Java Certified #14

A question lead guide to prepare Java certification

Using Object-Oriented Concepts in Java

Given:

```
record WithInstanceField(String foo, int bar) {
    double fuz;
}
record WithStaticField(String foo, int bar) {
    static double wiz;
}
record ExtendingClass(String foo) extends Exception {}
record ImplementingInterface(String foo) implements Cloneable {}
```

Which records compile? (Select 2)

- ➔ **WithInstanceField**
- ➔ **WithStaticField**
- ➔ **ExtendingClass**
- ➔ **ImplementingInterface**

WithStaticField ImplementingInterface

Let's explore one record at a time:

```
record WithInstanceField(String foo, int bar) {
    double fuz;
}
```

Do not compile because the Instance field 'fuz' is not allowed in the record.

```
record WithStaticField(String foo, int bar) {
    static double wiz;
}
```

Does compile.

```
record ExtendingClass(String foo) extends Exception {}
```

It does not compile because No extends clause allowed for record.

```
record ImplementingInterface(String foo) implements Cloneable {}
```

Does compile.

Details on JEP 395: https://openjdk.org/jeps/395

Using Object-Oriented Concepts in Java

Given:

```
interface SmartPhone {
    boolean ring();
}
```

```
class Iphone15 implements SmartPhone {
    boolean isRinging;
    boolean ring() {
        isRinging = !isRinging;
        return isRinging;
    }
}
```

Choose the right statement.

- ➔ SmartPhone interface does not compile
- ➔ Iphone15 class does not compile
- ➔ Everything compiles
- ➔ An exception is thrown at running Iphone15.ring();

Iphone15 class does not compile

In the provided code:

1. **SmartPhone Interface**
   - The `SmartPhone` interface declares a method `boolean ring();`
   - This compiles correctly because there is nothing wrong with the syntax or definition of the interface.
2. **Iphone15 Class**
   - The `Iphone15` class implements the `SmartPhone` interface but defines the `ring()` method with default (package-private) access.
   - According to Java rules, methods implementing an interface must have the same or greater access visibility than the interface method. Since `ring()` in the `SmartPhone` interface is implicitly `public`, the overriding `ring()` method in the `Iphone15` class must also be declared `public`.
   - As written, the `Iphone15` class does not compile because the `ring()` method attempts to reduce the access level, which is not allowed.
3. **Other Options**
   - **Smartphone:** The `SmartPhone` interface compiles correctly, so this is incorrect.
   - **Everything compiles:** Not everything compiles; the `Iphone15` class has a compilation error.
   - **An exception is thrown:** Since the code doesn't compile, an exception cannot be thrown at runtime.

Key Point:
The error message is:
`'ring()' in 'Iphone15' clashes with 'ring()' in 'SmartPhone'; attempting to assign weaker access privileges ('package-private'); was 'public'`.

Working with Arrays and Collections

Given the following snippets,

Which variable prints 2025-W01-2 (present-day is 12/31/2024).

-
  ```
  var now = LocalDate.now();
  var format1 = new DateTimeFormatter( ISO_WEEK_DATE );
  System.out.println( now.format( format1 ) );
  ```

-
  ```
  var now = LocalDate.now();
  var format2 = DateTimeFormatter.ISO_WEEK_DATE;
  System.out.println( now.format( format2 ) );
  ```

-
  ```
  var now = LocalDate.now();
  var format3 = new DateFormat( WEEK_OF_YEAR_FIELD );
  System.out.println( now.format( format3 ) );
  ```

-
  ```
  var now = LocalDate.now();
  var format4 = DateFormat.getDateInstance( WEEK_OF_YEAR_FIELD );
  System.out.println( now.format( format4 ) );
  ```

```
var now = LocalDate.now();
var format2 = DateTimeFormatter.ISO_WEEK_DATE;
System.out.println( now.format( format2 ) );
```

Explanation:

1. `var format1 = new DateTimeFormatter(ISO_WEEK_DATE);`
   - This does not compile because there is no such constructor in the `DateTimeFormatter` class.
2. `var format2 = DateTimeFormatter.ISO_WEEK_DATE;`
   - This compiles successfully. `ISO_WEEK_DATE` is a predefined constant in the `DateTimeFormatter` class that formats dates according to the ISO-8601 week-based calendar system.
3. `var format3 = new DateFormat(WEEK_OF_YEAR_FIELD);`
   - This does not compile because `DateFormat` is an abstract class and cannot be directly instantiated.
4. `var format4 = DateFormat.getDateInstance(WEEK_OF_YEAR_FIELD);`
   - This does not work as expected because `DateFormat` does not provide functionality to format a `LocalDate` instance in this manner.

When calling `now.format(REPLACE_HERE)` with the correct formatter, `format2` (which is `DateTimeFormatter.ISO_WEEK_DATE`) successfully formats the date `2024-12-31` into the ISO week date format: `2025-W01-2`.

https://bit.ly/javaOCP