CSE 12: Programming Assignment 1

10-08-2020

Focus: PA1, JUnit

PA 1 Description & Requirements

https://github.com/CSE12-W21-Assignments/cse12-wi21-pa1-Testing-starter

You are working for a Web shopping company where you need to create shopping cart functionality to keep track of items before a customer checks out.

- You are given 13 different implementations of **`Basket`**
- Your job is to write JUnit tests to cover the potential issues that might occur in the implementations

Note: All of your code will be written in **`BasketTest.java`**. DO NOT change any other file.

PA1 Overview

- Part 1: Write your tests for the `Basket` implementations
- Part 2: Answer the questions in the writeup on a Gradescope assignment

  Style will not be graded but comments will be added. Future assignments will have style points!

  How will my code be graded? Based on coverage. For this first assignment all tests will be visible on Gradescope, however, don't get used to this!

  Submission: only submit your `BasketTest.java` file to the code portion of this PA

EDIT: Winter 2021, there is an additional Part 3 to the assignment.

Why is this important? How does this relate to me?

- Releasing a product
- Specific industry jobs
- Future classes
- Overall: code should be stable

Unit Tests

Given a class `Student` with the method `addCourse` that adds a `Course` to a student's course list, `courses`.

What are some unit tests we can create to make sure any given implementation is correct?

Class `Course`

`Course(String name)` **Constructor** that creates a course specified by its name.	
`String`	**`getName()`** Returns name of the `Course` object.
`boolean`	**`isEqualTo(Course other)`** Returns whether or not a `Course` object's name is the same as another.

Interface `Student`

`String`	**`getName()`** Returns name of an object of a class that implements the `Student` interface.
`Course[]`	**`getCourses()`** Returns course list of an object of a class that implements the `Student` interface.
`void`	**`addCourse(Course course)`** Adds a course to a course list of an object of a class that implements the `Student` interface.

Class `StudentX` (implements `Student`)

`StudentX(String name, Course[] courses)` **Constructor** that creates a student specified by its name and course list.

What are some test ideas for `addCourse`?

- Make sure list contains the added course after
- Make sure list length increases by 1
- Test capacity bounds
- Doesn’t affect previous information
- Check for duplicates
- Make sure added course is actually a course

2 implementations of `addCourse`

```
public void addCourse(Course course){
      int size = courses.length + 1;
      Course[] newCourseList = new Course[size];

      for(int i = 0; i < courses.length; i++){
            if(course.isEqualTo(courses[i])){
                  return;
            }
      }

      for(int i = 0; i < size - 1; i++){
            newCourseList[i] = courses[i];
      }
      courses = newCourseList;

      return;
}
```

```
public void addCourse(Course course){
      int size = courses.length + 1;
      Course[] newCourseList = new Course[size];

      for(int i = 0; i < courses.length; i++){
            if(course.equals(courses[i])){
                  return;
            }
      }

       int i;
      for(i = 0; i < size - 1; i++){
            newCourseList[i] = course;
      }
      newCourseList[i] = course;
      courses = newCourseList;

      return;
}
```

Are there any more tests we should add?

Some things to note about each implementation (from previous slide)

The first implementation creates a new array that is one element longer to include the new course, but forgets to add the new course at the end (after the end for loop).

The second implementation uses the `equals` method to compare two courses, which would check the *references* of the two objects - it would test if both objects are the same object, not whether the course names are the same (the `isEqualTo` method in the `Course` class is for this). Additionally, notice the logic error in the end for loop: it fills the new course list with the new course at each element! It does not retain whatever courses previously made up the course list.

*Note: `assertEquals` calls the equals method inherited from the `Object` class. Keep in mind that some classes, like `String`, override this such that the method *does* check for character by character equivalence as opposed to checking equality between two objects' references.

Source code - JUnit testing example

Given 3 versions of the `Student` interface in classes `StudentA`, `StudentB`, and `StudentC`, we can implement unit tests to check for functionality. These will be added to the `StudentTest` class (similar to the `BasketTest` class in pa1).

Note: All code will be posted as well as a skeleton version of `StudentTest` so that you can try writing your own

Tips for PA1

- Review interfaces

  Review worksheet will be posted on the schedule
- Look at each implementation:
  - What are some possible logic errors that you might see?
  - What kind of test can catch these errors?