Free Quackers

```
public interface QuackInterface <T>
{
    public void  pushFront (T item);

    public void  pushBack  (T item);

    public T     popFront  () throws EmptyQuackException;

    public T     popBack   () throws EmptyQuackException;

    public T     front     () throws EmptyQuackException;

    public T     back      () throws EmptyQuackException;

    public int   size      ();
}
```

It seems you are quickly becoming a master of efficiency. After their agents' gasoline consumption fell by 99%, *Pen-Fifteen* was sufficiently impressed. They need another favor.

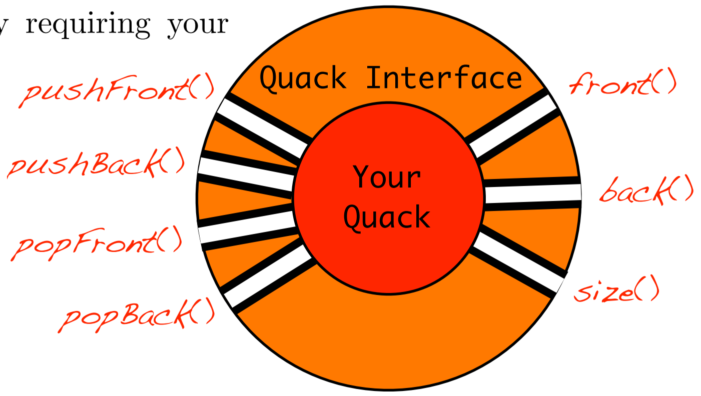
Quacks probably look something like this...

The message you found in the bathroom this morning mentioned an "Operation M." You don't know what it is, but that's just the way things are. Whatever it is, it begins next Wednesday. And for it, *Pen-Fifteen* is going to need a very strange data structure: a *quack*. Half queue, half stack, and the third half duck, it's a multi-headed beast to put Fluffy to shame.

or this...

This time though, *Pen-Fifteen* has discovered a feature of Java to aid with their clandestine practices: *interfaces*. By requiring your class to conform to an interface, they can control what information goes into and out of your quack. And they can easily substitute another in its place should you fail.

Needless to say, that doesn't mean you should.



To: Agent X
From: Pen-Fifteen
Subject: Quacks & Stueues

This assignment is a bit different. You will be writing your own class entirely from scratch. It won't be easy (unless it is), but we promise it will be fun. Before anything, carefully read through QuackInterface.java. These are the methods your Quack must provide, so make sure you understand them.

To begin, create your own *Quack* class (in a Quack.java file of course) that implements *QuackInterface*. Like this!

```
public class Quack <T> implements QuackInterface<T>
{
    // Your code goes here!
}
```

Before you can to compile, your class needs placeholder code for each of the 7 methods in *QuackInterface*. This can be as simple as copy-and-pasting all the methods from QuackInterface, replacing the semicolons with brackets, and throwing in a "return null;" or "return 0;" when necessary.

Once you code compiles, add a main() method that you can use as a sandbox to test the methods you write. And dive in! You can implement the 7 methods in whatever order you like, just keep in mind that your job will be much easier if you test things as you go rather than expecting everything to magically work at the end.

When you think you're finished, you should test your code against the rigorous battery of tests we have prepared. Compile and run *QuackTester* to do this. In the unlikely event that your code is perfect, you should see this:

```
.....................
Time: 1.234

OK (20 tests)
```

In the more likely event that it isn't, find and eliminate all the bugs.

Finally, when your *Quack* has been thoroughly debugged, it is time to enter the arena. Use *QuackTimer* to measure the speed of your *Quack* implementation on a large numbers of random operations. Make sure you can get up to 100,000,000 operations in a reasonable amount of time, because this is what will be tested. When you're ready, run the "submit" program (by opening it or by typing "./submit" into terminal). You can run this program as many times as you want, since only your highest score will be used. Try to find the fastest computer you can for maximum advantage!

Good luck and have fun! And please, don't make yourself dispensable.

Grading

Algorithms		
Simple test cases	(5)	
Compound test cases	(5)	
Exception test cases	(5)	
Code Quality		
Neatness, consistency, readability	(5)	
Extra Credit		
Performance competition	(5)	<-- 5 for first, 4 for second, etc...*
Add a reverse() function that runs in O(1) time	(5)	<-- Yes, this is a trick question. Yes, it actually can be done.

* A few extra rules:

- You may not modify QuackTimer.java in any way. =)
- If you use an array, no cheesy solutions like starting your array with a huge capacity! It must start with capacity 1 and it can only double every time it resizes.

Super Duper IMPORTANT Things

Talk through the following with your partner and ask me about about any you don't understand. These aren't graded because that would be annoying for all involved, but not being able to answer any of them indicates some conceptual misunderstanding which is in your (and your family and grade's) best interest to resolve.

1. What happens if you try to compile Quack.java before implementing all of the QuackInterface methods?

2. You don't have to understand how QuackTester.java works in gritty detail, but try to figure out where it's getting its test cases from. Add one of your own!

3. How could you make you Quack conform to both the Queue and Stack interfaces as well?