CSci 1933 Lab 4

October 06, 2015

1. Introduction

The purpose of this lab is to get experience with both using Exceptions, and handling Exceptions when they are thrown. In particular, the aim is to familiarize yourselves with scenarios when using an Exception is useful, and what to do in cases when you encounter code throwing Exceptions.

2. Getting started

2.1. In today's lab, you will implement a simple course registration system. You will modify the provided class UniversityClass, which provides an addStudent() method which manages the list of students enrolled in a class at a University. In building out this system, you will will modify addStudent() as well as write some code to handle your modifications in the main() method of UniversityClass.

2.2. In order to get started for this lab, do the following:

2.2.1. In IntelliJ, create a new Java project, called Lab4.

2.2.2. Download the following files, and import them into your project.

- ➢ UniversityClass.java - import this into your src/ directory.
- ➢ ClassFullException.java - import this into your src/ directory.
- ➢ Student.java - import this into your src/ directory.

3. Your task

3.1. You'll be implementing a simple course registration system in this lab. For example, when you are registering for courses at the University of Minnesota, if a course is full, you're given the option to add yourself to the waitlist of the class, in case another student drops the course. For the purposes of this lab, we don't want to give the student a choice, but instead assume that if they're attempting to add the course, they would like to be on the waitlist in case a spot opens up in the course. Additionally, if both the class and the waitlist are full, the student simply is notified of that fact and told they cannot register.

3.2. One of the ways that java provides error handling is to use Exceptions. As you can imagine, error handling is very important in a course registration system, like the simple one you'll be implementing today. In order to begin to understand how an Exception might be used, in this lab you will:

3.2.1. modify the addStudent() method so that:

- rather than returning a boolean value, it does not return anything if it succeeds.
- but throws a ClassFullException if the student cannot be added.

3.2.2. modify the main() method to add more students to the course, so that all cases of this program will be hit.

3.2.3. modify the main() method in order to properly handle the Exception that will be thrown from addStudent(). This entails:

- use a try/catch block to properly handle the Exception
- handle the exception by calling the addStudentToWaitlist() method, which tries to add the student to the waitlist
- implement the addStudentToWaitlist() method that:
  - ★ attempts to add the student to the waitlist

    Hint Add the waitlist array as a Student array field in the UniversityClass class. The constructor should instantiate the array.
  - ★ The size of the waitlist should be 4. Modify the constructor of UniversityClass such that size of the wait list is definable in the constructor of UniversityClass.
  - ★ if the student was successfully added to the waitlist, print out Added student X to the waitlist where X is the name of the student
  - ★ if the student could not be added to the waitlist, print out Sorry, the class and the waitlist are full. X cannot be added. where X is the name of the student

3.2.4. Your program will demonstrate(via output) that that:

- it can add students to the class
- it can add students to the waitlist
- it will notify the student if both the class and the waitlist are full

3.2.5. Your program will produce output similar to (but not necessarily identical to):

```
Added Dianne to the course
Added Kevin to the course.
Added Cliff to the waitlist.
Added Jonathan to the waitlist.
Sorry, the class and the waitlist are full. Elizabeth cannot be added.
```

4. One more step

4.1. Let's do a quick refresher on some concepts we've learned in the past, namely arrays, loops, and toString(). Implement a toString() method in UniversityClass that returns a String that looks as follows:

- The first line contains the course name, the number of students currently enrolled, and the maximum number of students who are allowed to enroll
- The next line is "Currently enrolled students:"
- The next series of lines consist of students that are enrolled (indented by 1 tab)
- If there is at least one student on the waitlist, continue. Else, return the string.
- The next line is "Students on waitlist: "
- The next series of lines consist of students that are on the waitlist (indented by 1 tab)

4.2. Remember that toString() methods are supposed to output very readable text, so format things accordingly!

4.3. Challenge problem: Can you figure out how to do this with exactly 1 loop in code?

Hint Think about a method you can write…

---END---

---Solutions will be posted on moodle page 30 minutes before the lab completion time---