Exercise 1

The following Java class shows an implementation of queues in Java.

```java
public class Queue {
    protected int in, out;
    protected Object[] buf;

    public Queue (int capacity) {
        buf = new Object[capacity];
    }

    public boolean empty() {
        return in - out == 0;
    }

    public boolean full() {
        return in - out == buf.length;
    }

    public void enqueue(Object o) {
        buf[in % buf.length] = o;
        in++;
    }

    public Object dequeue() {
        Object o = buf[out % buf.length];
        out++;
        return o;
    }
}
```

(i) Give reasonable pre- and postconditions (in first-order logic “syntax”) for all methods and the constructor of the Queue class. In particular, keep in mind that integers may overflow.
(ii) A *weak class invariant* is defined as a condition that holds between calls to methods of the class, but not during the execution of such methods. Are there any weak class invariants for the Queue class?

**Exercise 2**

Regard the *Test Quiz* shown in the lecture. You will have a simple program reading four integers from the command line. Each value represents the length of one side of a quadrangle *(A-B-C-D)*. The program will tell you whether the input describes a valid quadrangle or not and will divide the quadrangle in one of the following groups.

- **square** four equal sides
- **rectangle** two pairs of equal opposite sides
- **kite** two pairs of equal-length sides
- **quadrangle**
- **invalid quadrangle**

Create a set of *Test Cases* to verify the functionality of this program. Treat special cases and permutations of the input as well as overlappings.

**Exercise 3**

In the previous exercises, we have examined the specification of programs using pre- and postconditions. In this exercise, we consider the use of examples for explaining the behavior of a program. To this end we will use Pex, a tool from Microsoft Research that creates a set of test cases by analysing the source code. We will see that it is usually harder to understand the semantics of a program if a set of test cases is given instead of a specification. Familiarize yourself with Pex4Fun at [http://www.pexforfun.com/](http://www.pexforfun.com/). Provide code that matches a secret implementation. Test your solution by asking Pex. Pex either returns true if your solution is correct, or provides a counter-example for parameters for which your solution fails.

1. Provide code that matches the implementation of *Puzzle* at [http://goo.gl/t5SPC](http://goo.gl/t5SPC). What does *Puzzle* compute? *Hint:* Consider the triangle example discussed in the lecture.

2. Provide code that matches the implementation of *Puzzle* at [http://goo.gl/SZVZS](http://goo.gl/SZVZS). What does *Puzzle* compute?