Tricky and Interesting Java Interview Questions (Part-1)

1. What Happens If You Override a Private Method?

Question:

Can we override a private method in Java?

Explanation:

No, private methods cannot be overridden because they are **not visible** to subclasses. However, if you define a method with the same name in a subclass, it is considered **method hiding**, not overriding.

Example:

```
 class Parent {
    private void display() {
        System.out.println("Parent method");
    }
}
class Child extends Parent {
    private void display() {
        System.out.println("Child method");
    }
}
public class Test {
    public static void main(String[] args) {
        Parent obj = new Child();
        // obj.display(); // Compilation error
    }
}
```

Output: Compilation error because private methods are not accessible.

2. Can You Catch an Exception Thrown in a Static Block?

Question:

What happens if an exception is thrown in a static block?

Explanation:

Exceptions thrown in a static block can only be caught within a **static initializer block** using a `try-catch`. Otherwise, the program will fail to load the class.

Example:

```
 class Test {
    static {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught: " + e);
        }
    }
public static void main(String[] args) {
        System.out.println("Main method executed");
    }
}
```

Output:

```
 Exception caught: java.lang.ArithmeticException: / by zero
Main method executed
```

3. What Happens If You Call the Run Method Directly Instead of Start()?

Question:

What is the difference between calling `run()` and `start()` for a thread?

Explanation:

- Calling `start()` creates a **new thread** and executes the `run()` method in that thread.
- Calling `run()` directly executes the code in the **current thread** without starting a new thread.

Example:

```
 class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running");
    }
}
public class Test {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.run(); // Executes in main thread
        t1.start(); // Executes in a new thread
    }
}
```

Output:

```
 Thread running
Thread running
```

4. What Happens If You Call Wait() Outside a Synchronized Block?

Question:

What exception is thrown if `wait()` is called outside a synchronized block?

Explanation:

Calling `wait()` outside a synchronized block results in **IllegalMonitorStateException** because `wait()` must be called with a lock acquired.

Example:

```
 public class Test {
    public static void main(String[] args) throws InterruptedException {
        Object obj = new Object();
        obj.wait(); // Throws IllegalMonitorStateException
    }
}
```

5. Can a Constructor Be Synchronized?

Question:

Is it possible to declare a constructor as `synchronized`?

Explanation:

No, constructors cannot be synchronized because object locks do not exist until the object is created. However, synchronized blocks can be used inside constructors.

Example:

```
 class Test {
    Test() {
        synchronized (this) {
            System.out.println("Synchronized block in constructor");
        }
    }
}
```

6. What Happens If You Return From Try or Catch? Will Finally Execute?

Question:

If there is a `return` statement inside `try` or `catch`, will `finally` execute?

Explanation:

Yes, the `finally` block always executes, even if there is a `return` statement inside `try` or `catch`. However, if `finally` also has a `return`, it overrides other return values.

Example:

```
 public class Test {
    public static int testMethod() {
        try {
            return 1;
        } catch (Exception e) {
            return 2;
        } finally {
            return 3;
        }
    }
public static void main(String[] args) {
        System.out.println(testMethod()); // Output: 3
    }
}
```

7. What Happens If an Exception Is Thrown in Finally?

Question:

What happens if the `finally` block throws an exception?

Explanation:

Exceptions in the `finally` block suppress exceptions from the `try` or `catch` blocks.

Example:

```
 public class Test {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (Exception e) {
            System.out.println("Catch executed");
        } finally {
            throw new RuntimeException("Exception in finally");
        }
    }
}
```

Output:

```
Exception in finally
```