Eric Wilson
Java Lab #2, Due 20150908

1. In this code, when the stringBuffers array is declared, each individual StringBuffer within it is initialized to `null`. Therefore, there is no value within `stringBuffers[i]` to which our concatenated string can be appended. We can remedy this by initializing each `StringBuffer` in our loop before appending the string:

```
public class Lab2 {
    public static void main(String[] args) {
        StringBuffer[] stringBuffers = new StringBuffer[10];

        for (int i = 0; i < stringBuffers.length; i ++) {
            stringBuffers[i] = new StringBuffer();
            stringBuffers[i].append("StringBuffer at index " + i);
        }
    }
}
```

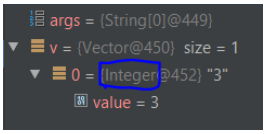
2. I chose to use a vector as opposed to an arrayList in part B because it is not specified what the integers will be used for. Since a vector is threadsafe, it would be useful if the integers were used in a multithreaded application. I also made my Vector of type Integer since we're reading in integers. I think it's safe to assume that a program that reads in integers intends to use them as integers and wouldn't want to convert them from strings all the time.

```
public class Lab2 {
    public static void main(String[] args) {
        BufferedReader consIn = new BufferedReader(new InputStreamReader(System.in));
        Vector<Integer> vector = new Vector();

        System.out.print("Type series of integers following each with \'Enter\'\n");
        try {
            String str = consIn.readLine();

            while(!str.equals("")){
                vector.add(Integer.parseInt(str));
                str = consIn.readLine();
            }
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        catch(NumberFormatException n){
            n.printStackTrace();
        }
    }
}
```

For the second part of this question the code

```
Vector v = new Vector();

v.add(3);
```

works because, firstly, the vector is not typed so it can accept any object you decide to add to it. Secondly, java automatically wraps the int variable in the built in `Integer` class as seen in this screenshot of my debugger.



C. While a true memory leak (memory allocation present even after program is terminated) is not, to my knowledge, possible in Java, I believe I have come pretty close.

```java
import MemTrix.MemLeak;

public class Lab2 {
    public static void main(String[] args) {
        MemLeak mem = new MemLeak();
        while(true){
            mem = new MemLeak(mem);
        }
    }
}
```

```java
package MemTrix;

public class MemLeak {
    MemLeak last = null;
    String [] s = new String [10000];//the meat and potatoes of our memory consumption.
    public MemLeak() {    }

    public MemLeak(MemLeak oldMem){
        last = oldMem;
    }
}
```

```
"C:\Program ...
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at MemTrix.MemLeak.<init>(MemLeak.java:8)
    at Lab2.main(Lab2.java:11) <5 internal calls>

Process finished with exit code 1
```

D.

- Bicycle:
    - assumptions:
        - There will be multiple bicycles.
        - increaseGearSetting() and decreaseGearSetting() are simple increment and decrement.
    - All int variables (frameSize, numberOfSpeeds, currentGearSetting) will be non-static since they will vary by instance (different bikes have different characteristics).
    - increaseGearSetting() and decreaseGearSetting() will be static since they will operate the same independent of the bicycle.
        - Assume that increaseGearSetting() can stop at each Bicycle's maximum gear setting by accessing each instance's nonstatic numberOfSpeeds variable
- CrossCountryCalculator
    - assumptions:
        - There would only ever be a need for one instance of CrossCountryCalculator just as there's only a need for one non-programmable calculator on a calculus final.
    - int time(int speed, int distance) and int speed(int time, int distance) would both be static because there would never be a need to use either of them multiple times at once.

- PairOfDice
    - Assumptions
        - This class will be used for lots of thing. It must be *robust.*
        - Some games use more than one pair of dice.
    - valueOfFace1 and valueOfFace2 will be non-static since their values will vary from pair to pair.
    - randomize() will be static since it will function the same way on all pairsOfDice and contains no instance-specific information.
    - getInstance() will be non-static since it will have to reference other non-static things.
- class A
    - assumptions
        - It's really hard to tell what this class is supposed to do so I'm allowed to be rather creative with it.
        - class A will be used on day 1 of CS431 after reviewing the syllabus and only having a few minutes for a quick demonstration
        - "Since we're short on time today, I'll be putting this class in the same file as the main() method."
        - "We're going to see how to perform simple operations on primitive data types through this f() method."
    - f() will be static because it will be referenced by the main() method which has to be static
    - int b and float c will be static since they will be referenced by f();