Java Certified #9

A question lead guide to prepare Java certification

Handling Exceptions

Given:

```
public static void main( String[] args ) {
    try {        throw new IOException();    }
    catch ( IOException e ) {  throw new RuntimeException();  }
    finally {  throw new ArithmeticException();  }
}
```

What is the output?

- ➔ IOException
- ➔ RuntimeException
- ➔ ArithmeticException
- ➔ Compilation fails

ArithmeticException

In the given Java code, the `main` method throws an `IOException` inside a `try` block, which is then caught by the `catch` block that throws a `RuntimeException`. However, because there is a `finally` block that throws an `ArithmeticException`, the `ArithmeticException` will be the one that actually gets thrown when the `main` method is executed.

Here's the reason:

- The `try` block throws an `IOException`.

- The `catch` block catches this exception and throws a new `RuntimeException`.

- Regardless of whether an exception was thrown or not, the `finally` block is always executed. In this case, it throws an `ArithmeticException`.

- Since the `finally` block is the last to execute and it throws an exception, the `ArithmeticException` will be the one that propagates up the call stack.

Therefore, the output will be:

* `ArithmeticException`

The other exceptions (`IOException` and `RuntimeException`) are thrown but not propagated, as the `ArithmeticException` from the `finally` block supersedes them. Compilation does not fail because the code is syntactically correct.

Handling Date, Time, Text, Numeric and Boolean Values

Given:

```
double amount = 42_000.00;

NumberFormat format = NumberFormat.getCompactNumberInstance( Locale.FRANCE,
NumberFormat.Style.SHORT );

System.out.println( format.format( amount ) );
```

What is the output?

- ➔ **42000**
- ➔ **42 k**
- ➔ **42000E**
- ➔ **42 000,00 €**

42 k

The correct answer is:

* `42 k`

Here's why:

The `NumberFormat.getCompactNumberInstance(Locale.FRANCE, NumberFormat.Style.SHORT)` method in Java is used to format numbers in a compact style for the specified locale.

For `Locale.FRANCE`, the compact number instance will format the number `42000.00` as "`42k`",

because the `SHORT` style typically represents the thousands with a '`k`'.

The currency symbol is not included in the output when using the `getCompactNumberInstance` method,

and the decimal part is not shown for whole numbers when formatted in the `SHORT` style.

Numbers and Currencies: Using Predefined Formats

https://docs.oracle.com/javase/tutorial/i18n/format/numberFormat.html

Managing Concurrent Code Execution

Given:

var sList = new CopyOnWriteArrayList<Customer>();

Which of the following statement is correct?

- The `CopyOnWriteArrayList` class is not thread-safe and does not prevent interference among concurrent threads.
- The `CopyOnWriteArrayList` class's iterator reflects all additions, removals, or changes to the list since the iterator was created.
- The `CopyOnWriteArrayList` class is a thread-safe variant of ArrayList where all mutative operations are implemented by making a fresh copy of the underlying array.
- Element-changing operations on iterators of `CopyOnWriteArrayList`, such as remove, set, and add, are supported and do not throw `UnsupportedOperationException`.
- The CopyOnWriteArrayList class does not allow null elements

The `CopyOnWriteArrayList` class is a thread-safe variant of ArrayList where all mutative operations are implemented by making a fresh copy of the underlying array.

Incorrect: The `CopyOnWriteArrayList` class is thread-safe and is designed to prevent interference among concurrent threads by creating a fresh copy of the underlying array for each mutative operation.

Incorrect: The iterator of a `CopyOnWriteArrayList` does not reflect any additions, removals, or changes to the list since the iterator was created. It provides a snapshot of the list at the time of its creation.

Incorrect: Element-changing operations on iterators of `CopyOnWriteArrayList`, such as remove, set, and add, are not supported. Attempting to perform these operations will result in an `UnsupportedOperationException`.

Incorrect: The `CopyOnWriteArrayList` class does allow `null` elements. It permits all elements, including `null`.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/concurrent/CopyOnWriteArrayList.html https://bit.ly/javaOCP