Programming Assignment: Stack/Queue, Linked List Implementation

Back

Objectives

- Gain familiarity with implementing the ADTs, stack and queue, using linked lists for underlying storage.
- Gain familiarity with raising exceptions.

As is typical, there are three implementation levels for the PA.

Minimal Level

For the minimal level, download the starting point code: stackqueue1.txt. This text file includes four simple interfaces: an unbounded stack, an unbounded queue, a bounded stack, and a bounded queue. Each has the expected mutator operations, which change the state of the object: push and pop for stacks and enqueue and dequeue for queues. There is a third operation, a query operation, peek. The peek method returns the same value as pop / dequeue would, but peek does not alter the state of the structure, as expected by a query operation. The mutator operations, pop and dequeue, change the state, because they remove the value which is returned.

These interfaces shall be in the **csc143.data_structures** package. Create the appropriate folder structure to support this Java package. Include JavaDoc comments for the interfaces and the methods within them.

Within the same package, csc143.data_structures, implement the interfaces. The short class names shall be the interface name with "_LinkedList" appended. That is, BoundedQueue_LinkedList, BoundedStack_LinkedList, UnboundedQueue_LinkedList, and UnboundedStack_LinkedList, listed alphabetically. Include JavaDoc comments for the classes, and the constructors and methods within them.

Notes about the implementations:

- ***Do not use any classes from the java.util package to implement the queue or stack***.
- The underlying data structure will be a linked list. This can (should) be structured based on the SimpleLinkedList presented in the notes.
- The classes will only have a single constructor.
  - The constructor for the unbounded implementations shall be parameter-less.
  - The constructor for the bounded implementations shall take a single int parameter, the maximum capacity.
- The data structures shall start out empty.
- The behavior of the operations is as expected based on the method names.
- Create a single Link helper class to be used by all four of these implementations. As a helper class, it shall be package private. For simplicity's sake, declare it in its own source code file.
- The behavior that violates preconditions is undefined for the Minimal Level. That is, it's fine if popping from an empty stack or adding to a full bounded queue causes an exception to occur. It would also be fine if these operations just gave unexpected results, like returned null or did nothing. (That's what *undefined* means.) This will be addressed in the Standard Level.

For more information about the standard behavior of the stack and queue, please refer to the Powerpoint slides. Another good source of information about stacks and queues is Wikipedia:

http://en.wikipedia.org/wiki/Stack_(data_structure)

http://en.wikipedia.org/wiki/Queue_(data_structure)

Test your implementations.

Collect the files for submission, both source code (.java) and bytecode (.class), into a Java archive (.jar) file which preserves the required folder structure. The contents of the .jar file should be usable on the classpath as submitted. Also, the contents of the .jar file should be compilable immediately after extraction from the archive.

Report

Write up a report about this programming assignment which addresses the following questions.

- How did you go about working this project?
- What works and what doesn't?
- How did you test your implementations?
- What the surprises or problems did you encounter while implementing this application?
- What is the most important thing(s) you learned from this portion of the assignment?
- What you would do differently next time?

The report does not have to be long, but it should show that your have thought about these questions. That's the value of the reflection. It improves your coding skills based on your own experiences.

Standard Level

For the standard level, support precondition checking. As implied in the Minimal Level write-up, the precondition failures happen when trying to read from an empty structure or add to a full one. Appropriate precondition checking entails communicating precondition failures back to the client code. The typical Java mechanism for this uses checked exceptions.

Include the following exception declarations into the csc143.data_structures package.

```
public class DataStructureException extends Exception {
    public DataStructureException(String msg) {
        super(msg);
    }
}

public class OverfillException extends DataStructureException {
    public OverfillException(String msg) {
        super(msg);
    }
}

public class UnderemptyException extends DataStructureException {
    public UnderemptyException(String msg) {
        super(msg);
    }
}
```

Include appropriate JavaDoc comments.

Update the interfaces in csc143.data_structures to support better precondition checking.

- The methods which may raise a checked exception must indicate this in the method header.
- "Responsible" precondition checking should include mechanisms to avoid the exceptions if practical.
- Update the interfaces with methods to avoid the exceptions. Use these method headers.

```
public boolean hasRoom();
public boolean hasItems();
```

We will discuss the rationale for these method names during class on Wednesday.

The classic stack and queue data structures are opaque. That is, the contents are unavailable, except at the working end(s) of the structure: the head and tail of the queue and the top of the stack. That's why operations like a size method are not included. It also actually simplifies the use (and implementation) of the data structure.

Needless to say, the contents of the Java archive shall reflect these changes.

Challenge Level

Update the stack and queue interfaces and implementations in csc143.data_structures to be generic.

Write a simple test application for the generic stack implementations. Here is the pseudo-code for main:

```
java.util.Scanner scan = new java.util.Scanner(System.in);
create a stack of strings.
String s;
for(int i = 0; i < 4; i++) {
   System.out.println("Enter a word: ");
   s = scan.next();
   push s onto the string_stack
}
System.out.println("Here are your words, reversed:");
while string_stack.hasItems() {
   System.out.print(string_stack.pop + " ");
}
System.out.println();
System.out.println();
create a stack of ints
int num;
for(int i = 0; i < 3; i++) {
   System.out.println("Enter a number: ");
   i = scan.nextInt();
   push i onto the int_stack
}
System.out.println("Here are your numbers, reversed:");
while int_stack.hasItems() {
   System.out.print(int_stack.pop + " ");
}
System.out.println();
```

Here is a sample run:

```
Enter a word: lions
Enter a word: tigers
Enter a word: bears
Enter a word: oh-my
Here are your words, reversed:
```

```
oh-my bears tigers lions

Enter a number: 1
Enter a number: 2
Enter a number: 3
Here are your numbers, reversed:
3 2 1
```

[fodder: What is the source of the four words used in the sample run?]

Name this test application: TestingGenericStack. It shall be in the anonymous package. Include this testing application as the entry point in the archive file submitted for the PA.

Deliverables

Two files, three files for challenge level:

- A .jar file of your data structures package (includes TestingGenericStack, for Challenge Level)
- An ASCII text file with the report

The .jar file should contain only the csc143.data_structures package, and TestingGenericStack, for the Challenge Level. Both source code (.java) and bytecode (.class) files shall be included in the archive. Classes or interface which do not have both source code and bytecode in the archive will be considered incomplete and not included in the grading.

Grading Rubric

Functionality:
4 points for Minimal Level:
- interfaces and classes in appropriate package
- compile and perform as expected
- implemented per write-up

7 points for Standard Level:
- Minimal Level requirements, plus
- support for preconditions
  * precondition checking implemented
  * given checked exceptions to communicate to client
  * appropriate query methods to avoid the precondition failure

9 points for Challenge Level:
- Standard Level requirements, plus
- support for generics in interfaces and implementing classes
- TestingGenericStack.java application, as entry point

Style: 3 points
- Conforms to homework guidelines

Documentation: 5 points
- Report addresses the required topics
- Comments: JavaDoc and method-internal

Back