).

Line n7 correctly overrides the Base() method of class Base.

Compiler adds super(); as the 1st statement inside the no-argument constructor of Base class and Derived class.

There is no compilation error, so let's check the execution.

new Derived() at Line n9 invokes the constructor of Derived class, at this point instance variable id is declared and 0 is assigned to it. In fact, instance variable id of Base class is also declared and 0 is assigned to it. Compiler added super(); as the first statement inside this constructor, hence control goes to the no-argument constructor of Base class.

Compiler added super(); as the first statement inside this constructor as well, hence it invokes the no-argument constructor of the Object class. No-argument constructor of Object class finishes its execution and control goes back to the constructor of Base class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 1000 is assigned to variable id of Base class.

Line n2 is executed next, Base() method defined in Derived class is executed. Which overriding method to invoke, is decided at runtime based on the instance. Instance is of Derived class (because of Line n9), hence control starts executing Base() method of Derived class.

Line n8 is executed next, Derived class hides the id variable of Base class and that is why at Line n8, id points to variable created at Line n5. This id variable still stores the value 0 as Base class's constructor has not finishes its execution.

value of id is decremented by 1, so id becomes -1 and -1 is printed on to the console. Base() method finishes its execution and control goes back to Line n2. No-argument constructor of Base class finishes its execution and control goes back to the constructor of Derived class. Before it starts executing remaining statements inside the constructor, instance variable assignment statement (if available) are executed. This means 2000 is assigned to variable id of Derived class.

No-argument constructor of Derived class finishes its execution and control goes back to Line n9. main(String []) method finishes its execution and program terminates successfully.

Hence, output is -1.

14. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Lock {
public void open() {
System.out.println("LOCK-OPEN");
}
}

class Padlock extends Lock {
public void open() {
System.out.println("PADLOCK-OPEN");
}
}

class DigitalPadlock extends Padlock {
public void open() {
/*INSERT*/
}
}

public class Test {
public static void main(String[] args) {
Lock lock = new DigitalPadlock();
lock.open();
}
}
```

Which of the following options, if used to replace /*INSERT*/, will compile successfully and on execution will print LOCK-OPEN on to the console?

- super.open();
- super.super.open();
- ((Lock)super).open();
- (Lock)super.open();
- **None of the other options**

Incorrect

SCP23426:

super.open(); => Using super keyword, you can access methods and variables of immediate parent class, hence if you replace /*INSERT*/ with `super.open();`, then open() method of Padlock class will be invoked.

super.super.open(); => super.super is not allowed in java, it causes compilation error.

((Lock)super).open(); => Not possible to cast super keyword in java, it causes compilation error.

(Lock)super.open(); => super.open(); will be evaluated first as dot (.) operator has higher precedence than cast. super.open(); returns void and hence it cannot be casted to Lock. It also causes compilation error.

In fact, it is not possible to directly reach to 2 levels, super keyword allows to access methods and variables of immediate parent class only (just 1 level up). Hence, correct answer is: 'None of the other options'

15. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Super {
final int NUM = -1; //Line n1
}

class Sub extends Super {
/*INSERT*/
}

public class Test {
public static void main(String[] args) {
Sub obj = new Sub();
obj.NUM = 200; //Line n2
System.out.println(obj.NUM); //Line n3
}
}
```

Above code causes compilation error, which modifications, done independently, enable the code to compile and on execution print 200 on to the console?

Select 4 options.

- [x] Remove final modifier from Line n1
- [x] Replace /*INSERT*/ with byte NUM;
- [x] Replace /*INSERT*/ with short NUM;
- [x] Replace /*INSERT*/ with int NUM;
- [ ] Replace /*INSERT*/ with float NUM;
- [x] Replace /*INSERT*/ with Object NUM;

Incorrect

SCP43019:

Variable NUM is declared in Super class and class Sub extends Super, hence NUM can be accessed by using obj.NUM.

But as NUM Is final, hence it cannot be reassigned, therefore Line n2 causes compilation error. Let's check all the options one by one:

Remove final modifier from Line n1

? Valid option and in this case output is 200.

Replace /*INSERT*/ with byte NUM;

? In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as byte range is from -128 to 127 and 200 is out of range value.

Replace /*INSERT*/ with short NUM;

? In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to short type. In this case output is 200.

Replace /*INSERT*/ with int NUM;

? In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to int type. In this case output is 200.

Replace /*INSERT*/ with float NUM;

? In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to float type. But output in this case will be 200.0 and not 200.

Replace /*INSERT*/ with double NUM;

? In this case, class Sub hides the variable NUM of Super class and 200 can be easily assigned to double type. But output in this case will be 200.0 and not 200.

Replace /*INSERT*/ with boolean NUM;

? In this case, class Sub hides the variable NUM of Super class but Line n2 will still not compile as boolean type in java allows 2 values true and false. 200 is not compatible with boolean type.

Replace /*INSERT*/ with Object NUM;

? In this case, class Sub hides the variable NUM of Super class and at Line n2, value 200 is boxed to Integer, which is then assigned to obj.NUM. So, obj.NUM refers to an instance of Integer class. Line n3 invokes toString() method of Integer class and hence 200 is printed on to the console.

16. Question

Given below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
private static String print(String... args) {
return String.join("-", args); //Line n1
}

private static Object print(Object... args) {
String str = "";
for(Object obj : args) {
if(obj instanceof String) { //Line n2
str += (String) obj; //Line n3
}
}
return str; //Line n4
}

public static void main(String... args) {
Object obj1 = new String("SPORT"); //Line n5
Object obj2 = new String("MAD"); //Line n6
System.out.println(print(obj1, obj2)); //Line n7
}
}
```

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

- Line n1 causes compilation error
- Line n2 causes compilation error
- Line n3 causes compilation error
- Line n7 causes compilation error
- **Code compiles successfully and on execution prints SPORTMAD on to the console**
- Code compiles successfully and on execution prints SPORT-MAD on to the console

Incorrect

SCP65269:

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

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

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

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

Based on above points Line n1 compiles successfully.

As String class extends Object class, hence instanceof check at Line n2 compiles successfully.

Syntax at Line n3 is: str += (String) obj; => str = str + (String) obj; No issues with Line n3 as well.

Super class reference variable can easily refer to an instance of sub class and hence syntax of Line n5 and Line n6 is also valid.

Which overloaded method to invoke, is decided at compile time. As print(...) method is overloaded in Test class, print(obj1, obj2) tags to print(Object...) because obj1 and obj2 are of Object type.

Given code compiles successfully and on execution prints SPORTMAD on to the console.

17. Question

Consider below code snippet:

```
package com.skillcertpro.ocp;

public class Test {
String testNo;
String desc;
/*
Other codes...
*/
}
```

Which of the options are correct so that instance variables 'testNo' and 'desc' are accessible only within 'com.skillcertpro.ocp' package?

- **No changes are necessary**
- Change the instance variable declarations to:
- private String testNo; private String desc; Change the instance variable declarations to:
- protected String testNo; protected String desc; Change the instance variable declarations to:
- public String testNo; public String desc;

Correct

SCP22559:

As member variables 'testNo' and 'desc' are declared with no explicit access specifier, this means these variables have package scope, hence these variables are accessible only to classes within the same package. Hence, no changes are necessary.

If you use private, then instance variables will not be accessible to any other classes, even within the same package.

If you use protected, then instance variables will be accessible to the subclasses outside 'com.skillcertpro.ocp' package.

If you use public, then instance variables will be accessible to all the classes.

18. Question

Consider incomplete code of M.java file

```
class M {
}

_____ class N {
}
```

Following options are available to fill the above blank:

1. public

2. private

3. protected

4. final

5. abstract

How many above options can be used to fill above blank (separately and not together) such that there is no compilation error?

- Only one option
- **Only two options**
- Only three options
- Only four options
- All five options

Incorrect

SCP72806:

Top-level class can use only two access modifiers [public and default(don't specify anything)]. private and protected cannot be used.

As file name is M.java, hence class N cannot be public.

Top-level class can be final, hence it is a correct option.

Top-level class can be abstract and hence it is also a correct option.

Question **19** of **55**

19. Question

Consider below code of Wall.java file:

```
package com.skillcertpro.ocp;

public class Wall {
public static void main(String args[]) {
double area = 5.7;
String color;
if (area < 7) color = "BLUE"; System.out.println(color); } } What will be the result of compiling and executing Wall class?
```

- BLUE
- An exception is thrown at runtime
- **Compilation error**

Incorrect

SCP47829:

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

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

There is no else block available as well.

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

Question **20** of **55**

20. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
public static void main(String[] args) {
double price = 90000;
String model;
if(price > 100000) {
model = "Tesla Model X";
} else if(price <= 100000) { model = "Tesla Model S"; } System.out.println(model); } } What will be the result of compiling and executing Test class?
```

- Tesla Model X
- Tesla Model S
- **Compilation Error**

Incorrect

SCP22549:

In this case "if – else if" block is used and not "if – else" block.

90000 is assigned to variable 'price' but you can assign parameter value or call some method returning double value, such as:

`double price = currentTemp();`.

In these cases, compiler will not know the exact value until runtime, hence Java Compiler is not sure which boolean expression will be evaluated to true and so variable model may not be initialized.

Usage of LOCAL variable, 'model' without initialization causes compilation error. Hence, `System.out.println(model);` statement causes compilation error.

21. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
public static void main(String[] args) {
short [] args = new short[]{50, 50};
args[0] = 5;
args[1] = 10;
System.out.println("[" + args[0] + ", " + args[1] + "]");
}
}
```

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

- **Compilation error**
- An exception is thrown at runtime
- [50, 50]
- [5, 10]

Incorrect

SCP68798:

main method's parameter variable name is "args" and it is LOCAL to main method. So, same name "args" can't be used directly within the curly brackets of main method.
short [] args = new short[]{50, 50}; causes compilation error for using same name for local variable.

22. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
static String msg; //Line n1
public static void main(String[] args) {
String msg; //Line n2
if(args.length > 0) {
msg = args[0]; //Line n3
}
System.out.println(msg); //Line n4
}
}
```

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

- Line n1 causes compilation failure
- Line n2 causes compilation failure
- An exception is thrown at runtime by Line n3
- **Line n4 causes compilation failure**

Incorrect

SCP31491:

Line n2 code shadows the variable at Line n1. 'msg' variable created at Line n2 is a local variable and should be initialized before it is used.

Initialization code is inside if-block, so compiler is not sure about the initialization of 'msg' variable initialization. Hence, Line n4 causes compilation failure.

23. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
static String str = "KEEP IT "; //Line n1
public static void main(String[] args) {
String str = str + "SIMPLE"; //Line n2
System.out.println(str);
}
}
```

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

- KEEP IT
- KEEP IT SIMPLE
- SIMPLE
- **Compilation error**

Incorrect

SCP42141:

At Line n2, local variable 'str' shadows the static variable 'str' created at Line n1. Hence, for the expression `str + "SIMPLE"`, Java compiler complains as local variable 'str' is not initialized.

24. Question

Given code of TestRectangle.java file:

```
package com.skillcertpro.ocp;

class Rectangle {
private int height;
private int width;

public Rectangle(int height, int width) {
this.height = height;
this.width = width;
}

public int getHeight() {
return height;
}

public int getWidth() {
return width;
}
}

public class TestRectangle {
public static void main(String[] args) {
private int i = 100;
private int j = 200;
Rectangle rect = new Rectangle(i, j);
System.out.println(rect.getHeight() + ", " + rect.getWidth());
}
}
```

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

- 100, 200
- 200, 100
- **Compilation Error**
- 0, 0

Incorrect

SCP68793:

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

Only final modifier can be used with local variables.

25. Question

Consider below code of TestBook.java file:

```
package com.skillcertpro.ocp;

class Book {
private String name;
private String author;

Book() {}

Book(String name, String author) {
name = name;
author = author;
}

String getName() {
return name;
}

String getAuthor() {
return author;
}
}

public class TestBook {
public static void main(String[] args) {
private Book book = new Book("Head First Java", "Kathy Sierra");
System.out.println(book.getName());
System.out.println(book.getAuthor());
}
}
```

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

- Compilation error in Book class
- **Compilation error in TestBook class**
- Head First Java
- Kathy Sierra

Incorrect

SCP88383:

Variable 'book' in main(String[]) method of TestBook class cannot be declared private as it is a local variable. Hence, there is a compilation error in TestBook class.

Only final modifier can be used with local variables.

Question **26** of **55**

26. Question

Choose the option that meets the following specification:

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

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

- public class Clock { public String model; }
- public class Clock { public String model; public String getModel() { return model; } public void setModel(String val) { model = val; } }
- **public class Clock { private String model; public String getModel() { return model; } public void setModel(String val) { model = val; } }**
- public class Clock { public String model; private String getModel() { return model; } private void setModel(String val) { model = val; } }

Incorrect

SCP65282:

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

Out of the given 4 options, below option provides a well encapsulated Clock class:

```
public class Clock {
   private String model;
   public String getModel() { return model; }
   public void setModel(String val) { model = val; }
}
```

Question 27 of 55

27. Question

Consider the following class:

```
public class Employee {
public int passportNo; //line 2
}
```

Which of the following is the correct way to make the variable 'passportNo' read only for any other class?

- Make 'passportNo' private
- **Make 'passportNo' private and provide a public method getPassportNo() which will return its value**
- Make 'passportNo' static and provide a public static method getPassportNo() which will return its value
- Remove 'public' from the 'passportNo' declaration. i.e., change line 2 to int passportNo;

Incorrect

SCP70626:

'passportNo' should be read-only for any other class.

This means make 'passportNo' private and provide public getter method. Don't provide public setter as then 'passportNo' will be read-write property.

If passportNo is declared with default scope, then other classes in the same package will be able to access passportNo for read-write operation.

Question 28 of 55

28. Question

_____ uses access modifiers to protect variables and hide them within a class.

Which of the following options accurately fill in the blank above?

- Polymorphism
- Inheritance
- **Encapsulation**
- Abstraction

Correct

SCP10576:

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

29. Question

Consider below code of Circle.java file:

```
package com.skillcertpro.ocp;

public class Circle {
private double radius;

public Circle(double radius) {
this.radius = radius;
}

public double getArea() {
return Math.PI * radius * radius;
}
}
```

User must be allowed to read and change the value of radius field. What needs to be done so that all the classes can read/change the value of radius field and Circle class is well encapsulated as well?

- Nothing needs to be done
- Change radius declaration from `private double radius;` to `double radius;`
- Change radius declaration from `private double radius;` to `protected double radius;`
- Change radius declaration from `private double radius;` to `public double radius;`
- Add below 2 methods in Circle class: public double getRadius() { return radius; } public void setRadius(double radius) { this.radius = radius; }
- Add below 2 methods in Circle class: protected double getRadius() { return radius; } protected void setRadius(double radius) { this.radius = radius; }

Incorrect

SCP80405:

Circle class needs to be well encapsulated, this means that instance variable radius must be declared with private access modifier and getter/setter methods must be public, so that value in radius variable can be read/changed by other classes.

Out of the given options, below option is correct:

Add below 2 methods in Circle class:

```
public double getRadius() {
    return radius;
}

public void setRadius(double radius) {
    this.radius = radius;
}
```

Question **30** of **55**

30. Question

Consider the code of Greet.java file:

```
package com.skillcertpro.ocp;

public final class Greet {
private String msg;
public Greet(String msg) {
this.msg = msg;
}

public String getMsg() {
return msg;
}

public void setMsg(String msg) {
this.msg = msg;
}
}
```

Is Greet class an immutable class?

- Yes
- No

Incorrect

SCP79899:

Immutable class should have private fields and no setters.

In this case it is possible to change the msg after an instance of Greet class is created. Check below code:

```
public class Test {
public static void main(String[] args) {
Greet greet = new Greet("Hello"); //msg refers to "Hello"
greet.setMsg("Welcome"); //msg refers to "Welcome"
System.out.println(greet.getMsg()); //Prints "Welcome" on to the console.
}
}
```

To make Greet class immutable, delete the setter and add modifier 'final' for variable msg.

```
final class Greet {
private final String msg;
public Greet(String msg) {
this.msg = msg;
}

public String getMsg() {
return msg;
}
}
```

Once value is assigned to msg variable in the constructor, it cannot be changed later.

Question **31** of **55**

31. Question

super keyword in java is used to:

- refer to the static variable of the class
- refer to the static method of the class
- refer to the object of current class
- **refer to the object of parent class**

Incorrect

SCP19392:

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

32. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class A {
A() {
this(1);
System.out.println("M");
}

A(int i) {
System.out.println("N");
}
}

class B extends A {

}

public class Test {
public static void main(String[] args) {
new B();
}
}
```

Correct

SCP19394:

Default constructor added by Java compiler in B class is:

```
B() {
super();
}
```

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

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

N is printed first and after that M is printed.

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

- M N
- **N M**
- M
- N

33. Question

Consider below code of Test.java file:

```
class Super {
Super(int i) {
System.out.println(100);
}
}

class Sub extends Super {
Sub() {
System.out.println(200);
}
}

public class Test {
public static void main(String[] args) {
new Sub();
}
}
```

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

- ◉ 200 200
- ○ 100 100
- ○ 200
- ○ Compilation Error in Super class
- ◉ **Compilation Error in Sub class**

Incorrect

SCP56336:

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

```
Super(int i) {
super();
System.out.println(100);
}

and
Sub() {
super();
System.out.println(200);
}
```

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

But no-argument constructor is not available in Super class, hence calling super(); from Sub class's constructor causes compilation error.

34. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class Super {
Super() {
System.out.print("Reach");
}
}

class Sub extends Super {
Sub() {
Super();
System.out.print("Out");
}
}

public class Test {
public static void main(String[] args) {
new Sub();
}
}
```

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

- ○ Compilation Error in Super class
- ○ Compilation Error in Sub class
- ○ Compilation Error in Test class
- ○ It prints ReachOut on to the console
- ○ It prints OutReach on to the console

Incorrect

SCP54156:

Parent (Super) class constructor is invoked by `super();` (all letters in lowercase) from within the constructor of subclass.

First statement inside no-argument constructor of Sub class is: `Super();` (Letter 'S' is in uppercase) and hence it causes compilation error.

35. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class Super {
void Super() {
System.out.print("KEEP_");
}
}

class Base extends Super {
Base() {
Super();
System.out.print("GOING_");
}
}

public class Test {
public static void main(String[] args) {
new Base();
}
}
```

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

- Compilation Error in Super class
- Compilation Error in Base class
- Compilation Error in Test class
- **It prints KEEP_GOING_ on to the console**
- It prints GOING_KEEP_ on to the console
- It prints KEEP_KEEP_GOING_ on to the console

Incorrect

SCP26923:

Super class defines a method with name Super() but not any constructor. Hence compiler adds below default constructor in Super class:

```
Super() {
   super();
}
```

Class Super extends from Object class and Object class has no-argument constructor, which is called by the super(); statement in above default constructor.

Java compiler also adds `super();` as the first statement inside the no-argument constructor of Base class:

```
Base() {
   super();
   Super();
   System.out.print("GOING_");
}
```

As Base extends Super and both the classes are in the same package, hence `super();` invokes the no-argument constructor of Super class and `Super();` invokes the Super() method of Super class. Base class inherits the Super() method of Super class.

No compilation error in any of the classes.

On executing Test class, main(String[]) is invoked, which executes `new Base();` statement.

No-argument constructor of Base class is invoked, which executes `super();`, hence no-argument constructor of Super class is invoked.

Next, `Super();` is executed and this invokes the Super() method of Super class and hence KEEP_ is printed on to the console.

After that, `System.out.print("GOING_");` is executed and GOING_ is printed on to the console.

main(String []) method finishes its execution and program terminates successfully after printing KEEP_GOING_ on to the console.

36. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class MyClass {
MyClass() {
System.out.println(101);
}
}

class MySubClass extends MyClass {
final MySubClass() {
System.out.println(202);
}
}
public class Test {
public static void main(String[] args) {
System.out.println(new MySubClass());
}
}
```

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

- **Compilation error**
- 101
- 202
- 202
- 202 101
- 101

Correct

SCP54151:

Constructors cannot use final, abstract or static modifiers. As no-argument constructor of MySubClass uses final modifier, therefore it causes compilation error.

37. Question

Consider below codes of 3 java files:

```
//Animal.java
package a;

public class Animal {
Animal() {
System.out.print("ANIMAL-");
}
}

//Dog.java
package d;

import a.Animal;

public class Dog extends Animal {
public Dog() {
System.out.print("DOG");
}
}

//Test.java
package com.skillcertpro.ocp;

import d.Dog;

public class Test {
public static void main(String[] args) {
new Dog();
}
}
```

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

- Compilation error in Animal.java file
- **Compilation error in Dog.java file**
- Compilation error in Test.java file
- It executes successfully and prints ANIMAL-DOG on to the console
- It executes successfully and prints DOG on to the console
- It executes successfully but nothing is printed on to the console

Correct

SCP45203:

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

```
Animal() {
super();
System.out.print("ANIMAL-");
}
```

and

```
public Dog() {
super();
System.out.print("DOG");
}
```

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

Animal class's constructor has package scope, which means it is accessible to all the classes declared in package 'a'. But Dog class is declared in package 'b' and hence `super();` statement inside Dog class's constructor causes compilation error as no-argument constructor of Animal class is not visible.

There is no compilation error in Test.java file as Dog class's constructor is public and therefore `new Dog();` compiles successfully.

38. Question

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

- constant
- super
- **final**
- class
- const

Incorrect

SCP45688:

Class declared as final cannot be inherited. Examples are: String, Integer, System etc.

39. Question

Consider below code of Test.java file:

```
//Test.java
package com.skillcertpro.ocp;

class Parent {
int i = 10;
Parent(int i) {
super();
this.i = i;
}
}

class Child extends Parent {
int j = 20;

Child(int j) {
super(0);
this.j = j;
}

Child(int i, int j) {
super(i);
this(j);
}

}

public class Test {
public static void main(String[] args) {
Child child = new Child(1000, 2000);
System.out.println(child.i + ":" + child.j);
}
}
```

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

- Compilation error in Parent(int) constructor
- Compilation error in Child(int) constructor
- **Compilation error in Child(int, int) Constructor**
- Compilation error in Test class
- It executes successfully and prints 1000:2000 on to the console
- It executes successfully and prints 1000:0 on to the console

Incorrect

SCP46536:

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

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

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

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

40. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class PenDrive {
int capacity;
PenDrive(int capacity) {
this.capacity = capacity;
}
}

class OTG extends PenDrive {
String type;
String make;
OTG(int capacity, String type) {
/*INSERT-1*/
}
OTG(String make) {
/*INSERT-2*/
this.make = make;
}
}

public class Test {
public static void main(String[] args) {
OTG obj = new OTG(128, "TYPE-C");
System.out.println(obj.capacity + ":" + obj.type);
}
}
```

Currently above code causes compilation error.

Which of the options can successfully print 128:TYPE-C on to the console?

- ○ Replace /*INSERT-1*/ with: super(capacity);
- ◉ Replace /*INSERT-2*/ with: super(128);
- ○ Replace /*INSERT-1*/ with: super.capacity = capacity; this.type = type;
- ○ Replace /*INSERT-2*/ with: super(128);
- ○ Replace /*INSERT-1*/ with: super(capacity); this.type = type;
- ◉ **Replace /*INSERT-2*/ with: super(0);**

Incorrect

SCP48714:

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.

Compiler adds super(); as the first line in OTG's constructor: OTG(int capacity, String type) { super(); } but PenDrive class doesn't have a no-argument constructor and that is why OTG's constructor causes compilation error.

For the same reason, OTG(String make) constructor also causes compilation error.

To correct these compilation errors, parent class constructor should be invoked by using super(int); This would resolve compilation error.

Remember: Constructor call using this(...) or super(...) must be the first statement inside the constructor.

In the main(String[]) method, OTG(int, String) constructor is invoked, which means, we OTG(String) constructor will not be executed. So, to solve the compilation error in OTG(String) constructor, super(0); or super(128); both will work and these will not affect the expected output.

We have to make changes in OTG(int, String) constructor such that on execution, output is 128:TYPE-C.

super(capacity); will only assign value to capacity property, to assign value to type another statement is needed.

this.type = type; must be the 2nd statement.

So, /*INSERT-1*/ must be replaced with:

super(capacity);

this.type = type;

41. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class Document {
int pages;
Document(int pages) {
this.pages = pages;
}
}

class Word extends Document {
String type;
Word(String type) {
super(20); //default pages
/*INSERT-1*/
}

Word(int pages, String type) {
/*INSERT-2*/
super.pages = pages;
}
}

public class Test {
public static void main(String[] args) {
Word obj = new Word(25, "TEXT");
System.out.println(String.join(",", obj.type, obj.pages + ""));
}
}
```

Currently above code causes compilation error.

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

- Replace /*INSERT-1*/ with: this(type);
- Replace /*INSERT-2*/ with: this.type = type;
- Replace /*INSERT-1*/ with: this.type = type;
- **Replace /*INSERT-2*/ with: this(type);**
- Replace /*INSERT-1*/ with: super.type = type;
- Replace /*INSERT-2*/ with: this(type);

Correct

SCP46539:

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.

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

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

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

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

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

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

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

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

Based on above points, `System.out.println(String.join(",", obj.type, obj.pages + ""));` will print TEXT,25 on to the console.

42. Question

Given codes of A.java file:

```
//A.java
package com.skillcertpro.ocp;

public class A {
public int i1 = 1;
protected int i2 = 2;
}
```

and B.java file:

```
//B.java
package com.skillcertpro.ocp.test;

import com.skillcertpro.ocp.A;

public class B extends A {
public void print() {
A obj = new A();
System.out.println(obj.i1); //Line 8
System.out.println(obj.i2); //Line 9
System.out.println(this.i2); //Line 10
System.out.println(super.i2); //Line 11
}

public static void main(String [] args) {
new B().print();
}
}
```

One of the statements inside print() method causes compilation error.

Which of the below solutions will help to resolve compilation error?

- Comment the statement at Line 8
- **Comment the statement at Line 9**
- Comment the statement at Line 10
- Comment the statement at Line 11

Correct

SCP66589:

class A is declared public and defined in com.skillcertpro.ocp package, there is no problem in accessing class A outside com.skillcertpro.ocp package.

class B is defined in com.skillcertpro.ocp.test package, to extend from class A either use import statement `import com.skillcertpro.ocp.A;` or fully qualified name of the class 'com.skillcertpro.ocp.A'. No issues with this class definition as well.

Variable i1 is declared public in class A, so Line 8 doesn't cause any compilation error. Variable i2 is declared protected so it can only be accessed in subclass using using inheritance but not using object reference variable. obj.i2 causes compilation error.

class B inherits variable i2 from class A, so inside class B it can be accessed by using either this or super. Line 10 and Line 11 don't cause any compilation error.

43. Question

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

- class Dog {}
- abstract class Cat {}
- **final class Electronics {}**
- private class Car {}

Correct

SCP30191:

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

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

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

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

44. Question

Consider below code of Test.java file:

```
//Test.java
package com.skillcertpro.ocp;

abstract class Animal {
private String name;

Animal(String name) {
this.name = name;
}

public String getName() {
return name;
}
}

class Dog extends Animal {
private String breed;

Dog(String breed) {
this.breed = breed;
}

Dog(String name, String breed) {
super(name);
this.breed = breed;
}

public String getBreed() {
return breed;
}
}

public class Test {
public static void main(String[] args) {
Dog dog1 = new Dog("Beagle");
Dog dog2 = new Dog("Bubbly", "Poodle");
System.out.println(dog1.getName() + ":" + dog1.getBreed() + ":" +
dog2.getName() + ":" + dog2.getBreed());
}
}
```

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

- ○ Compilation error for Test Class
- ◉ Compilation error for Animal(String) constructor
- ○ **Compilation error for Dog(String) constructor**
- ○ Compilation error for Dog(String, String) constructor
- ○ null:Beagle:Bubbly:Poodle
- ○ :Beagle:Bubbly:Poodle

Incorrect

SCP88376:

abstract class can have constructors and it also possible to have abstract class without any abstract method. So, there is no issue with Animal class.

Java compiler adds super(); as the first statement inside constructor, if call to another constructor using this(...) or super(...) is not available.

Inside Animal class Constructor, compiler adds super(); => Animal(String name) { super(); this.name = name; }, super() in this case invokes the no-argument constructor of Object class and hence no compilation error here.

Compiler changes Dog(String) constructor to: Dog(String breed) { super(); this.breed = breed; }. No-argument constructor is not available in Animal class and as another constructor is provided, java compiler doesn't add default constructor. Hence Dog(String) constructor causes compilation error.

There is no issue with Dog(String, String) constructor and code in Test class is also fine.

45. Question

Consider below code of TestBaseDerived.java file:

```
//TestBaseDerived.java
package com.skillcertpro.ocp;

class Base {
protected void m1() {
System.out.println("Base: m1()");
}
}

class Derived extends Base {
void m1() {
System.out.println("Derived: m1()");
}
}

public class TestBaseDerived {
public static void main(String[] args) {
Base b = new Derived();
b.m1();
}
}
```

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

- ○ Base: m1()
- ◉ Derived: m1()
- ○ Base: m1()
- ○ Derived: m1()
- ◉ **None of the other options**

Incorrect

SCP48707:

Derived class overrides method m1() of Base class.

Access modifier of method m1() in Base class is protected, so overriding method can use protected or public. But overriding method in this case used default modifier and hence there is compilation error.

46. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class Vehicle {
public int getRegistrationNumber() {
return 1;
}
}

class Car {
public double getRegistrationNumber() {
return 2;
}
}

public class Test {
public static void main(String[] args) {
Vehicle obj = new Car();
System.out.println(obj.getRegistrationNumber());
}
}
```

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

- ○ 1
- ○ 2
- ○ An exception is thrown at runtime
- ◉ Compilation error in Vehicle class
- ○ Compilation error in Car class
- ◉ **Compilation error in Test class**

Incorrect

SCP10583:

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

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

As Vehicle and Car classes are not related, hence these don't cause any compilation error.

47. Question

Given code of LogHelper.java file:

```
package com.skillcertpro.ocp;

abstract class Helper {
int num = 100;
String operation = null;

protected abstract void help();

void log() {
System.out.println("Helper-log");
}
}

public class LogHelper extends Helper {
private int num = 200;
protected String operation = "LOGGING";

void help() {
System.out.println("LogHelper-help");
}

void log() {
System.out.println("LogHelper-log");
}

public static void main(String [] args) {
new LogHelper().help();
}
}
```

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

Select 3 options.

- [ ] Remove the private modifier from the num variable of LogHelper class
- [x] Remove the protected modifier from the operation variable of LogHelper class
- [x] **Remove the protected modifier from the help() method of Helper class**
- [ ] Add the protected modifier to the log() method of Helper class
- [x] **Add the protected modifier to the help() method of LogHelper class**
- [x] **Add the public modifier to the help() method of LogHelper class**

Incorrect

SCP42179:

Let us first find out the issue:

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

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

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

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

OR

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

OR

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

48. Question

Consider below code fragment:

```
package com.skillcertpro.ocp;

abstract class Food {
protected abstract double getCalories();
}

class JunkFood extends Food {
double getCalories() {
return 200.0;
}
}
```

Which 3 modifications, done independently, enable the code to compile?

- [ ] Make the getCalories() method of Food class public
- [x] **Remove the protected access modifier from the getCalories() method of Food class**
- [ ] Make the getCalories() method of Food class private
- [x] **Make the getCalories() method of JunkFood class protected**
- [x] **Make the getCalories() method of JunkFood class public**
- [ ] Make the getCalories() method of JunkFood class private

Incorrect

SCP63076:

abstract methods cannot be declared with private modifier as abstract methods need to be overridden in child classes.

abstract methods can be declared with either public, protected and package (no access modifier) modifier and hence overriding method cannot be declared which private modifier in the child class. That is why getCalories() method in Food and JunkFood classes cannot be declared private.

Access modifier of overriding method should either be same as the access modifier of overridden method or it should be less restrictive than the access modifier of overridden method. Hence below solutions will work:

1. Remove the protected access modifier from the getCalories() method of Food class: By doing this, both the overridden and overriding methods will have same access modifier (no access modifier)

or

2. Make the getCalories() method of JunkFood class protected: By doing this, both the overridden and overriding methods will have same access modifier (protected)

or

3. Make the getCalories() method of JunkFood class public: By doing this, access modifier of overriding method (which is public) is less restrictive than the access modifier of overridden method (which is protected)

49. Question

Consider below code of Test.java file:

```
//Test.java
package com.skillcertpro.ocp;

class Parent {
public String toString() {
return "Inner ";
}
}

class Child extends Parent {
public String toString() {
return super.toString().concat("Peace!");
}
}

public class Test {
public static void main(String[] args) {
System.out.println(new Child());
}
}
```

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

- (•) Inner
- ( ) Peace!
- (•) **Inner Peace!**
- ( ) Compilation error

Incorrect

SCP30195:

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

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

50. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

abstract class Log {
abstract long count(); //Line n1
abstract Object get(); //Line n2
}

class CommunicationLog extends Log {
int count() { //Line n3
return 100;
}

String get() { //Line n4
return "COM-LOG";
}
}

public class Test {
public static void main(String[] args) {
Log log = new CommunicationLog(); //Line n5
System.out.print(log.count());
System.out.print(log.get());
}
}
```

Which of the following statement is correct?

- Line n3 causes compilation error
- Line n4 causes compilation error
- Line n5 causes compilation error
- Given code compiles successfully and on execution prints 100COM-LOG on to the console

Incorrect

SCP32374:

CommunicationLog class overrides count() and get() methods of Log class.

There are 2 rules related to return types:

1. If return type of overridden method is of primitive type, then overriding method should use same primitive type.

2. If return type of overridden method is of reference type, then overriding method can use same reference type or its sub-type (also known as covariant return type).

count() method at Line n1 returns long but overriding method at Line n3 returns int and that is why Line n3 causes compilation error.

get() method at Line n2 returns Object but overriding method at Line n4 returns String. String is a subclass of Object, so it is a case of covariant return type and hence allowed. Line n4 compiles successfully.

51. Question

Consider below code of Test.java file:

```
public class Test {
public static void main(String[] args) {
P p = new R(); //Line n1
System.out.println(p.compute("Go")); //Line n2
}
}

class P {
String compute(String str) {
return str.repeat(3);
}
}

class Q extends P {
String compute(String str) {
return super.compute(str.toLowerCase());
}
}

class R extends Q {
String compute(String str) {
return super.compute(str.replace('o', 'O'));
//2nd argument is uppercase O
}
}
```

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

- [x] gogogo
- [ ] GoGoGo
- [x] GOGOGO
- [ ] Go
- [ ] GO
- [ ] go

Incorrect

SCP63078:

Class Q correctly overrides the compute(String) method of P class and class R correctly overrides the compute(String) method of Q class. Keyword super is used to invoke the method of parent class.

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".

At Line n1, reference variable 'p' refers to an instance of class R, hence p.compute("Go") invokes the compute(String) method of R class.

return super.compute(str.replace('o', 'O')); => return super.compute("Go".replace('o', 'O')); => return super.compute("GO");

It invokes the compute(String) method of Parent class, which is Q.

=> return super.compute(str.toLowerCase()); => return super.compute("GO".toLowerCase()); => return super.compute("go");

It invokes the compute(String) method of Parent class, which is P.

=> return str.repeat(3); => return "go".repeat(3); => return "gogogo";

Control goes back to compute(String) method of Q and to the compute(String) method of R, which returns "gogogo".

Line n2 prints gogogo on to the console.

52. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class X {
void A() {
System.out.print("A");
}
}

class Y extends X {
void A() {
System.out.print("A-");
}

void B() {
System.out.print("B-");
}

void C() {
System.out.print("C-");
}
}

public class Test {
public static void main(String[] args) {
X obj = new Y(); //Line n1
obj.A(); //Line n2
obj.B(); //Line n3
obj.C(); //Line n4
}
}
```

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

- A-B-C-
- AB-C-
- Compilation error in class Y
- **Compilation error in class Test**

Incorrect

SCP31043:

Class Y correctly extends class X and it overrides method A() and provides two new methods B() and C().

At Line n1, obj is of X type and therefore obj.B(); and obj.C(); cause compilation error as these methods are not defined in class X.

53. Question

Given:

```
class A{}
class B extends A{}
```

Which of the following is not a valid statement based on above code?

- **B obj1 = new A();**
- A obj2 = new B();
- B obj3 = new B();
- A obj4 = new A();

Correct

SCP35402:

B obj1 = new A(); => child class reference cannot refer to parent class object. This causes compilation error.

A obj2 = new B(); => parent class reference can refer to child class object. This is Polymorphism.

B obj3 = new B(); => No issues at all.

A obj4 = new A(); => No issues at all.

54. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

class M { }
class N extends M { }
class O extends N { }
class P extends O { }

public class Test {
public static void main(String args []) {
M obj = new O();
if(obj instanceof M)
System.out.print("M");
if(obj instanceof N)
System.out.print("N");
if(obj instanceof O)
System.out.print("O");
if(obj instanceof P)
System.out.print("P");
}
}
```

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

- **MNO**
- MNP
- NOP
- MOP
- MNOP
- O

Incorrect

SCP38913:

M

^

N

^

O [obj refers to instance of O class]

^

P

obj instanceof M => true, hence "M" is printed on to the console.

obj instanceof N => true, hence "N" is printed on to the console.

obj instanceof O => true, hence "O" is printed on to the console.

but

obj instanceof P => false, "P" is not printed.

Output is: MNO

55. Question

Consider below codes of 4 java files:

```
//A.java
package com.skillcertpro.ocp;

public class A {
public void print() {
System.out.println("A");
}
}

//B.java
package com.skillcertpro.ocp;

public class B extends A {
public void print() {
System.out.println("B");
}
}

//C.java
package com.skillcertpro.ocp;

public class C extends A {
public void print() {
System.out.println("C");
}
}

//Test.java
package com.skillcertpro.ocp.test;

import com.skillcertpro.ocp.*;

public class Test {
public static void main(String[] args) {
A obj1 = new C();
A obj2 = new B();
C obj3 = (C)obj1;
C obj4 = (C)obj2;
obj3.print();
}
}
```

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

- It executes successfully and prints A on to the console
- It executes successfully and prints B on to the console
- It executes successfully and prints C on to the console
- Compilation error
- **An exception is thrown at runtime**

Incorrect

SCP36738:

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

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

Let's check the code inside main method.

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

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

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

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

1. Question

Given code:

```
package com.skillcertpro.ocp;

class A {
public void print(String name) {
class B {
B() {
System.out.println(name); //Line n1
}
}
}
B obj = new B(); //Line n2
}

public class Test {
public static void main(String[] args) {
new A().print("OCP"); //Line n3
}
}
```

Incorrect

SCP14065:

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.

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

- OCP
- Compilation error at Line n1
- Compilation error at Line n2
- Compilation error at Line n3

2. Question

Given code:

```
package com.skillcertpro.ocp;

class Outer {
private String msg = "A";
public void print() {
final String msg = "B";
class Inner {
public void print() {
System.out.println(this.msg);
}
}
Inner obj = new Inner();
obj.print();
}
}

public class Test {
public static void main(String[] args) {
new Outer().print();
}
}
```

Correct

SCP62588:

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.

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

- (●) Compilation error
- ( ) A
- ( ) B
- ( ) Exception is thrown at runtime

3. Question

Given code:

```
package com.Stellakhattr.ocp;

class Message {
public void printMessage() {
System.out.println("Hello!");
}
}

public class Test {
public static void main(String[] args) {
Message msg = new Message() {}; //Line n1
msg.printMessage(); //Line n2
}
}
```

Incorrect

SCP20738:

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.

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

- Compilation error at Line n1
- NullPointerException is thrown by Line n2
- **Hello!**
- HELLO!

4. Question

Given code:

```
package com.skillcertpro.ocp;

class Greet {
public void sayHello() {
System.out.println("Hello!");
}
}

public class Test {
public static void main(String[] args) {
Greet obj = new Greet() {
public void SayHello() {
System.out.println("HELLO!");
}
};
obj.sayHello();
}
}
```

Incorrect

SCP54504:

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.

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

- Compilation error
- Runtime error
- **Hello!**
- HELLO!

5. Question

Given code:

```
package com.skillcertpro.ocp;

class Logger {
public void log() {
System.out.println("GO FOR IT");
}
}

public class Test {
public static void main(String[] args) {
Logger obj = new Logger() {
public void Log() {
System.out.println("LET IT BE");
}
};
obj.Log();
}
}
```

Correct

SCP27389:

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.

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

- (●) Compilation error
- ( ) Runtime error
- ( ) GO FOR IT
- ( ) LET IT BE

6. Question

Given code:

```
package com.skillcertpro.ocp;

class Printer {
public void getType() {
System.out.println("LASER");
}
}

public class Test {
public static void main(String[] args) {
/*INSERT*/ obj = new Printer() { //Line n1
public void GetType() { //Line n2
System.out.println("INKJET");
}
};
obj.GetType(); //Line n3
}
}
```

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**
- Only two options
- All three options

Correct

SCP50145:

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.

7. Question

Given code:

```
package com.skillcertpro.ocp;

class Season {
public void printCurrentSeason() {
System.out.println("SUMMER");
}
}

public class Test {
public static void main(String[] args) {
var season = new Season() { //Line n1
@Override public void PrintCurrentSeason() { //Line n2
System.out.println("WINTER");
}
};
season.PrintCurrentSeason(); //Line n3
}
}
```

What is the result?

- ○ Line n1 causes compilation error
- ○ Line n2 causes compilation error
- ◉ 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

Incorrect

SCP10944:

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.

8. Question

Given code:

```
package com.skillcertpro.ocp;

abstract class Greetings {
abstract void greet();
}

public class Test {
public static void main(String[] args) {
var obj = new Greetings() {
@Override public void greet() {
System.out.println("BELIEVE IN YOURSELF");
}
};
obj.greet();
}
}
```

Incorrect

SCP75425:

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.

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

- Compilation error
- NullPointerException
- **BELIEVE IN YOURSELF**
- Nothing is printed on to the console

9. Question

Below is the code of Test.java file:

```
package com.skillcertpro.ocp;

interface Flyable {
void fly();
}

public class Test {
public static void main(String[] args) {
/*INSERT*/
}
}
```

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

Select 2 options.

- [x] Flyable flyable = new Flyable(); Flyable flyable = new Flyable(){};
- [ ] Flyable flyable = new Flyable() { public void fly() { System.out.println("Flying high"); } }
- [ ] Flyable flyable = new Flyable() { public void fly() { System.out.println("Flying high"); } };
- [ ] var flyable = new Flyable() { @Override public void fly() { System.out.println("Flying high"); }
- [ ] public void stop() { System.out.println("Stopping"); } }; flyable.fly(); flyable.stop();

Incorrect

SCP20750:

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.

```
Flyable flyable = new Flyable() {
   public void fly() {
      System.out.println("Flying high");
   }
}
```

? semicolon is missing at the end

```
Flyable flyable = new Flyable() {
   public void fly() {
      System.out.println("Flying high");
   }
};
```

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

```
var flyable = new Flyable() {
   @Override
   public void fly() {
      System.out.println("Flying high");
   }

   public void stop() {
      System.out.println("Stopping");
   }
};
flyable.fly();
flyable.stop();
```

? Local variable Type inference was added in JDK 10.

10. Question

Given code:

```
package com.skillcertpro.ocp;

interface I1 {
void m1();
}

public class Test {
public static void main(String[] args) {
I1 i1 = new I1() {
@Override
public void m1() {
System.out.println(1234);
}
}
i1.m1();
}
}
```

Incorrect

SCP84358:

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

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

- 1234
- No output
- **Compilation error**
- Runtime exception

11. Question

Below is the code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
public static void main(String [] args) {
System.out.println(new Object() {
public String toString() {
return "ANONYMOUS";
}
});
}
}
```

Correct

SCP88717:

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.

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

- (●) ANONYMOUS
- ( ) Some text containing @ symbol
- ( ) Compilation error
- ( ) Runtime exception

12. Question

Below is the code of TestSellable.java file:

```
package com.skillcertpro.ocp;

interface Sellable {
double getPrice();
}

public class TestSellable {
private static void printPrice(Sellable sellable) {
System.out.println(sellable.getPrice());
}

public static void main(String[] args) {
/*INSERT*/
}
}
```

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

Select 3 options.

- [x] printPrice(null);
- [x] printPrice(new Sellable());
- [ ] printPrice(new Sellable() { });
- [ ] printPrice(new Sellable() { @Override public double getPrice() { return 45.34; } });
- [ ] "var obj = new Sellable() { @Override public double getPrice() { return 20.21; } " " public Sellable getThisObject() { return this; } }; " printPrice(obj.getThisObject());

Incorrect

SCP66946:

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.

```
printPrice(new Sellable() {
   @Override
   public double getPrice() {
      return 45.34;
   }
});
```

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

```
var obj = new Sellable() {
   @Override
   public double getPrice() {
      return 20.21;
   }

   public Sellable getThisObject() {
      return this;
   }
};
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.

13. Question

Given code:

```
package com.skillcertpro.ocp;

interface Logger {
Object get();
void log();
}

public class Test {
private static void testLogger(Logger logger) {
logger.log();
}

public static void main(String[] args) {
var obj = new Logger() { //Line n1
@Override
public Logger get() { //Line n2
return this;
}

@Override
public void log() {
System.out.println("WINNERS NEVER QUIT"); //Line n3
}
};
testLogger(obj.get()); //Line n4
}
}
```

Incorrect

SCP71060:

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.

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
- Test class compiles successfully and on execution prints nothing

14. Question

Can an anonymous inner class implement multiple interfaces?

- Yes
- No

Incorrect

SCP40345:

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.

Back | Report Question | Next

15. Question

Below is the code to Test.java file:

```
package com.skillcertpro.ocp;

enum ShapeType {
CIRCLE, SQUARE, RECTANGLE;
}

abstract class Shape {
private ShapeType type = ShapeType.SQUARE; //default ShapeType

Shape(ShapeType type) {
this.type = type;
}

public ShapeType getType() {
return type;
}

abstract void draw();
}

public class Test {
public static void main(String[] args) {
Shape shape = new Shape() {
@Override
void draw() {
System.out.println("Drawing a " + getType());
}
};
shape.draw();
}
}
```

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

- Drawing a CIRCLE
- Drawing a SQUARE
- Drawing a RECTANGLE
- Compilation error

Incorrect

SCP38026:

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(){}

Back Report Question Next

16. Question

Below is the code of Test.java file:

```
package com.skillcertpro.ocp;

class A {
static class B {

}
}

public class Test {
/*INSERT*/
}
```

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

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

Incorrect

SCP42523:

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();'.

17. Question

Below the code of A.java file:

```
package com.skillcertpro.ocp;

public class A {
private static class B {
private void log() {
System.out.println("BE THE CHANGE");
}
}

public static void main(String[] args) {
/*INSERT*/
}
}
```

Incorrect

SCP64776:

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.

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

Select 3 options.

- [x] B obj1 = new B(); obj1.log();
- [x] B obj3 = new A().new B(); obj3.log();
- [x] A.B obj2 = new A.B(); obj2.log();
- [ ] A.B obj4 = new A().new B(); obj4.log();
- [ ] var obj5 = new A.B(); obj5.log();

18. Question

Below is the code of Test.java file:

```
package com.skillcertpro.ocp;

class Outer {
abstract static class Animal { //Line n1
abstract void eat();
}

static class Dog extends Animal { //Line n2
void eat() { //Line n3
System.out.println("DOG EATS BISCUITS");
}
}
}

public class Test {
public static void main(String[] args) {
Outer.Animal animal = new Outer.Dog(); //Line n4
animal.eat();
}
}
```

Incorrect

SCP82180:

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.

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

9. Question

elow is the code of Test.java file:

```
ackage com.skillcertpro.ocp;

lass Outer {
rivate static int i = 10;
rivate int j = 20;

tatic class Inner {
oid add() {
ystem.out.println(i + j);




ublic class Test {
ublic static void main(String[] args) {
Outer.Inner inner = new Outer.Inner();
nner.add();
```

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

- Compilation error in Test class code
- **Compilation error in Inner class code**
- 30
- Exception is thrown at runtime

Incorrect

SCP39359:

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.

20. Question

Below is the code of Test.java file:

```
package com.skillcertpro.ocp;

class Outer {
static class Inner {
static void greetings(String s) {
System.out.println(s);
}
}
}

public class Test {
public static void main(String[] args) {
/*INSERT*/
}
}
```

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

- [x] Outer.Inner inner1 = new Outer().new Inner(); inner1.greetings("HELLO!");
- [x] Outer.Inner inner2 = new Outer.Inner(); inner2.greetings("HELLO!");
- [ ] Outer.Inner.greetings("HELLO!");
- [ ] Inner.greetings("HELLO!");

Incorrect

SCP86544:

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();

21. Question

Will below code compile successfully?

- package com.skillcertpro.ocp;
- class Outer { interface I1 { void m1(); } } Yes
- No

Correct

SCP37179:

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).

22. Question

Will below code compile successfully?

- package com.skillcertpro.ocp;
- interface I1 { void m1();
- interface I2 { void m2(); }
- abstract class A1 { public abstract void m3(); }
- class A2 { public void m4() { System.out.println(4); } } } Yes
- No

Incorrect

SCP14074:

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.

23. Question

Following statement in a Java program compiles successfully:

student.report(course);

What can you say for sure?

- student is the reference variable name
- student is the class name
- **report is the method name**
- course must be of String type

Unattempted

SCP61265:

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

student can be either class name or reference variable name.

Syntax to invoke static method is: Class_Name.method_name(); OR reference_variable_name.method_name();

Syntax to invoke instance method is: reference_variable_name.method_name();

If student represents class_name or refernce_variable_name, then report might be the static method of the class.

If student represents reference_variable_name, then report is the instance method of the class.

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

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

24. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
public static void main(String [] args) {
var arr = new int[] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; //Line n1
String str = process(arr, 3, 8); //Line n2
System.out.println(str);
}

/*INSERT*/
}
```

Line n2 causes compilation error as process method is not found.

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

- private static int[] process(int [] arr, int start, int end) { return null; }
- private static String process(int [] arr, int start, int end) { return null; }
- private static int process(int [] arr, int start, int end) { return null; }
- private static String[] process(int [] arr, int start, int end) { return null; }
- private static String process(var arr, int start, int end) { return null; }
- private static var process(int [] arr, int start, int end) { return null; }

Incorrect

SCP18430:

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.

var type cannot be used as method parameters or method return type.

At Line n1, arr infers to int[] type.

It is clear from Line n2 that, method name should be process, it should be static method, it should accept 3 parameters (int[], int, int) and its return type must be String. Hence below definition is correct:

```
private static String process(int [] arr, int start, int end) {
   return null;
}
```

25. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
public static void main(String [] args) {
String str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println(subtext(str, 3, 8)); //Line n1
}

/*INSERT*/
}
```

Line n1 causes compilation error as subtext method is not found.

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

Select 3 options.

- [x] private static int[] subtext(String text, int start, int end) { return null; }
- [x] private static String subtext(String text, int start, int end) { return null; }
- [x] private static int subtext(String text, int start, int end) { return null; }
- [x] private static String[] subtext(String text, int start, int end) { return null; }
- [ ] private static var subtext(String text, int start, int end) { return null; }
- [ ] private static String subtext(var text, int start, int end) { return null; }

Incorrect

SCP78924:

It is clear from Line n1 that, method name should be subtext, it should be static method, it should accept 3 parameters (String, int, int).

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

In the given options, method specifying int as return type cannot return null as null can't be assigned to primitive type. int subtext(...) would give 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 or package name but it cannot be used as a class or interface name.

var type cannot be used as method parameters or method return type.

26. Question

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

```
//Test.java
package com.skillcertpro.ocp;

class Student {
String name;
int marks;

Student(String name, int marks) {
this.name = name;
this.marks = marks;
}
}

public class Test {
public static void main(String[] args) {
Student student = new Student("James", 25);
int marks = 25;
review(student, marks);
System.out.println(marks + "-" + student.marks);
}

private static void review(Student stud, int marks) {
marks = marks + 10;
stud.marks+=marks;
}
}
```

- 25-25
- 35-25
- 35-60
- 25-60

Incorrect

SCP19766:

In below statements: student

means student inside main method.

On execution of main method: student

–> {"James", 25}, marks
= 25.

On execution of review method: stud –> {"James", 25} (same object referred by student

), marks = 25 (this marks is different from the marks defined in main method). marks = 35 and stud.marks = 60. So at the end of review method: stud –> {"James", 60}, marks = 35.

Control goes back to main method: student

–> {"James", 60}, marks
= 25. Changes done to reference variable are visible in main method but changes done to primitive variable are not reflected in main method.

27. Question

Consider below code of TestSquare.java file:

```
package com.skillcertpro.ocp;

class Square {
int length;
Square sq;

Square(int length) {
this.length = length;
}

void setInner(Square sq) {
this.sq = sq;
}

int getLength() {
return this.length;
}
}

public class TestSquare {
public static void main(String[] args) {
Square sq1 = new Square(10); //Line n1
Square sq2 = new Square(5); //Line n2
sq1.setInner(sq2); //Line n3
System.out.println(sq1.sq.length); //Line n4
}
}
```

Incorrect

SCP75077:

As both the classes: Square and TestSquare are in the same file, hence variables 'length' and 'sq' can be accessed using dot operator. Given code compiles successfully.

Line n1 creates an instance of Square class and 'sq1' refers to it. sq1.length = 10 and sq1.sq = null.

Line n2 creates an instance of Square class and 'sq2' refers to it. sq2.length = 5 and sq2.sq = null.

On execution of Line n3, sq1.sq = sq2.

Line n4: System.out.println(sq1.sq.length); => System.out.println(sq2.length); => Prints 5 on to the console.

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

- It prints 0 on to the console
- It prints 5 on to the console
- It prints 10 on to the console
- It prints null on to the console
- Compilation error
- An exception is thrown at runtime

28. Question

Consider below code fragment:

```
private void emp() {}
```

And the statements:

1. Given code compiles successfully if it is used inside the class named 'emp'

2. Given code compiles successfully if it is used inside the class named 'Emp'

3. Given code compiles successfully if it is used inside the class named 'employee'

4. Given code compiles successfully if it is used inside the class named 'Employee'

5. Given code compiles successfully if it is used inside the class named 'Student'

6. Given code compiles successfully if it is used inside the class named '_emp_'

How many statements are true?

- Only one statement
- Two statements
- Three statements
- Four statements
- Five statements
- All six statements

Incorrect

SCP56315:

`private void emp() {}` is a valid method declaration.

Class name and method name can be same and that is why given method can be declared in any of the given classes: 'emp', 'Emp', 'employee', 'Employee', 'Student' and '_emp_'.

'_emp_' is also a valid Java identifier.

29. Question

Given code of Sport.java file:

```
package com.skillcertpro.ocp;

public class Sport {
String name = "";

}
```

And below definitions:

1.

```
public String Sport() {
return name;
}
```

2

```
public void Sport(String name) {
this.name = name;
}
```

3

```
public void Sport(String... sports) {
name = String.join(",", sports);
}
```

4

```
public Sport() {
name = "";
}
```

5

```
public Sport(String name) {
this.name = name;
}
```

6

```
public Sport(String name1, String name2) {
name = String.join(",", name1, name2);
}
```

How many of the given definitions can be added to Sport class, such that there are no compilation errors?

- (●) All 6 definitions
- ( ) Only 5 definitions
- ( ) Only 4 definitions
- ( ) Only 3 definitions
- ( ) Only 2 definitions
- ( ) Only 1 definition

Correct

SCP56796:

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"

30. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
private static void m(int x) {
System.out.println("INT VERSION");
}

private static void m(char x) {
System.out.println("CHAR VERSION");
}

public static void main(String [] args) {
int i = '5';
m(i);
m('5');
}
}
```

Incorrect

SCP46511:

Method m is overloaded and which overloaded method to invoke, is decided at compile time.

m(i) is tagged to m(int) as i is of int type and m('5') is tagged to m(char) as '5' is char literal.

`m(i);` prints int version on to the console.

`m('5');` prints char version on to the console.

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

- INT VERSION INT VERSION
- CHAR VERSION CHAR VERSION
- INT VERSION CHAR VERSION
- CHAR VERSION INT VERSION
- Compilation error

31. Question

Consider the code of TestStudent.java file:

```
package com.skillcertpro.ocp;

class Student {
String name;
int age;

void Student() {
Student("James", 25); //Line n1
}

void Student(String name, int age) {
this.name = name;
this.age = age;
}
}

public class TestStudent {
public static void main(String[] args) {
Student s = new Student(); //Line n2
System.out.println(s.name + ":" + s.age); //Line n3
}
}
```

Correct

SCP22548:

Methods can have same name as the class. Student() and Student(String, int) are methods and not constructors of the class, note the void return type of these methods.

Line n1 is a valid instance method call from Student() method, hence Line n1 doesn't cause any compilation error.

As no constructors are provided in the Student class, java compiler adds default no-argument constructor. That is why the Line n2 doesn't cause any compilation error.

Default values are assigned to instance variables, hence null is assigned to name and 0 is assigned to age.

Line n3 prints null:0 on to the console.

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

- (●) null:0
- ( ) James:25
- ( ) Line n1 causes compilation error
- ( ) Line n2 causes compilation error
- ( ) An exception is thrown at runtime

32. Question

Consider the code of Test.java file:

```
package com.skillcertpro.ocp;

class Report {
public String generateReport() {
return "CSV";
}

public Object generateReport() {
return "XLSX";
}
}

public class Test {
public static void main(String[] args) {
Report rep = new Report();
String csv = rep.generateReport();
Object xlsx = rep.generateReport();
System.out.println(String.join(":", csv, (String)xlsx));
}
}
```

Correct

SCP20369:

Both the methods of Report class have same signature(name and parameters match).

Having just different return types don't overload the methods and therefore Java compiler complains about duplicate generateReport() methods in Report class.

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

- (●) **Compilation error**
- ( ) An exception is thrown at runtime
- ( ) CSV:XLSX
- ( ) CSV
- ( ) XLSX
- ( ) CSVXLSX

33. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Calculator {
int calculate(int i1, int i2) {
return i1 + i2;
}

double calculate(byte b1, byte b2) {
return b1 % b2;
}
}

public class Test {
public static void main(String[] args) {
byte b = 100;
int i = 20;
System.out.println(new Calculator().calculate(b, i));
}
}
```

Incorrect

SCP23879:

calculate method is correctly overloaded as both the methods have different signature: calculate(int, int) and calculate(byte, byte). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Calculator().calculate(b, i)` tags to `calculate(int, int)` as byte value is implicitly casted to int type.

Given code compiles successfully and on execution prints 120 on to the console.

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

- Compilation error
- An exception is thrown at runtime
- 120
- 120
- 5
- 5

34. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Car {
void speed(Byte val) { //Line n1
System.out.println("DARK"); //Line n2
} //Line n3

void speed(byte... vals) {
System.out.println("LIGHT");
}
}

public class Test {
public static void main(String[] args) {
byte b = 10; //Line n4
new Car().speed(b); //Line n5
}
}
```

Correct

SCP17193:

speed method is correctly overloaded in Car class as both the methods have different signature: speed(Byte) and speed(byte...). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Car().speed(b);` tags to speed(Byte) as boxing is preferred over variable arguments. Code as is prints DARK on to the console.

Variable arguments syntax '...' can be used only for method parameters and not for variable type and type-casting. Hence the option of replacing Line n4 and Line n5 are not correct.

If you delete speed(Byte) method, i.e. Line n1, Line n2 and Line n3, then `new Car().speed(b);` would tag to speed(byte...) method and on execution would print LIGHT on to the console.

Which of the following needs to be done so that LIGHT is printed on to the console?

- No changes are required as given code prints LIGHT on execution
- **Delete Line n1, Line n2 and Line n3**
- Replace Line n4 with byte... b = 10;
- Replace Line n5 with new Car().speed((byte...)b);

Correct

35. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
public static void change(int num) {
num++;
System.out.println(num);
}

public static void main(String[] args) {
int i1 = 1;
Test.change(i1);
System.out.println(i1);
}
}
```

Incorrect

SCP53281:

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

i1 = 1.

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

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

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

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

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

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

- Compilation Error
- 2
- 1
- 2
- 2
- 1

36. Question

Consider below code of TestMessage.java file:

```
package com.skillcertpro.ocp;

class Message {
String msg = "LET IT GO!";

public void print() {
System.out.println(msg);
}
}

public class TestMessage {
public static void change(Message m) { //Line n5
m.msg = "NEVER LOOK BACK!"; //Line n6
}

public static void main(String[] args) {
Message obj = new Message(); //Line n1
obj.print(); //Line n2
change(obj); //Line n3
obj.print(); //Line n4
}
}
```

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

- null NEVER LOOK BACK!
- NEVER LOOK BACK! NEVER LOOK BACK!
- null null
- LET IT GO! NEVER LOOK BACK!
- LET IT GO! LET IT GO!
- Compilation error

Incorrect

SCP24721:

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

Message() {super();}

Line n1 creates an instance of Message class and initializes instance variable 'msg' to "LET IT GO!". Variable 'obj' refers to this instance.

Line n2 prints LET IT GO! on to the console.

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

Line n6, assigns "NEVER LOOK BACK!" to the 'msg' variable of the instance referred by 'm'. As 'obj' and 'm' refer to the same instance, hence obj.msg also refers to "NEVER LOOK BACK!". change(Message) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg refers to "NEVER LOOK BACK!", so this statement prints NEVER LOOK BACK! on to the console.

Hence in the output, you get:

LET IT GO!

NEVER LOOK BACK!

37. Question

Consider below code of AvoidThreats.java file:

```
package com.skillcertpro.ocp;

public class AvoidThreats {
public static void evaluate(Threat t) { //Line n5
t = new Threat(); //Line n6
t.name = "PHISHING"; //Line n7
}

public static void main(String[] args) {
Threat obj = new Threat(); //Line n1
obj.print(); //Line n2
evaluate(obj); //Line n3
obj.print(); //Line n4
}
}

class Threat {
String name = "VIRUS";

public void print() {
System.out.println(name);
}
}
```

Correct

SCP22546:

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

Threat() {super();}

Line n1 creates an instance of Threat class and initializes instance variable 'name' to "VIRUS". Variable 'obj' refers to this instance.

Line n2 prints VIRUS on to the console.

Line n3 invokes evaluate(Threat) method, as it is a static method defined in AvoidThreats class, hence `evaluate(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 't' copies the content of variable 'obj' (which stores the address to Threat instance created at Line n1). 't' also refers to the same instance referred by 'obj'.

On execution of Line n6, another Threat instance is created, its instance variable 'name' refers to "VIRUS" and 't' starts referring to this newly created instance of Threat class. Variable 'obj' of main(String[]) method still refers to the Threat instance created at Line n1. So, 'obj' and 't' now refer to different Threat instances.

Line n7, assigns "PHISHING" to the 'name' variable of the instance referred by 't'. evaluate(Threat) method finishes its execution and control goes back to main(String[]) method.

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

Hence in the output, you get:

VIRUS

VIRUS

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

- VIRUS PHISHING
- PHISHING PHISHING
- **VIRUS VIRUS**
- null VIRUS
- null null
- None of the other options

38. Question

Consider below code of Test.java file:

```
public class Test {
public static void print() {
System.out.println("STATIC METHOD!!!");
}

public static void main(String[] args) {
Test obj = null; //Line n1
obj.print(); //Line n2
}
}
```

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

- An Exception is thrown at runtime
- Compilation error at Line n2
- It prints STATIC METHOD!!! on to the console
- None of the other options

Incorrect

SCP82607:

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

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

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

39. Question

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

```
//Test.java
package com.skillcertpro.ocp;

class Point {
static int x;
int y, z;

public String toString() {
return "Point(" + x + ", " + y + ", " + z + ")";
}
}

public class Test {
public static void main(String[] args) {
Point p1 = new Point();
p1.x = 17;
p1.y = 35;
p1.z = -1;

Point p2 = new Point();
p2.x = 19;
p2.y = 40;
p2.z = 0;

System.out.println(p1); //Line n1
System.out.println(p2); //Line n2
}
}
```

- Point(17, 35, -1) Point(19, 40, 0)
- **Point(19, 35, -1) Point(19, 40, 0)**
- Point(17, 35, -1) Point(17, 40, 0)
- Point(19, 40, 0) Point(19, 40, 0)
- Point(17, 35, -1) Point(17, 35, -1)
- Point(19, 35, -1) Point(19, 35, -1)

Correct

SCP87068:

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

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

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

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

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

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

Output is:

Point(19, 35, -1)

Point(19, 40, 0)

40. Question

Given code of Test.java file:

```
//Test.java
package com.skillcertpro.ocp;

public class Test {
public static void main(String[] args) {
int x = 1;
while(checkAndIncrement(x)) {
System.out.println(x);
}
}

private static boolean checkAndIncrement(int x) {
if(x < 5) { x++; return true; } else { return false; } } } What will be the result of compiling and executing Test class?
```

- 2 3 4 5
- 1 2 3 4
- 1 2 3 4 5
- Infinite loop

Incorrect

SCP32350:

x of checkAndIncrement method contains the copy of variable x defined in main method. So, changes done to x in checkAndIncrement method are not reflected in the variable x of main. x of main remains 1 as code inside main is not changing its value.

Every time checkAndIncrement method is invoked with argument value 1, so true is returned always and hence while loop executes indefinitely.

41. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
int i1 = 10;
static int i2 = 20;

private void change1(int val) {
i1 = ++val; //Line n1
i2 = val++; //Line n2
}

private static void change2(int val) {
i1 = –val; //Line n3
i2 = val–; //Line n4
}

public static void main(String[] args) {
change1(5); //Line n5
change2(5); //Line n6
System.out.println(i1 + i2); //Line n7
}
}
```

Incorrect

SCP18047:

i1 is an instance variable and i2 is a static variable.

Instance method can access both instance and static members. Hence, Line n1 and Line n2 compile successfully.

Static method can access only static members. Hence, Line n3 [accessing instance variable i1], Line n5 [accessing instance method change1(int)] and Line n7 [accessing instance variable i1] cause compilation error.

Which of the following statements are correct regarding above code?

Select 3 options.

- [x] Line n1 causes compilation error
- [x] Line n2 causes compilation error
- [x] Line n3 causes compilation error
- [ ] Line n4 causes compilation error
- [ ] Line n5 causes compilation error
- [ ] Line n7 causes compilation error

42. Question

Consider below code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
static int i1 = 10;
int i2 = 20;

int add() {
return this.i1 + this.i2; //Line n1
}

public static void main(String[] args) {
System.out.println(new Test().add()); //Line n2
}
}
```

Correct

SCP68301:

'i1' is a static variable and 'i2' is an instance variable. Preferred way to access static variable 'i1' inside add() method is by using 'i1' or 'Test.i1'. Even though 'this.i1' is not the recommended way but it works.

And instance variable 'i2' can be accessed inside add() method by using 'i2' or 'this.i2'. Hence, Line n1 compiles successfully.

As add() is an instance method of Test class, so an instance of Test class is needed to invoke the add() method. `new Test().add()` correctly invokes the add() method of Test class and returns 30. Line n2 prints 30 on to the console.

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

- (●) It executes successfully and prints 30 on to the console
- ( ) It executes successfully and prints 20 on to the console
- ( ) It executes successfully and prints 10 on to the console
- ( ) Line n1 causes compilation error
- ( ) Line n2 causes compilation error

43. Question

Consider below code of Test.java file

```
package com.skillcertpro.ocp;

class Counter {
static int ctr = 0;
int count = 0;
}

public class Test {
public static void main(String[] args) {
Counter ctr1 = new Counter();
Counter ctr2 = new Counter();
Counter ctr3 = new Counter();

for(int i = 1; i <= 5; i++ ) { ctr1.ctr++; ctr1.count++; ctr2.ctr++; ctr2.count++; ctr3.ctr++; ctr3.count++; } System.out.println(ctr3.ctr + ":" + ctr3.count); } }
```

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

- ◉ Compilation error
- ○ 5:5 is printed on to the console
- ○ 15:15 is printed on to the console
- ○ 15:5 is printed on to the console

Incorrect

SCP71940:

Each instance of the class contains separate copies of instance variable and share one copy of static variable.

There are 3 instances of Counter class created by main method and these are referred by ctr1, ctr2 and ctr3.

As 'ctr' is a static variable of Counter class, hence ctr1.ctr, ctr2.ctr and ctr3.ctr refer to the same variable. In fact, 'Counter.ctr' is the preferred way to refer the static variable 'ctr' but ctr1.ctr, ctr2.ctr and ctr3.ctr are also allowed.

As 'count' is an instance variable, so there are 3 separate copies: ctr1.count, ctr2.count, ctr3.count.

On the completion of for loop: ctr1.count = 5, ctr2.count = 5 and ctr3.count = 5 and Counter.ctr = 15.

15:5 is printed on to the console.

44. Question

Consider the code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
private static void m(int i) {
System.out.print(1);
}

private static void m(int i1, int i2) {
System.out.print(2);
}

private static void m(char... args) {
System.out.print(3);
}

public static void main(String... args) {
m('A');
m('A', 'B');
m('A', 'B', 'C');
m('A', 'B', 'C', 'D');
}
}
```

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

Incorrect

SCP30176:

If choice is between implicit casting and variable arguments, then implicit casting takes precedence because variable arguments syntax was added in Java 5 version. m('A'); is tagged to m(int) as 'A' is char literal and implicitly casted to int. m('A', 'B'); is tagged to m(int, int) as 'A' and 'B' are char literals and implicitly casted to int. m('A', 'B', 'C'); is tagged to m(char...) m('A', 'B', 'C', 'D'); is tagged to m(char...) There is no compilation error and on execution output is: 1233SCP30176:

If choice is between implicit casting and variable arguments, then implicit casting takes precedence because variable arguments syntax was added in Java 5 version.

m('A'); is tagged to m(int) as 'A' is char literal and implicitly casted to int.

m('A', 'B'); is tagged to m(int, int) as 'A' and 'B' are char literals and implicitly casted to int.

m('A', 'B', 'C'); is tagged to m(char...)

m('A', 'B', 'C', 'D'); is tagged to m(char...)

There is no compilation error and on execution output is: 1233

- Above code causes compilation error
- It compiles successfully and on execution prints 3333 on to the console
- **It compiles successfully and on execution prints 1233 on to the console**
- It compiles successfully and on execution prints 1333 on to the console

Question **45** of **55**

45. Question

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

- public class Planet { }
- public void Planet(){}
- public void Planet(int x){}
- **public Planet(String str) {}**
- None of the other options

Incorrect

SCP66122:

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

Out of the given 4 options, `public Planet(String str) {}` is the valid constructor declaration.

46. Question

Consider below code of Person.java file:

```
package com.skillcertpro.ocp;

public class Person {
public String name;
public void Person() {
name = "James";
}

public static void main(String [] args) {
Person obj = new Person();
System.out.println(obj.name);
}
}
```

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

- James
- Compilation error
- None of the other options

Incorrect

SCP35863:

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

In java it is allowed to use same method name as the class name (though not a recommended practice), so no issues with Person() method declaration.

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

public Person() {super();}

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

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

47. Question

When does a class get the default constructor?

- If you define parameterized constructor for the class
- You have to define at least one constructor to get the default constructor
- If the class does not define any constructors explicitly
- All classes in Java get a default constructor

Correct

SCP70611:

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

48. Question

Consider below code of Apple.java file:

```
package com.skillcertpro.ocp;

public class Apple {
public String color;

public Apple(String color) {
/*INSERT*/
}

public static void main(String [] args) {
Apple apple = new Apple("GREEN");
System.out.println(apple.color);
}
}
```

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

- color = color;
- this.color = color;
- color = GREEN;
- this.color = GREEN;

Correct

SCP88395:

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

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

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

49. Question

Consider below code of Greetings.java file:

```
package com.skillcertpro.ocp;

public class Greetings {
String msg = null;

public Greetings() {
this("Good Morning!");


public Greetings(String str) {
msg = str;


public void display() {
System.out.println(msg);

public static void main(String [] args) {
Greetings g1 = new Greetings();
Greetings g2 = new Greetings("Good Evening!");
g1.display();
g2.display();
```

Incorrect

SCP58507:

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

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

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

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

g1.display(); prints Good Morning!

g2.display(); prints Good Evening!

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

- null Good Evening!
- **Good Morning! Good Evening!**
- Good Morning! null
- null null

50. Question

Consider below code of Quotes.java file:

```
package com.skillcertpro.ocp;

public class Quotes {
String quote = null;
public Quotes() {
}

public Quotes(String str) {
quote = str;
}

public void display() {
System.out.println(quote);
}

public static void main(String [] args) {
Quotes q1 = new Quotes();
Quotes q2 = new Quotes("NEVER GIVE UP!");
q1.display();
q1.display();
}
}
```

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

- null NEVER GIVE UP!
- null null
- NEVER GIVE UP! null
- NEVER GIVE UP! NEVER GIVE UP!
- Compilation error

Incorrect

SCP80413:

Quotes class contains overloaded constructors: a no-argument constructor Quotes() and a parameterized constructor Quotes(String).

Quotes q1 = new Quotes(); invokes no-argument constructor. null is assigned to the property 'quote' of object referred by q1.

Quotes q2 = new Quotes("NEVER GIVE UP!"); invokes parameterized constructor, which assigns "NEVER GIVE UP!" to 'quote' of object referred by q2.

q1.display(); prints null.

Again we have same call q1.display(); which prints null.

NOTE: We haven't called display() on object referred by q2.

51. Question

Consider the code of TestEmployee.java file:

```
package com.skillcertpro.ocp;

class Employee {
String name;
nt age;

Employee() {
Employee("Michael", 22); //Line n1


Employee(String name, int age) {
his.name = name;
his.age = age;


ublic class TestEmployee {
ublic static void main(String[] args) {
Employee emp = new Employee();
System.out.println(emp.name + ":" + emp.age); //Line n2
```

Correct

SCP80898:

Employee class has overloaded constructors: Employee() is a no-argument constructor and Employee(String, int) is a parameterized constructor.

A constructor can call another constructor by using this(...) and not the constructor name. Hence Line n1 causes compilation error.

As both Employee and TestEmployee classes are defined in the same package, hence emp.name and emp.age are valid syntaxes. Line n2 compiles successfully.

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

- (●) Compilation error at Line n1
- ( ) Compilation error at Line n2
- ( ) null:0
- ( ) Michael:22
- ( ) An exception is thrown at runtime

52. Question

Consider the code of TestPerson.java file:

```
package com.skillcertpro.ocp;

class Person {
String name;
int age;

Person() {
Person("Rohit", 25);
}

Person(String name, int age) {
this.name = name;
this.age = age;
}
}

public class TestPerson {
public static void main(String[] args) {
Person p = new Person();
System.out.println(p.name + ":" + p.age);
}
}
```

Incorrect

SCP56787:

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

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

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

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

There is a compilation error in the Person class.

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

- [x] Add below code in the Person class:
- [ ] **void Person(String name, int age) { this.name = name; this.age = age; }**
- [x] Replace Person("Rohit", 25); with super("Rohit", 25);
- [ ] **Replace Person("Rohit", 25); with this("Rohit", 25);**
- [ ] Replace Person("Rohit", 25); with this.Person("Rohit", 25);

53. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

public class Test {
static int a = 10000;
static {
a = -a--;
}
{
a = -a++;
}

public static void main(String[] args) {
System.out.println(a);
}
}
```

What is the result?

- Compilation error
- -10000
- 10000
- 9999
- -9999

Incorrect

SCP54125:

Variable 'a' is of static type, so both static and instance initializer blocks can access it. Given code compiles successfully.

We are not creating the instance of Test class, so instance initializer block will not be executed. Only static initializer block will be executed in this case.

If static variable declaration / initialization statements are present along with static initializer blocks, then these are invoked in top to bottom order. So, for the given code, order of execution will be:

1. static int a = 10000;

and then

2. static { a = -a--; }

Statement 1, initializes variable 'a' to 10000.

Let's solve the statement inside static initializer block:

a = -a--; [a = 10000].

a = -(a--); [a = 10000] Postfix operator has got higher precedence than unary operator.

a= -(10000); [a = 9999] Use the value of a (10000) in the expression and after that decrement the value of a to 9999.

a = -10000; [a = -10000] Assigns -10000 to a

System.out.println(a); inside main method prints -10000

54. Question

Given code of Test.java file:

```
package com.skillcertpro.ocp;

class Initializer {
static int num;
static int den;
{
num = 100;
den = 10;
}
static {
num = num/den;
}
}

public class Test {
public static void main(String[] args) {
System.out.println(Initializer.num);
}
}
```

What is the result?

- Compilation error
- 10
- 100
- 1
- 0
- ExceptionInInitializerError is thrown at runtime

Incorrect

SCP56301:

Variables 'num' and 'den' are of static type, so both static and instance initializer blocks can access these. Given code compiles successfully.

We are not creating the instance of Initializer class, so instance initializer block will not be executed. Only static initializer block will be executed in this case.

If static variable declaration / initialization statements are present along with static initializer blocks, then these are invoked in top to bottom order. So, for the given code, order of execution will be:

static int num; => It creates static variable 'num' and assigns 0 to it.

static int den; => It creates static variable 'den' and assigns 0 to it.

```
static {

   num = num/den;

}
```

As den is 0, hence ArithmeticException is thrown by this statement but as this statement is inside static initializer block, hence ExceptionInInitializerError will be thrown.

Please Note: Any exception caused by static declaration and initialization statements or statements inside static initializer block will be thrown as ExceptionInInitializerError.

Back | Report Question | Next

55. Question

Consider below code:

```
package com.skillcertpro.ocp;

public class Test {
static Double d1;
static int x = d1.intValue();

public static void main(String[] args) {
System.out.println("HELLO");
}
}
```

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

- Yes, HELLO is printed on to the console
- No, HELLO is not printed on to the console

Incorrect

SCP77262:

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

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

Please Note: Any exception caused by static declaration and initialization statements or statements inside static initializer block will be thrown as ExceptionInInitializerError.