Week 4

Comment

Tutorial exercises should be done without a computer

1 XFiles

The following class maintains a collection of strings, all of which must begin with ‘X’.

```java
public class XFiles {
    public LinkedList<String> files;

    public void add(String newFile) {
        if (newFile.charAt(0) == 'X') {
            files.add(newFile);
        }
    }

    public void remove(String file) {
        files.remove(file);
    }

    public List<String> getFiles() {
        return files;
    }
}
```

1. Fix the above code, so it won’t throw any exceptions (do not use `try/catch`).
2. A class invariant is a condition that must always be true about an object’s state and/or behaviour. A class’ implementation must preserve the class invariant. Identify one invariant for the `XFiles` class. This should be a condition you can determine from the description of the class provided above.
3. Find two ways to break the invariant for this class. Describe how to fix each problem so that the class invariant cannot be broken.

Exceptions

Questions 2, 3 and 4 below make use of the following exception classes. It may be useful for you to draw a diagram to help visualise the inheritance relationships between the classes.

```
class E1 extends Exception {};
class E2 extends E1 {};
class E3 extends E1 {};
class E4 extends E3 {};
class E5 extends E3 {};
```

```
class F1 extends Exception {};
class F2 extends F1 {};
class F3 extends F1 {};
class F4 extends F1 {};
```

The code that follows is intended to illustrate concepts. Some details (for example, class declarations) have been omitted. In all exercises, assume `x` and `y` are member variables.

2 Exceptions Exercise

```
int x = 0, y = 0;
public void exercise1() {
    try {
        try {
            f();
            y+=10;
        } catch (F3 e) {
            x+=1;
        } catch (F1 e) {
            x+=10;
        } catch (F4 e) {
            x+=100;
        } catch (E5 e) {
            y+=1;
            throw e;
        } finally {
            x += 2;
        }
        y+=20;
    } catch (Exception e) {
        y+=100;
    }
}
```

Work out the values of `x` and `y` if the `exercise1()` method is called and `f()` throws the following types of exception:

1. `F2`
2. `F3`
3. `F4`
4. `E4`
5. `E5`
6. Does the `exercise1` method need a `throws` declaration? Why, or why not?

3 Exceptions Exercise - Extra

```java
int x = 0, y = 0;
public void exercise2() throws.. {
    try {
        y += g();
    } catch (E3 e) {
        x += 100;
    }
}
public int g() throws.. {
    try {
        f();
        y+=10;
        return 400;
    } catch (E1 e) {
        x+=1;
    } finally {
        x+=10;
    }
    return 1;
}
```

Work out the values of `x, y` if the `exercise2` method is called and `f()` throws the following types of exception:

1. `E1`
2. `E2`
3. `E3`

Note

The `throws` declarations for the methods `exercise2` and `g` would be different, depending on which type of exception `f()` is throwing.

4 Exceptions Exercise - Extra

```java
int x = 0, y = 0;
public void exercise3() throws.. {
    try {
        try {
            try {
                f();
                x+=100;
            } catch (F2 e) {
                x+=5;
                g();
            } catch (F1 e) {
                x+=1;
            } catch (E1 e) {
                y+=2;
                throw e;
            }
            y+=10;
        } catch (F2 f) {
            y+=100;
        }
    } catch (Exception e) {
        x+=1000;
    }
}
```

In the following, `g()` throws F2. Work out the values of `x, y` if `exercise3()` is called and `f()` throws the following types of exception:

1. `F3`
2. `F2`
3. `E1`
4. `F1`

5 Name

Suppose the `Name` class represents one person's name, consisting of a first and a last name. Assume that the relevant getter/accessor methods exist.

```
public class Name implements Comparable<Name> {

    private String first;
    private String last;

    public int compareTo(Name other) {
        // your code here
    }

    public boolean equals(Object o) {
        // your code here
    }

    public int hashCode() {
        // your code here
    }
    ...
}
```

1. Implement the `compareTo` method[1]. Remember that this method is defined by the `Comparable` interface[2]. Consider the first names, then if they are identical, move on to the last names. You may make use of `String.compareTo()`.
2. Implement `equals()`. Remember that this method is inherited from the `Object` class[3]. You may make use of `String.equals()`.
3. Implement `hashCode()`. You may make use of `String.hashCode()`.
4. Are your implementations of `equals()` and `hashCode()` compatible?
5. Can you identify two `Names` which are different, but still have the same hash code? Does this matter?

[1] https://docs.oracle.com/javase/10/docs/api/java/lang/Comparable.html#compareTo(T)
[2] https://docs.oracle.com/javase/10/docs/api/java/lang/Comparable.html
[3] https://docs.oracle.com/javase/10/docs/api/java/lang/Object.html