Learning Outcomes

- Implement missing methods in a stack made with an array data structure
- Consider and compare the implications of using different implementations of stacks
- Gain experience in creating computer simulations of a real word phenomenon in Java

Pre-Lab

- Create a new Java project called Lab5
- Download the files: StackADT.java, ArrayStack.java, EmptyCollectionException.java, TestStack.java, HockeyTeam.java, HockeyEvent.java, and HockeyGame.java
- Save these downloaded files into the Lab5 src folder

Exercise 1 – Completing the Array Stack class

1. Open ArrayStack.java and TestStack.java and examine the code in both classes.
2. Note that ArrayStack is not complete; most of its methods are left empty. Without adding any additional instance variables or methods or changing any of the provided code, complete the six empty methods given the following instructions:
    a. `push(T element)` – if the array is full, call `expandCapacity()` (which is provided). Add the given element to the top of the stack.
    b. `pop()` – if the stack is empty, throw an EmptyCollectionException. If not, remove and return the element at the top of the stack.
    c. `peek()` – if the stack is empty, throw an EmptyCollectionException. If not, retrieve the element at the top of the stack and return it without removing it.
    d. `isEmpty()` – return true if there are no elements, and false otherwise.
    e. `size()` – return the number of elements in the stack.
    f. `toString()` – if the stack is empty, return "The stack is empty." Otherwise return a single-line string starting with "Stack: " and then containing all the elements of the stack from the top to the bottom. There must be a comma and space between elements but the last element must end with a period instead. i.e. Stack: 1, 2, 3, 4, 5.
    Hint: how can you determine when the loop has reached the last element?
    g. NOTE: In this implementation, the default value of top is 1 rather than 0. This is different than the example shown in lecture. Do **not** change this default value, but rather think about how to adjust these methods to work with this value of top.

3. Run the TestStack file. This will test that the ArrayStack methods were properly implemented. If any of the test failed, fix the corresponding methods in ArrayStack until all the tests pass. You may add print lines in ArrayStack to help with the debugging. Do not add print lines in TestStack or alter this file in any way.

Exercise 2 – Simulating Hockey

When trying to model a sport, it's important to consider numerous metrics or measures of performance. By studying these, coaches, fans, statisticians and more try to figure out what can be improved to get the largest changes in importance. One approach for seeing what changes make is to change parameters in a simulation.

In this lab, we will focus on a simplistic simulation of a hockey game. We have made a method for you in HockeyGame.java called rollDice(), which will return an int between 1 and 100. We will use this to simulate two teams playing against each other. In Hockey, players can pass the puck between each other, shoot, or lose the puck to the other team. Here, you will print out the actions of the game as they are simulated

You will need to write code that implements the following scenarios when HockeyGame.java is run. An example of all events to be pushed onto the team's stack are included with the lab code:

- Stage 1: Loose Puck
    - Call rollDice(), if result is 1-50, team 1 (stack 1) gets the puck.
    - Else (51-100) team 2 (stack 2) gets the puck.
    - For the appropriate team's stack, push a node onto the stack.

- Stage 2: Puck acquired or Pass Received
    - Call rollDice() and then do the following:
        - 1-10, return to stage 1 (Team loses puck, Empty stack)
        - 11-20, other team intercepts puck, return to Stage 2 with other team. Empty stack.
        - 21-80, Team passes successfully, return to Stage 2 for same team. Record a pass.
        - 81-90, Team shoots but doesn't score. Proceed to Stage 1 & empty stack. Record a shot.
        - 91-100, Team shoots and scores! Add a goal for Team. Proceed to Stage 1 & empty stack. Record a shot.

- Your program should print out:
    - All actions taken and by which team,
    - the number of shots taken by each team,
    - the number of passes made by each team, and
    - the final score.

Your final line should look something like (with different numbers most likely):

FINAL SCORE

TORONTO with 3 goals and 5 shots and 9 passes

VS

MONTREAL with 2 goals and 2 shots and 1 pass

Questions:

1. Consider the possibility of using a Linked Stack instead of the Array Stack in this algorithm. Would the simulation's results be impacted by switching to a different Stack implementation? Which of these classes/methods would you have to modify if you were going to use the other Stack implementation?

2. Which methods in HockeyGame.java are static and which are non-static? Explain why the static methods are static, and why other methods are not. Could HockeyTeam.java and HockeyEvent.java call static methods from HockeyGame?

***Optional**:* A more developed simulation would include things like individual player objects, which may have different percentages for pass success, shot accuracy, etcetera. List 5 changes that would help make your simulation more helpful for a sports team.

Type your answers to these questions in the textbox on OWL in the Lab 5 submission page to be submitted along with the attached Java files.

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- ArrayStack.java
- HockeyGame.java