Tutorial 8

Comment

Tutorial exercises should be done without a computer.

A robust software system is able to run in a variety of conditions. Simply put, we want to design software such that "garbage in does not mean garbage out". How do we design a system to be robust? Unfortunately, there is no perfect answer. In very general terms, there are two basic approaches - defensive programming and programming by contract. This tutorial will take a look at both ways in the context of file I/O.

Question 1

Consider the following class that keeps track of the current time.

```
public class Time {
    private final int hour;
    private final int minute;
    private final int second;

    /**
     * Creates a new time object
     *
     * @requires hour, minute, seconds >= 0
     * @ensures 0 <= hour <= 23, 0 <= minute <= 59, 0 <= seconds <= 59
     */
    public Time(int hour, int minute, int second) {
        // ...
    }

    @Override
    public String toString() {
        String morning = hour < 12 ? "AM" : "PM";

        return String.format("%02d:%02d:%02d %s",
                hour % 12, minute, second, morning);
    }
}
```

a) Write a constructor that satisfies the postcondition given above.

b) Now suppose that the javadoc changed to the following:

```
/**
 * Creates a new time object
 *
 * @throws IllegalArgumentException
 * When the domain of each input falls outside of:
 * 0 <= hour <= 23, 0 <= minute <= 59, 0 <= seconds <= 59
 */
```

Re-write the constructor, this time using an illegal argument exception for invalid inputs.

c) Recall the contracts from a). Does your answer to b) satisfy these contracts?

d) Which of the following methods is more robust? In other words, which constructor allows more flexibility in terms of its inputs?

Question 2

Now consider the following methods used to `save` and `load` time objects from a file.

```
public void save(String filename) throws IOException {
    BufferedWriter writer = new BufferedWriter(
            new FileWriter(filename));
    writer.write(this.toString());
    writer.close();
}

public static Time load(String filename) throws IOException {
    BufferedReader reader = new BufferedReader(
            new FileReader(filename));

    String time = reader.readLine();
    String[] dayComponents = time.split(" ");
    String[] timeComponents = dayComponents[0].split(":");

    int hour = Integer.parseInt(timeComponents[0]);
    int minute = Integer.parseInt(timeComponents[1]);
    int second = Integer.parseInt(timeComponents[2]);
    if (dayComponents[1].equals("PM")) {
        hour += 12;
    }

    return new Time(hour, minute, second);
}
```

a) List three exceptions that the load method would throw for at least three different input files. **Hint:** Look for NumberFormatException's, ArrayIndexOutOfBoundsException's and NullPointerException's

b) Would the following approach be a good solution to the problem? What inputs would it successfully read that do not represent times?

```
public static Time load(String filename)
        throws IOException, BadTimeFormatException {
    BufferedReader reader = new BufferedReader(
            new FileReader(filename));

    try {
        return parseTime(reader);
    } catch (NullPointerException
            | ArrayIndexOutOfBoundsException
            | NumberFormatException exp) {
        throw new BadTimeFormatException(exp.toString());
    }
}

private static Time parseTime(BufferedReader reader)
                throws IOException {
    String time = reader.readLine();
    String[] dayComponents = time.split(" ");
    String[] timeComponents = dayComponents[0].split(":");

    int hour = Integer.parseInt(timeComponents[0]);
    int minute = Integer.parseInt(timeComponents[1]);
    int second = Integer.parseInt(timeComponents[2]);

    hour += dayComponents[1].equals("PM") ? 12 : 0;

    return new Time(hour, minute, second);
}
```

c) Fill in the following table with checks to be added to the `load` method, such that it will throw a `BadTimeFormatException` when the input does not match the specification given by the Time class. The first row is given as an example. **Hint:** Remember the exceptions thrown in a) and the invalid times from b).

Issue	What to check
Negative times	`hour < 0 || minute < 0 || second < 0`