Problem Solving Session

- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

Problem Solving Team Members

Record the name of each of your problem solving team members here.

Do not forget to ***add every team member's name***! Your instructor (or course assistant) may or may not use this to determine whether or not you participated in the problem solving session.

Luke Demi
Logan Nickerson

Range Iterator

- You should recall that the `range()` function in Python can be used to create a range of integers. It expects up to 3 parameters:
  - `range(start, stop)` - creates a range from `start` to `stop - 1`.
  - `range(stop)` - creates a range from `0` to `stop-1`.
  - `range(start, stop, k)` - creates a range from `start` to `stop-1` that contains every k^{th} value.
- Python ranges work very well with Python's for each loop. For example, the following loop prints the even integers from 2 to 100.

```
for(i in range(2, 101, 2)):
    print(i)
```

The Java language does not include a range type, so for this assignment you will make one.

Your range will be an ***iterable*** data structure - essentially an ***immutable collection of integers***. When it is finished, you will be able to use it to write for each loops in Java that work just like they do in Python.

```
1  for(int i : new IntRange(2, 101, 2)) {
2      System.out.println(i);
3  }
```

Problem Solving 1

A range is defined by three values: `start`, `stop`, and `step`. Work with your team to answer the questions to the right.

Given `start=2`, `stop=35`, and `step=3` what is the size of the range (i.e. how many integers does it contain)?

2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32

Size 11

Given `start=2`, `stop=35`, and `step=3` what is the value at index 7 in the range?

2, 5, 8, 11, 14, 17, 20, 23

23

Given any `start<=stop` and `step>=1` derive a general formula for computing the ***value*** at any index `i`.

value = start + step * i;

Given any `start<=stop` and `step>=1` derive a general formula for computing the ***size*** of the range.

size = (stop - start) / step;

Will your formula work if the size of the range is/is not evenly divisible by the `step`?

No

```
public interface Range extends Iterable<Integer> {
      public abstract int getSize();

      public abstract int get(int index);

      public default Iterator<Integer> iterator() {
            throw new UnsupportedOperationException();
      }
}
```

Problem Solving 2

In the space to the right, define the Java interface to represent a ***range*** abstract data type. It must meet the following requirements:

- It must be ***iterable***.
- The must be a way to get the ***size*** of the range, e.g. the number of integers in the range.
- There must be a way to ***get*** the integer at a specific ***index*** in the range.

Problem Solving 3

In order to work with a Java for each loop, the `iterator()` method in your range implementations will need to return a functioning `Iterator`.

Use the space to the right to write an `Iterator` that will work with ***any*** range.

- Don't forget the ***full*** class definition.
- How will you know which value to return when `next()` is called?
- How will you know when `hasNext()` should return `false`?

```
public class RangeIterator implements Iterator<Integer> {

private Range range;
private int index;

public RangeIterator(Range range) {
      this.index = 0;
      this.range = range;
}

@Override
public boolean hasNext() {
      return index < range.getSize();
}

@Override
public int next() {
      int value = range.get(index);
      index++;
      return value;
}

}
```

```java
public static int fibonacci(int n) {
      if(n == 0) {
            return 0;
      }
      int firstNumber = 1;
      int secondNumber = 0;
      int temporary;
      while(n > 1) {
            temporary = firstNumber;
            firstNumber = secondNumber +
            firstNumber;
            secondNumber = temporary;
            n = n - 1;
      }
      return firstNumber;
}
```


Problem Solving 4

Surprise! Fibonacci is back!

Note a ***recursive*** implementation of the Fibonacci function in Python below. Work with your team to translate it into an ***iterative*** Java method.

```python
def fibonacci(n, fn_1=1, fn_2=0):
    if n == 1:
        return fn_1
    elif n == 0:
        return fn_2
    else:
        temp = fn_1
        fn_1 = fn_1 + fn_2
        fn_2 = temp
        return fibonacci(n-1, fn_1, fn_2)
```