Java Certified #3

A question lead guide to prepare Java certification

Working with Arrays and Collections

Given:

```
Deque<Integer> deque = new ArrayDeque();
deque.offer( 1 ); deque.offer( 2 );
var i1 = deque.peek(); var i2 = deque.poll(); var i3 = deque.peek();
System.out.println( i1 + " " + i2 + " " + i3 );
```

What is the output of the given code fragment?

- ➔ **1 1 1**
- ➔ **1 1 2**
- ➔ **1 2 1**
- ➔ **1 2 2**
- ➔ **An exception is thrown**

1 1 2

```
1 1 2
```

The `offer` method inserts new elements at the tail of the deque. Therefore, after two invocations of this method, our deque has two elements: the first is 1 and the second is 2.

Both the `peek` and `poll` method read at the head of the deque. The difference between them is that the `peek` method doesn't remove the retrieved element, while the `poll` method does.

In the given code, number `1` is read by `peek` and assigned to variable `i1` . This number is read the second time by the `poll` method and assigned to `i2` . At this point, it's also removed from the deque. The last invocation of the `peek` method retrieves number `2` , which was at the head of the deque after removal of number `1` .

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Deque.html

Working with Streams and Lambda expressions

```
Given: interface Calculable {     long calculate( int i ); }

public class Test {    public static void main( String[] args ) {
       Calculable c1 = i -> i + 1; // Line 1
       Calculable c2 = i -> Long.valueOf( i );// Line 2
       Calculable c3 = i -> { throw new ArithmeticException(); };// Line 3
   }} // Which lines fail to compile?
```

- ➔ **Line 1 only**
- ➔ **Line 2 only**
- ➔ **Line 3 only**
- ➔ **Line 1 and line 2**
- ➔ **Line 2 and line 3**
- ➔ **The program successfully compiles**

The program successfully compiles

According to the Java Language Specification:

- If the function type's result is a (non-`void`) type R, then either (i) the lambda body is an expression that is compatible with R in an assignment context, or (ii) the lambda body is a value-compatible block, and each result expression (§15.27.2) is compatible with R in an assignment context.
- A checked exception that can be thrown in the body of the lambda expression may cause a compile-time error, as specified in §11.2.3.

From the Specification, we can see that the result of a lambda body doesn't need to be of a type that is the same as or a subtype of the target function's return type. The restriction is that the body's return value is assignable to the return type of the target function.

On line 1 and line 2, the bodies' return values are of type `int` and `Long` , respectively. These values are assignable to the `long` data type, thanks to casting and unboxing. Therefore, both lines 1 and 2 are valid.

On line 3, the thrown exception is unchecked, hence the expression is also correct. Line 3 would have failed to compile if the exception had been a checked exception.

Working with Streams and Lambda expressions

Given:

```
Optional o1 = Optional.empty(); Optional o2 = Optional.of( 1 );
Optional o3 = Stream.of( o1, o2 )
                    .filter( Optional::isPresent )
                    .findAny()
                    .flatMap( o -> o );

System.out.println( o3.orElse( 2 ) );
 //What is the given code fragment's output?
```

- ➔ 0
- ➔ 1
- ➔ 2
- ➔ Optional.empty
- ➔ Optional[1]
- ➔ Compilation fails

1

The `Stream.filter` operation allows only the `Optional` object referenced by variable `o2` to pass through.

After the `Stream.findAny` terminal operation is performed, we have an `Optional` object that contains the `Optional` to which `o2` refers.

This outer `Optional` then goes to the `Optional.flatMap` method, where the inner `Optional` is extracted and assigned to variable `o3` .

After this assignment, variable `o2` and `o3` point to the same object. Since this object was created with value `1` , the given code prints number `1` to the console.

If a value is present, `orElse` returns the value, otherwise returns `other`.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Optional.html#orElse(T) https://bit.ly/javaOCP