Java OOPs (Object Oriented Programming) Tricky Coding Interview Questions

Author: Ramesh Fadatare
Interview Java Quiz MCQ Oops
This page contains Java OOPs (Object Oriented Programming) coding interview questions on OOPs concepts such as Abstraction, Encapsulation, Inheritance, and Polymorphism.

Also, check out the Java Tricky Coding interview questions: Top 10 Java Tricky Coding Interview Questions.

These questions may be asked in interviews, or similar questions may appear in interviews, so prepare yourself.
Learn OOPs at OOPs Concepts in Java with Realtime Examples

1. What is the output of the following Java program?

```
class Automobile {
    private String drive() {
        return "Driving vehicle";
    }
}

class Car extends Automobile {
    protected String drive() {
        return "Driving car";
    }
}

public class ElectricCar extends Car {

    @Override
    public final String drive() {
        return "Driving an electric car";
    }

    public static void main(String[] wheels) {
        final Car car = new ElectricCar();
        System.out.print(car.drive());
    }
}
```

A. Driving vehicle
B. Driving an electric car
C. Driving car
D. The code does not compile

Answer :

B. Driving an electric car

Explanation:

The *drive()* method in the *Car* class does not override the version in the *Automobile* class since the method is not visible to the *Car* class.

The *drive()* method in the *ElectricCar* class is a valid override of the method in the *Car* class, with the access modifier expanding in the subclass. For these reasons, the code compiles, and Option D is incorrect.

In the *main()* method, the object created is an *ElectricCar*, even if it is assigned to a *Car* reference. Due to polymorphism, the method from the *ElectricCar* will be invoked, making Option B the correct answer.

2. Look at the following code and choose the right option for the word :

```
// Shape.java
public class Shape {
    protected void display() {
        System.out.println("Display-base");
    }
}
// Circle.java
public class Circle extends Shape { <
    < access - modifier > void display() {
        System.out.println("Display-derived");
    }
}
```

a) Only the protected can be used.
b) public and protected both can be used.
c) public, protected, and private can be used.
d) Only the public can be used.

Answer :

B. public and protected both can be used.

Explanation:

You can provide only a less restrictive or same-access modifier when overriding a method.

3. What will be the output of the following Java program?

```
class Base {
    public Base() {
```

```
        System.out.println("Base");
    }
}

class Derived extends Base {
    public Derived() {
        System.out.println("Derived");
    }
}

class DeriDerived extends Derived {
    public DeriDerived() {
        System.out.println("DeriDerived");
    }
}

public class Test {
    public static void main(String[] args) {
        Derived b = new DeriDerived();
    }
}
```

a)

```
Base
Derived
DeriDerived
```

b)

```
Derived
DeriDerived
```

c)

```
DeriDerived
Derived
Base
```

d)

```
DeriDerived
Derived
```

Answer:

a)

```
Base
Derived
DeriDerived
```

Explanation:

Whenever a class gets instantiated, the constructor of its base classes (the constructor of the root of the hierarchy gets executed first) gets invoked before the constructor of the instantiated class.

4. What is the output of the following Java program?

```
class One{
        public One(){
                System.out.print("One,");
        }
}
class Two extends One{
        public Two(){
                System.out.print("Two,");
        }
}
class Three extends Two{
        public Three(){
                System.out.print("Three");
        }
}

public class Test{

        public static void main(String[] args){
                Three three = new Three();
        }
}
```

a) Three
b) One
c) One,Two,Three
d) Run-time error

Answer:

c) One,Two,Three

Explanation:

When we create an object of class *Three*, the constructors are executed in the following order:
One(): Prints "One,".
Two(): Prints "Two,".
Three(): Prints "Three".
So, the overall output is "One,Two,Three". The constructors are executed in the order of inheritance hierarchy from the topmost superclass (One) to the subclass (Three).

5. Consider the following program:

```
class Base {
    public Base() {
        System.out.print("Base ");
    }
```

```
    public Base(String s) {
        System.out.print("Base: " + s);
    }
}

class Derived extends Base {
    public Derived(String s) {
        super(); // Stmt-1
        super(s); // Stmt-2
        System.out.print("Derived ");
    }
}

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

Select three correct options from the following list:
a) Removing Stmt-1 will make the program compilable and it will print the following: Base Derived.
b) Removing Stmt-1 will make the program compilable and it will print the following: Base: Hello Derived.
c) Removing Stmt-2 will make the program compilable and it will print the following: Base Derived.
d) Removing both Stmt-1 and Stmt-2 will make the program compilable and it will print the following: Base Derived.
e) Removing both Stmt-1 and Stmt-2 will make the program compilable and it will print the following: Base: Hello Derived.

Answer:

b) Removing Stmt-1 will make the program compilable and it will print the following: Base: Hello Derived.
c) Removing Stmt-2 will make the program compilable and it will print the following: Base Derived.
d) Removing both Stmt-1 and Stmt-2 will make the program compilable and it will print the following: Base Derived.

Explanation:

If you remove Stmt-1, a call to super(s) will result in printing Base: Hello, and then the constructor of the Derived class invocation will print Derived. Similarly, the removal of Stmt-2 will also produce the correct program. In fact, if you remove both these statements, you will also get a compilable program.

6. What is the output of the following Java program?

```
abstract class Car {
    static {
        System.out.print("1");
    }

    public Car(String name) {
        super();
        System.out.print("2");
    }

    {
        System.out.print("3");
    }
}

public class BlueCar extends Car {
    {
        System.out.print("4");
    }

    public BlueCar() {
        super("blue");
        System.out.print("5");
    }

    public static void main(String[] gears) {
        new BlueCar();
    }
}
```

a) 23451
b) 12354
c) 13245
d) The code does not compile.

Answer:

c) 13245

Explanation:

The class is loaded first, with the static initialization block called and 1 is outputted first. When the *BlueCar* is created in the *main()* method, the superclass initialization happens first. The instance initialization blocks are executed before the constructor, so 32 is outputted next.

Finally, the class is loaded with the instance initialization blocks again being called before the constructor, outputting 45. The result is that 13245 is printed, making **Option C the correct answer.**

7. What is the output of the following Java program?

```
class Math {
    public final double secret = 2;
}

class ComplexMath extends Math {
    public final double secret = 4;
}

public class InfiniteMath extends ComplexMath {
    public final double secret = 8;

    public static void main(String[] numbers) {
        Math math = new InfiniteMath();
        System.out.print(math.secret);
    }
}
```

A. 2
B. 4
C. 8
D. The code does not compile.

Answer:

A. 2

Explanation:

The code compiles without issue, so Option D is incorrect. Java allows methods to be overridden, but not variables. Therefore, marking them final does not prevent them from being reimplemented in a subclass. Furthermore, polymorphism does not apply in the same way it would to methods as it does to variables. In particular, the reference type determines the version of the *secret* variable that is selected, making Output 2 and **Option A the correct answer.**

8. What is the output of the following Java program?

```
public class Test {
    public void print(Integer i) {
        System.out.println("Integer");
    }

    public void print(int i) {
        System.out.println("int");
    }

    public void print(long i) {
        System.out.println("long");
```

```
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.print(10);
    }
}
```

a) The program results in a compiler error (“ambiguous overload”).
b) long
c) Integer
d) int

Answer:

d) int

Explanation:

For an integer literal, the JVM matches in the following order: **int, long, Integer, int...**. In other words, it first looks for an *int* type parameter; then it looks for *long* type; and so on. Here, since the *int* type parameter is specified with an overloaded method, it matches with *int*.

9. What is the output of the following Java program?

```
class One{
        public static void print(){
                System.out.println("1");
        }
}

class Two extends One{
        public static void print(){
                System.out.println("2");
        }
}

public class Test{
        public static void main(String args[]){
                One one = new Two();
                one.print();
        }
}
```

a) 2
b) 1
c) Compile-time error
d) Run-time error

Answer:

b) 1

Explanation:

Static methods are defined at the class level. In the case of the static methods, regardless of which object the reference is pointing to, it always calls the static method defined by the reference class. Here, the reference is of type *One*, so the static method of class *One* will be called.

10. What is the output of the following Java program?

```
class Parent{
        public void className(){
                System.out.println("Parent");
        }
}
class Child extends Parent{
        void className(){
                System.out.println("Child");
        }
}

public class Test{

        public static void main(String[] args){
                Parent parent = new Child();
                parent.className();
        }
}
```

a) Parent
b) Child
c) Compile-time error
d) Run-time error

Answer:

c) Compile-time error

Explanation:

When overriding a parent class method in a child class, we cannot reduce the visibility of the method. For example, if the method is defined as public in the parent class, a child class cannot override it with protected. The code will give the compilation error **“Cannot reduce the visibility of the inherited method from Parent”.**

11. What is the output of the following Java program?

```
class Demo{
```

```
        public Demo(int i){
                System.out.println("int");
        }

        public void Demo(short s){
                System.out.println("short");
        }
}

public class Test{

        public static void main(String[] args){
                short s = 10;
                Demo demo = new Demo(s);
        }
}
```

a) int
b) short
c) Compile-time error
d) Run-time error

Answer:

a) int

Explanation:

The class *Demo* has one constructor i.e. with *int* argument. The *short* value is automatically promoted to an *int* value during object creation so the constructor with the *int* argument will be called and it will print “int”.

Conclusion

In conclusion, mastering Object-Oriented Programming (OOPs) concepts is essential for any Java developer looking to excel in coding interviews. OOPs provides a powerful foundation for creating modular, maintainable, and extensible Java applications. In this blog post, we have explored a collection of tricky coding interview questions related to Java OOPs concepts.

Check out 100+ quiz questions: 100+ Quiz Questions to Test Your Java, Spring Boot, Microservices, Hibernate, REST API Skills.