Lab 6

Objectives

- Throwing checked exceptions
- Stack data structure
- Generics
- Queue data structure

Part1

In this part of the lab you will be implementing the Stack interface and adding exceptions in place of preconditions.

1. Download the files for Part1 of this lab into your Lab6 folder.
2. Implement each of the methods in `StackArrayBased.java` according to the documentation in the Stack interface.
3. Find the methods in the Stack interface that have the precondition that the Stack must not be empty. For each of these method, remove the precondition and change the method so it throws a `StackEmptyException` (provided for you) in place of the precondition. This will cause you to change your code in multiple places for these updated methods:
    a. Signatures of method prototypes in the interface
    b. Signatures of method implementations in the class that implement the interface
    c. Add code to throw the new exception in the correct case in the method implementations
    d. All calls to these methods (in the tester) must be wrapped in a try/catch block
4. Implement the methods `reverseString` and `doBracketsMatch` in `Lab6TesterPart1.java` according to the documentation and tests provided. You must use a stack in your implementation

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see you compile and run `Lab6TesterPart1.java` and run the tests.

Part2

In this part of the lab you will be implementing a generic Queue.

1. Download the files for Part2 of this lab into your Lab6 folder.
2. Change the interface (`Queue.java`) to be of generic type. This will force you to make changes in multiple places
   a. The class that implements the interface and its dependent classes (`QueueRefBased.java` and `QueueNode.java`) must be made generic
   b. Any code that creates a new `QueueRefBased` must now indicate the type of data that Queue will hold. This type must be the object type. For example:

```
Queue<Integer> intQ = new QueueRefBased<Integer>();
Queue<Character> charQ = new QueueRefBased< Character>();
```

   TIP: make changes to one class at a time, compiling/updating until no compilation errors.
   For example, start with `Queue.java`, repeatedly make the changes/compile until complete:

```
javac -Xlint:unchecked Queue.java
```

   Then move on to `QueueNode.java` and do the same, and finally for `QueueRefBased.java`.

3. Uncomment the code provided in the tester following the line:

```
//Testing generic queue with Character type
```

4. If you have updated your code in all of the necessary places it should compile without warnings.

   NOTE: if you get the warning like the following…

```
Note: Lab6TesterPart2.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Do as suggested and recompile as follows (it will indicate the line numbers of where the problem code is):

```
javac -Xlint:unchecked Lab6TesterPart2.java
```

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see you compile and run `Lab6TesterPart2.java` and with no warnings.

5. Complete the implementation of the stubs provided in `QueueRefBased`.java according to the documentation in the Queue interface and the tests provided.

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see you compile and run `Lab6TesterPart2.java` and that your tests pass.

Finished early – start your Assignment!