Lab 8 - Exploring Exceptions with Queues and Generic Arrays
Due (via Bitbucket) Thursday, 20 April 2017
50 points

Lab Goals

- Experiment with Exceptions
- Create an Exception-throwing Queue by implementing an interface
- Answer a few questions to test your knowledge of course material

Assignment Details

In this lab, we will explore and combine a few topics from the past few classes, and expand on a few of the topics we didn’t cover fully. We will implement a Queue based on a resizing array, and generate an exception when the user tries to take from an empty Queue. You will also answer a few short questions at the end.

Part 1: Implementing your own Exception (10 points)

We will be making a new Queue class in the next section, but first, we need to create a custom exception to throw in the event that something exception-worthy happens. To create our own exception, we will need to create a new class that directly or indirectly extends from Throwable. When creating custom exceptions, keep in mind that:

- If you want to write a checked exception, you should extend the Exception class.
- If you want to write a runtime exception, you should extend the RuntimeException class, or the Error class in certain situations.
- If you want to write an unchecked exception, you should extend the Throwable class, or one of its subclasses that are not also subclasses of Exception, Error, or RuntimeException.

In this instance, we want to create an unchecked exception, since we do not want users of our Queue class to have to write try-catch blocks around method calls. So, our exception should extend the Throwable class. Lastly, we need to create the constructor for our exception. Inside this constructor we will need to use super() to pass a string to our superclass’s constructor. This string should be a description of the exception.
Now we will test our exception. You will need to create another class to host a \texttt{main} method. Exceptions can be thrown using the keyword \texttt{throw}. This terminates the current method if there is no try-catch block to catch it, and throws the exception to the calling method. This chain continues until the exception either reaches a try-catch block, at which time it is caught and handled, or until it reaches the end of \texttt{main}, in which case the JVM crashes and you get an exception in the terminal. Now try out your new exception in \texttt{main} by following the steps below:

1. Throw a new instance of your exception with the \texttt{throw} keyword. Run the code – what happens?

2. Now make your exception extend \texttt{Exception}. What happens now?

3. Finally, change your exception back to extending \texttt{Throwable}, and surround the throw statement in a try-catch block. Now what happens?

**Part 2: Exceptions, Queues, and Generic Arrays (28 points)**

A few weeks ago, we saw an ArrayQueue implementation. However, that implementation was limited in both size and element type. Today we will fix both of these limitations, along with adding some exceptions. We will also implement an interface. As we discussed earlier this semester, interfaces provide an abstract base for a class, and tell the class what it must implement – what methods it must have. We will be using the \texttt{ExceptionQueue\langle T\rangle} interface, provided in the code directory. You will make a class that implements this interface, and provides all the methods specified, along with possibly one additional \texttt{resize()} method. Notice that the interface has a type variable (a generic). Your class must also have a type variable in order to know what kind of elements it should store. We will also be using a generic array, which allows constant access time for all elements while keeping the generic element ability that is useful with other data structures. The methods you must implement are detailed below:

1. A constructor – you will need to create a constructor that initializes your backing, generic array. To do this, you can create an instance variable that has the type \texttt{T[]} if your type variable is \texttt{T}, and then create a new array of type \texttt{Object} and cast that \texttt{Object[]} to a \texttt{T[]} now you should set your instance variable to the result. Zac and Saejin, your teaching assistants, can help with this step if needed.

2. \texttt{add()} – you will need to create an add method that correctly adds an item to the Queue. Remember to use your type variable in the parameters! Here is also where you will need to figure out if you need to resize the backing array, and if so, resize it (you could create a helper function for this). This method should have an amortized time complexity of \texttt{O(1)}.

3. \texttt{peek()} – just as in regular Queues, you will need to implement a \texttt{peek()} function that will return the front of the Queue. Remember that this is where you may need to throw an exception – what if there is no front to return? This method should have a time complexity of \texttt{O(1)}.

4. \texttt{remove()} – you will need to create a remove function that will both remove the front item from the Queue, and return it. This also may need some exception handling. You could also
consider testing if you should shorten the backing array here, to save memory if your array is too big. Remember that you do not want to resize the backing array too often, though! This method should have an amortized time complexity of $O(1)$.

5. `size()` – this function should return the current size of the Queue. It should run in constant time.

6. `empty()` – just as with `size()`, this method should run in constant time. It should return true if the Queue is empty and false if it is not. This could be useful in checking if you should throw an exception!

Remember that you should write some code to test your ExceptionQueue implementation. Another class with a main method could be utilized. Just because your code compiles does not mean that it is correct; ensure that your custom exception is being thrown, and that your Queue behaves as it should when adding and removing items. Also remember to test resizing!

**Part 3: Additional Questions (12 points)**

Please answer the following questions thoroughly:

1. In your own words, what is the difference between a checked exception and a runtime exception? Is one better than the other in any way?

2. Draw the 13-entry hashtable that results from using the hash function $h(i) = (3i + 5) \mod 13$ to hash the keys 12, 44, 13, 88, 23, 94, 11, 29, 20, 16, and 5, assuming collisions are handled by separate chaining.

3. Given a set of $n$ values to insert, what is the worst-case height of a binary tree? Draw a tree that demonstrates this case.

4. Given a set of $n$ values to insert, what is the best-case height of a general (not necessarily binary) tree? Draw a tree that demonstrates this case.

**Submission Details**

For this assignment, please submit the following to your cs112s2017-<your user name> repository (and ensure that the instructor has access to your repository):

1. Your answer to the questions in Part One.

2. The source code for your custom exception from Part One.

3. The source code for your ExceptionQueue implementation from Part Two.

4. The source code for any testing done in Part Two.

5. The answers to the questions from Part Three.