Assignment 4

Refer to Canvas for assignment due dates for your section.

Objectives:

- Develop test suites for data collections.
- Implement an ADT.

General Requirements

Create a new Gradle project for this assignment in your course GitHub repo. Make sure to follow the instructions provided in “Using Gradle with IntelliJ” on Canvas.

Create a separate package for each problem in the assignment. Create all your files in the appropriate package.

To submit your work, push it to GitHub and create a release. Refer to the instructions on Canvas.

Your repository should contain:

- One .java file per Java class.
- One .java file per Java test class.
- One pdf or image file for each UML Class Diagram that you create. UML diagrams can be generated using IntelliJ or hand-drawn.
- All non-test classes and non-test methods must have valid Javadoc.

Your repository should **not** contain:

- Any .class files.
- Any .html files.
- Any IntelliJ specific files.

Problem 1

For this problem, you will work with two ADTs, a mutable Stack of Integers, and a mutable Queue of Integers. The ADTs are specified in two Java interfaces, `iStack.java` and `iQueue.java`, which you can find in the lecture-code repo. Look for the starter_code > assignment4 folder.

Your job is to **write a collection of JUnit tests** that can determine whether **any** implementation of each ADT satisfies the respective ADT’s specification:

- If your tests run against a correct implementation of the ADT, all your tests must pass.

- If your tests run against an incorrect implementation of the ADT, at least one of your tests must fail.

Your challenge is to smartly design test cases that cover the expected behavior of the ADT; it is NOT about the sheer number of tests that you propose. You are only expected to write tests for **functionality specified by the ADT**. This means you do not need to include tests for equals, hashCode, or toString.

To get started, create a package called `problem1` in your project, then copy `IStack.java`, `IQueue.java`, and `EmptyQueueException.java` into the package. Next, create test files for each interface in the same way that you would create a test file for any other class. After this step, you should have two test files: `IStackTest.java` and `IQueueTest.java`.

General requirements:

- Define a single test object in each test file with the respective interface type. In `IStackTest.java`, you will need a test object like this:
  `IStack stack;`
  ...and in `IQueueTest.java`, you will need a test object like this:
  `IQueue queue;`
- Because interfaces don't have constructors, you will not be able to instantiate your test objects in the `setUp` method as you normally do. Just leave the `setUp` method empty and write your tests using the test objects as you normally would. **Important note:** without a concrete implementation of the ADTs, your tests will always fail, even if they are correct. This is expected and OK! Additionally, your Jacoco report will show 0% coverage for this problem. This is also expected and OK! You do not need to implement the ADTs, although you may choose to if you wish. If you do implement them, your implementations will not be graded.

How we'll grade this problem, at a high level:

- We have several additional implementations of the ADTs, some of which are correct and some of which are not.
- We will run your tests against our implementations and check to see that all of your tests pass for the correct implementations and that at least one of your tests fails for incorrect implementations.
- In order to run your tests, we will instantiate your test objects using our own implementations of the ADT. For example, we have a class called `IStackImpl` that is a correct implementation of `IStack`. Assuming you define your test object in `IStackTest` as required above, we will edit the `setUp` method to contain the following:
  `stack = new IStackImpl();`

You do not need to submit a UML diagram for this problem or provide any additional Javadoc.

Problem 2

Provide the design and implementation of a `Set`, as in the mathematical notion of a set. Here is the specification:

- `Set emptySet()`: Creates and returns an empty Set.
- `Boolean isEmpty()`: Checks if the Set is empty. Returns true if the Set contains no items, false otherwise.
- `Set add(Integer n)`: Adds the given Integer to the Set if and only if that Integer is not already in the Set.
- `Boolean contains(Integer n)`: Returns true if the given Integer is in the Set, false otherwise.
- `Set remove(Integer n)`: Returns a copy of the Set with the given Integer removed. If the given Integer is not in the Set, returns the Set as is.
- `Integer size()`: Gets the number of items in the Set.

Your implementation of `equals(Object o)` should return true if and only if the two sets have the same number of elements and, for every element in `this`, the same element exists in `o` and vice versa. Ensure that your implementations of `hashCode()` and `equals()` satisfy the contracts for both methods.

You may not use any built-in Java collections, other than arrays, as the underlying data structure. As the specification suggests, your implementation should be immutable.

Write a test class for your list of Strings and provide a UML diagram for this problem. The Jacoco test coverage requirements **do** apply to this problem—you'll need 70% for full credit, and 100% for maximum confidence in the correctness of your work.