

Solutions to Quick Check Questions

8

Exceptions and Assertions

8.1 Catching Exceptions

1. What would be an output from the following code if the second `parseInt` throws an exception?

```
try {
    int num1 = Integer.parseInt("14");
    System.out.println("Okay 1");
    int num2 = Integer.parseInt("one");
    System.out.println("Okay 2");
} catch (NumberFormatException e) {
    System.out.println("Error");
}
```

Answer:

```
Okay 1
Error
```

2. What is wrong with the following code? It attempts to loop until the valid input is entered.

```
String inputStr;
int num;
```

```

try {
    while (true) {
        inputStr = JOptionPane.showInputDialog(null,
                                                "Input:");
        num      = Integer.parseInt(inputStr);
    }
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(null, "Error");
}

```

The try-catch block must be inside the loop. As the code is now written, when there's an exception, the control jumps out of the loop, so the enduser won't get another try.

8.2 Throwing Exceptions

1. What's wrong with the following code? Identify all errors.

```

try {
    number = Integer.parseInt("123");

    if (num > 100) {
        A → catch new Exception("Out of bound");
    }

    B → } catch {

        System.out.println("Cannot convert to int");

    C → } finally (Exception e) {

        System.out.println("Always print");
    }
}

```

A. The correct reserved word here is throw.

B. The catch clause requires a parameter.

C. The finally clause does not include a parameter.

2. Determine the output of the following code:.

```

try {
    number = Integer.parseInt("a23");
    if (number < 0) {
        throw new Exception("No negative");
    }
}

```

```

    }

} catch (NumberFormatException e) {
    System.out.println("Cannot convert to int");

} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());

} finally {
    System.out.println("DONE");
}

```

Answer:

```

    Cannot convert to int
    Done

```

3. Determine the output of the following code:

```

try {
    number = Integer.parseInt("-30");
    if (number < 0) {
        throw new Exception("No negative");
    }
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());

} catch (NumberFormatException e) {
    System.out.println("Cannot convert to int");
}

```

Answer:

```

    Error: No negative

```

8.3 Propagating Exceptions

1. What's wrong with the following code?

```

public void check(int num) {
    if (num < 0) {
        throw new Exception();
    }
}

```

The method header must include the clause throws Exception.

2. What is the difference between the reserved words throw and throws?

The reserved word throw is used when you throw an exception. The reserved word throws is used in the method header if this method includes a statement that can throw an exception.

3. What's wrong with the following code?

```
public NumberFormatException getData( ) {
    try {
        String str = JOptionPane.showInputDialog(
            null, "Input:");

        int num = Integer.parseInt(str);

        return num;
    }
}
```

The exception type such as NumberFormatException cannot be a return type. The correct declaration is

```
public int getData() throws NumberFormatException {
    ...
}
```

8.4 Types of Exceptions

1. Is the following code wrong?

```
public void check(int num) {
    if (num < 0) {
        throw new IllegalArgumentException();
    }
}
```

The code is correct. Runtime exceptions do not have to be declared in the method header.

2. What is the difference between the checked and unchecked exceptions?

The checked exceptions are those checked at the compile time. The unchecked exceptions are those detected at runtime. For this reason, unchecked exceptions are also called runtime exceptions.

3. Is the following code wrong?

```
public void getData( ) {
    String str = JOptionPane.showInputDialog(
        null, "Input:");

    if (str.equals("")) {
        throw new IOException();
    }
}
```

The code is wrong because IOException is a checked exception, the method declaration must be

```
public void getData() throws IOException {
    ...
}
```

8.5 Programmer-Defined Exceptions

1. When do we want to define a customized exception class?

When we need to include additional information to a thrown exception.

2. Should a customized exception class be a checked or unchecked exception?

Logically, a customized exception class should be a checked exception because we want it to be checked at the compile time.

8.6 Assertions

1. Why is the following code wrong?

```
public void doWork(int num) {
    assert num > 0;
    total += num;
}
```

From a design standpoint, assertions are not recommended for checking the validity of arguments. Use assertions for detecting internal errors. Use exceptions to detect the misuse of a method by the client programmers.

2. Name three types of assertions?

*precondition assertion
postcondition assertion
control-flow invariant*

3. Why is the following code wrong?

```
public void getData( ) {  
    String str = JOptionPane.showInputDialog(  
                                null, "Input:");  
  
    assert str.equals("") : "Input Error";  
}
```

Similar to Question 1, assertions should not be used to detect the errors caused by the enduser. Use a simple control statement to check for the validity of input values. Assertions are for detecting internal logical errors.

8.7 Supervisor-Subordinate Design Pattern

No Quick Check Questions.

8.8 Sample Development: Keyless Entry System

No Quick Check Questions.