CS 455 Lab 5: Invariants/Processing lines

Spring 2019 [Bono]

Goals and Background

In this lab you are going to do a few tasks that will help you with programming assignment 2 (pa2), some written, and some coding. They involve three different issues: (ex. 1) assert statements (2 & 3) representation invariants and (4) reading a bunch of numbers on one line. Because the different parts are about different issues, the readings and reference material links below mention what exercise they pertain to.

A note about lab partnerships: This is a reminder that pa2 is to be done individually. In this lab you may still work with a partner, because for the most part you will not be writing code that you will directly be using in pa2. However, if by your lab section time you have already done more on your own on pa2 than what is asked for in Ex 3 and/or 4 here, you should work on your own for this lab, and you you may use your pa2 solution-in-progress to satisfy those exercises. Similarly, for students not in that situation, once you complete this lab with your partner, if you wanted to go further working on pa2 during lab, you would do that individually rather than with your lab partner. If you are working with a partner, remember to switch roles with your partner from what you were last time. For more information on lab partnerships, see Lab 2.

Prelab

Before the lab read over the whole programming assignment description (linked in the next section), paying particular attention to the subsections with their own links below.

Reading and reference material

- Programming assignment 2 (exercise 1 - 4) and these parts in particular:
  - Bulgarian solitaire game (exercise 2)
  - SolitaireBoard: representation/implementation (exercises 2 & 3)
  - Representation invariants (exercises 1, 2 & 3)
  - Converting a String into an array of numbers (exercise 4)

- Horstmann, Special Topic 11.6 covers assert statements (exercise 1)
- Horstmann, Section 11.2.5 Scanning a String (exercise 4)
- Horstmann, Section 7.7.4 Wrappers and Auto-boxing (exercise 4)

- CS 455 relevant lecture topics this week: preconditions, assert statements (exercise 1) and invariants (exercise 3)

Exercise 1 (1 checkoff point)

The code for a Term class can be found in your Vocareum Lab 5 workspace in the file `Term.java`

It stores the coefficient and exponent that make up a single term in a polynomial. It has a precondition on one of it constructors. The constructor uses an assert statement to check the precondition. Write a little test program, `AssertTester.java`, to force the assertion to fail. Your test program should also include calls to the constructor that don't make the assertion fail. The purpose of this is to make sure you know how turn on assertion checking, and to see what happens when one fails.

Run the program two times, once with and once without assertion-checking enabled. To enable assertions you use the `-ea` flag with `java` (the run-time system); otherwise, assertions are ignored. You do not have to compile the code any special way.

Question 1.1 Besides the name of the method called, what information in the exception error message can help lead you to the exact location of the failure?

To get checked off show the TA the code running with the assertion failure. DEN students should put a copy of the message they get when the assertion fails in their README file.

Exercise 2 (1 checkoff point)

The first question is to demonstrate your understanding of how Bulgarian Solitaire works. The game is described in the assignment section of the pa2 writeup.

Question 2.1 On paper (or in README) complete the following example of a Bulgarian Solitaire game (the same example that's in the assignment description. Show what would be on the board after each round (number each round). We're starting it out for you:

```
initial: 20 5 1 9 10
after round 1: 19 4 8 9 5
```

The rest of this exercise concerns your `SolitaireBoard` representation. You will need to have read the section on SolitaireBoard: representation/implementation for pa2 before doing this part.

Question 2.2 On paper (or in README) write down the variable declarations for the instance variable(s) for your `SolitaireBoard` class.

Question 2.3 On paper (or in README) for each of the following, either (1) show a diagram of what the corresponding SolitaireBoard will look like internally, including array indices, values, capacity and size (you don't have to show every last element, but may use ". . ." in places). To say it another way, what will the inside of a SolitaireBoard object look like for that board configuration. Or, (2) if that sequence is not a legal board configuration, say so:

1. 20 5 1 9 10
2. 45 1
3. 45
4. 1 1 1 1 1
5. 1 1 1 . . . 1 `[repeated 45 times]`
6. 100 -55
7. 1 0 44

Save your answer, because you may want to use some of the above as some of the test cases for your BulgarianSolitaireSimulator.java. later.

Exercise 3 (1 checkoff point)

You should read the pa2 section on Representation invariants before doing this part. The textbook doesn't discuss representation invariants, but we discussed the concept and went through a few examples of representation invariants in lecture this week.

Write the representation invariant for the SolitaireBoard class representation you are using. This would take the form of a list of conditions that must be true about your representation for the SolitaireBoard methods to work properly when you implement them. These conditions will involve the instance variables you supplied in Qn. 2.1; like with preconditions, you don't include things that are checked by the compiler, such as the type of an instance variable. We would assume this list is "and"-ed together, but you could also put in some "or" conditions by stating that explicitly. You want the condition to be as specific as possible.

All of your answers that you declared as legal from Question 2.3 should satisfy the invariant you write.

Save your answer, because you will be working more with this invariant for pa2, in particular, putting it in a comment in your code, and writing a method to test it.

Exercise 4 (1 checkoff point)

Part of pa2 involves your reading an indeterminate number of numeric values from one line of input. The end of that line is the sentinel; that is, the end of line indicates the end of the list of values. The code you have to write for the assignment itself will be a little different than this, but here we're going to write a similar test program to make sure we can get this type of code to work.

The section about this in the assignment gives more information about how to write code to do this.

Here we're going to write a little test program to test out the technique. Call this program `ReadTester.java`. Structure it as an infinite loop (so you'll have to exit the program with ctrl-c). For each loop iteration, print a prompt for the user, and then read all the data from the line into an ArrayList, and then print out the whole ArrayList. Running it might look something like this (user input shown in bold):

```
Enter a space separated list of numbers: 1 3 5
The numbers were: [1, 3, 5]
Enter a space separated list of numbers:
The numbers were: []
Enter a space separated list of numbers:  7
The numbers were: [7]
```

Unlike the input for pa2, for this lab you are not required to error-check that all the values entered were integers. Some hints:

- Remember that if you want an ArrayList of a primitive type element, you have to use a wrapper class. E.g., you define `ArrayList<Integer>` (not `int`). Java autoboxing takes care of wrapping and unwrapping your ints (this was discussed in more detail in section 7.7.4 of the textbook).

- Many Java classes have a `toString` method that converts them to a String form. When you put such an object in a `println` statement, if `toString` is defined for that class it automatically gets called so the String gets printed. (Usually in a format suitable for debugging print statements.) So for example, the following line of code will print a whole ArrayList:

  ```
  ArrayList<...> myArrList;
  ...
  System.out.println(myArrList);
  ```

  (If you want to know more about `toString` see section 9.5.1 in the textbook -- you don't need to know that material to do this lab problem, however.)

To get the checkoff point, show the TA your source code, and show the program running on various numbers of ints on the line with varying amounts of spaces before, between and after them, and including a case where there are no ints entered on the line (i.e., just a "return" -- the second case shown in the example interaction above).

Checkoff for DEN students

Make sure you put your name and NetID in all the files you submit.

When you click the `Submit` button, it will be looking for and compiling (for source code) the files `README`, `AssertTester.java`, and `ReadTester.java`.