Programming Assignment: Stack/Queue, Array Implementation

Back

Objectives

- Gain familiarity with implementing the ADTs, stack and queue, using an array for underlying storage.

Background

In the previous PA, Stack/Queue, Linked List Implementation, we implemented the stack and queue ADTs using a linked list as the underlying data structure. As the basis for that assignment, we created an initial core for the csc143.data_structures package: four (4) interfaces and three (3) exceptions, which describe bounded and unbounded stacks and queues. There were some slight modifications that were needed to support precondition checking and structured exception handling in the interfaces.

For reference sake, an archive showing the core interfaces and exceptions will be posted on Tuesday, after the previous PA can not longer be submitted for credit. The interfaces in this archive are generic, as needed for the Challenge Level in the previous PA. The files in the archive also lack JavaDoc comments.

In this PA, we will provide additional implementations of the four interfaces; this time using an array as the underlying data structure, rather than a linked list.

Minimal Level

Provide implementations of the interfaces in csc143.data_structures using an array as the underlying data structure. The new classes shall be named BoundedQueue_Array, BoundedStack_Array, UnboundedQueue_Array, and UnboundedStack_Array. They are all in the csc143.data_structures package.

Notes about the implementations:

- ***Do <u>not</u> use any classes from the java.util package to implement the queue or stack***.
- The underlying data structure will be an array. This can (should) be structured based on the SimpleArrayList 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.

For the Minimal Level, the implementation classes no not need to be generic.

Generics and Arrays

There is an issue that arises when using arrays in generics. The generic formal type, for example <E> in the provided interfaces, can be used in the declaration of an array reference, but it cannot be used to instantiate an array object.

That is, given a generic formal type <E>, the following is legal Java for a field declaration:

```
private E[] storage;
```

But, the following code will not compile:

```
storage = new E[10];
```

There is a relatively simple work-around. Let's use the example from the Generics notes:

Here is the generic class from the notes:

```
public class GenericCache<T> {

    private T stuff;

    public GenericCache() {
        stuff = null;
    }

    public void store(T s) {
        stuff = s;
    }

    public T retrive() {
        T s = stuff;
        stuff = null;
        return s;
    }

    public boolean hasStuff() {
        return stuff != null;
    }

}
```

Here is the interface analog:

```
public interface GenericCache<T> {

    public void store(T s);

    public T retrive();

    public boolean hasStuff();

}
```

To create a non-generic String-based cache, we could write:

```
public class StringCache implements GenericCache<String> {

    private String stuff;

    public StringCache() {
        stuff = null;
    }

    public void store(String s) {
        stuff = s;
        }

        public String retrive() {
            String s = stuff;
            stuff = null;
            return s;
        }

        public boolean hasStuff() {
            return stuff != null;
        }

    }
```

Notice that the StringCache type is not generic, There is no <T> following the name of the class, StringCache. There is a <String> which is filling in the generic formal type of the interface. That is, the class StringCache is implementing the interface GenericCache with the formal type <T> being replaced by the actual type <String>. That's why the store and retrieve methods are written with the type String as the parameter and return types, respectively.

So, coding non-generic implementations of the generic interfaces is relatively straight forward. That is what you'll do for the Minimal Level. For maximum flexibility, use the actual type java.lang.Object, that is fill in <E> with <Object>. The Standard Level is coding generic implementations of these interfaces.

BTW, the type parameter for GenericCache is T, capital "T", to stand for "type", since it represents a single value. In the stack and queue interfaces the type parameter is E, capital "E", to stand for "element", since the stack and queue represent collections of values. All of this is just convention, as is using a single capital letter for the type parameter.

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) for csc143.data_structures, 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, the stack and queue implementations in csc143.data_structures shall be generic. This is not too much work if you have working linked list implementations. (Yes, the Challenge Level work for the previous PA is relatively easy, if you have implemented precondition checking. In fact, you still have time to earn those points … if you read this early enough.)

As noted in the Minimal Level notes, the compiler has no problem with declarations of array references of the generic formal type, <E>:

```
    private E[] storage;
```

But, the following code will not compile:

```
    storage = new E[10];
```

For a full discussion, see the generic tutorial on the Oracle website. Because of this, you will need to find a work-around for instantiating the underlying array.

The work-around is to instantiate the array of a fully-known type. In this case, an array of java.lang.Object and cast the array to the desired type. This will cause a compiler warning. To remain in keeping with the homework guidelines, the warning message from the compiler should be suppressed using the annotation, @SuppressWarnings.

For complete Standard Level points, use this annotation sparingly. It should be applied to the most limited scope and as infrequently as possible. The goal should be to use the annotation only once per implementation class. That is, put the problematic instantiation into a private method marked with @SuppressWarnings. Then, call this method as needed within the rest of the code within the class. For more information, check the tutorial on annotations or ask in the forum.

The conceptual challenge for making the linked-list implementation generic can be addressed by a second use of the @SuppressWarnings annotation in each implementation class.

Challenge Level

For the Challenge Level, make the methods of the stack and queue implementations all constant time operations, O(k).

The linked list implementations are most likely already O(k) operations. More work / thought may be needed to implement the operations to be O(k) when the underlying data structure is an array.

Hint: Refer to the Wikipedia articles listed above for a technique to make the methods O(k).

Note:
It is not possible to make the constructors of the array-based stacks and queues O(k) operations. This is because the language specifies that the elements of an array are initialized at instantiation. Hence, the larger the instantiation, the more work involved in the instantiation. [fodder: What is the big-O for these constructors?]

[fodder: Is is possible to determine the big-O of the methods in the interfaces? Why or why not.]

Remember, you cannot get credit for [fodder:] responses if someone has already given the answer. If this encourages you to read the PA write-ups early …

Deliverables

Two files:

- A .jar file of your data structures package, csc143.data_structures
- An ASCII text file with the report

The .jar file should contain only the csc143.data_structures package. Both source code (.java) and bytecode (.class) files shall be included in the archive. Include both the linked-list and array implementations. 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:
5 points for Minimal Level:
 - generic interfaces with precondition checking (available in archive on Tuesday, if needed)
 - array-based implementations of stack and queue, both bounded and unbounded
 - implementations may be non-generic
 - automatic "growing" of the underlying array, as needed
 - compile and perform as expected
 - implemented per write-up
8 points for Standard Level:
 - Minimal Level requirements, plus
 - generic support in all classes
10 points for Challenge Level:
 - Standard Level requirements, plus
 - constant time, O(k), methods for the stacks and queues

Style: 3 points
 - Conforms to homework guidelines

Documentation: 4 points
 - Report addresses the required topics
 - Comments: JavaDoc and method-internal

Back