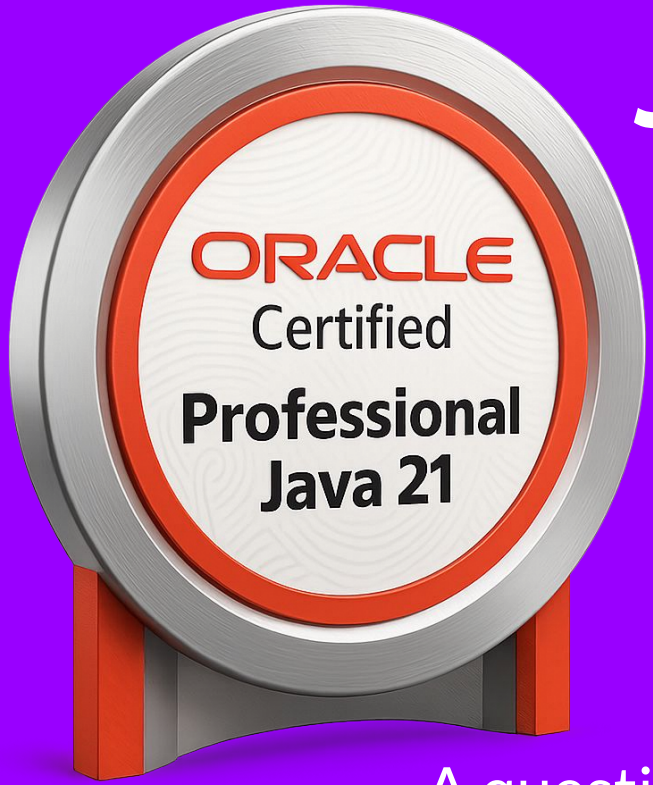


Java Certified #1

A question lead guide to prepare Java certification

Using Object-Oriented Concepts in Java

Given:

```
public class Test {
    class A {}
    static class B {}
    public static void main( String[] args ) {
        // Insert here
    }
}
```

Which three of the following are valid statements when inserted into the given program?

- ➔ **A a = new A();**
- ➔ **B b = new B();**
- ➔ **A a = new Test.A();**
- ➔ **B b = new Test.B();**
- ➔ **A a = new Test().new A();**
- ➔ **B b = new Test().new B();**

answer

`A a  = new A();` is wrong since class `A` is a non-static member of the Test class - it cannot be directly referenced from a static context.

`A a = new Test.A();` is wrong since class `A` is a non-static member - it cannot be associated with the `Test` class itself.

`B b = new Test().new B();` is incorrect as class `B` is a static member of the `Test` class. It must be associated with the containing class rather than one of its instances.

Using Java I/O API

Which of the following isn't a correct way to write a string to a file?

➔
```
try( FileWriter writer = new FileWriter( "file.txt" ) ){
    writer.write("Hello");
}
```

➔
```
try ( PrintWriter printWriter = new PrintWriter( "file.txt" ) ) {
    printWriter.printf( "Hello %s", "James" );
}
```

➔
```
try ( BufferedWriter writer = new BufferedWriter( "file.txt" ) ) {
  writer.write( "Hello" );
}
```

```
try ( BufferedWriter writer = new BufferedWriter( "file.txt" ) ) {
    writer.write( "Hello" );
}
```

A `BufferedWriter` object must wrap a `Writer` and cannot be constructed with a `String` argument. Consequently, the code fragment in option

```
try ( BufferedWriter writer = new BufferedWriter( "file.txt" ) ) {
    writer.write( "Hello" );
}
```

fails to compile.

Which one of the following are correct about the Java module system?

- ➔ **If a request is made to load a type whose package is not defined in any known module, then the module system will attempt to load it from the classpath.**
- ➔ **The unnamed module can only access packages defined in the unnamed module.**
- ➔ **We must add a module descriptor to make an application developed using a Java version prior to SE9 running on Java 21.**

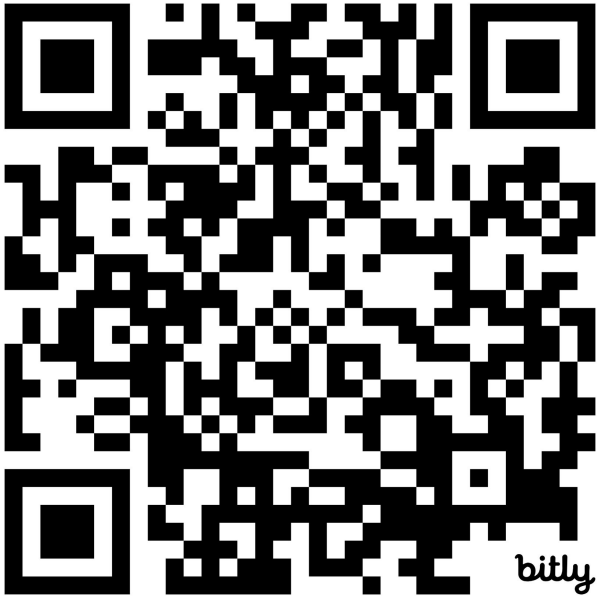
If a request is made to load a type whose package is not defined in any known module, then the module system will attempt to load it from the classpath.

- ✅ **If a request is made to load a type whose package isn't in any known (named) module, JPMS will try the classpath** (i.e., the **unnamed module**) and load it from there if available.

- ❌ **"The unnamed module can only access packages defined in the unnamed module."** Incorrect. The unnamed module can read **all named modules** and access their **exported** packages.

- ❌ **"We must add a module descriptor to run pre-SE9 apps on Java 21."** Nope. You can still run on the **classpath** without any `module-info.java`.

—



https://bit.ly/javaOCP