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.

Kylie Higgins
Rodney Smith

Problem Solving 1

A Python range is a data structure that represents a range of integer values. Ranges are created with ***start***, ***stop***, and ***step*** values. Refresh your memory by filling in the table to the right.

The values in the range column are meant to be ***inclusive***, e.g. both 0 and 10 should be included in the range of values in the first row.

In the event where one or more values are optional, write OPT instead of an integer value.

Make sure that your instructor or a CA checks you off before moving to the next problem.

Rodney, Kylie

Range	**start**	**stop**	**step**
0 to 10	Opt	10	Opt
50 to 100	50	101	Opt
Even integers from 12 to 34	12	35	2
Odd integers from 33 to 99	33	100	2
Multiples of 5 between 50 and 500	50	501	5
10 to 0	10	-1	-1

```
public static void printRange(int start, int stop,
                              int step) {
      for i in range(start, stop, step):
       print(i)
}

public static int size(int start, int stop, int step) {
    int counter = 0
      while(counter <= stop){
            start += step;
            Counter ++;}
      return counter}
public static int getValue(int start, int stop, int step,
                            int index) {

      If (index < 0){
            Throw new IndexOutofBoundsException
      }
      int value = start + index * step;
      return value;

}
```

Problem Solving 2

A range does not store all of the integer values in the range. That would be a very wasteful use of memory, especially for extremely large ranges, e.g. 0 to 10,000,000,000.

Instead, the start, stop, and step values can be used to compute the value at a specific index in the range.

Working together with your team, implement the methods shown to the left. For this problem, you only need to consider *positive* values for start, stop, and step.

- `printRange` - given start, stop, and step values, prints all of the values in the range. You only need to consider positive values.

- `size` - given the start, stop, and step return the number of values in the range. Be sure to consider edge cases. *Hint*: use the start, stop, and step values from the previous problem to test your algorithm.

- `getValue` - given start, stop, and step values and an index returns the value at the specified index in the range. If the index is out of range, throw an `IndexOutOfBoundsException` . Again, only consider positive values.

Problem Solving 3

Draw the UML class diagram for an abstract data type (ADT) representing a range in Java. It should include at least the following behavior:

- The new ADT should be iterable.
- A method to get the size of the range.
- A method to get the value at a specific index in the range.

When you have finished your diagram, write the code for the Java type (e.g. *class*, *abstract class*, or *interface*).

```
public interface Range extends Iterable<Integer>{

      Int size();
      Int getValue(int index);
```

```
<<interface>> Iterable<E>
-------------------------------


+iterator(): Iterator<E>
```

```
<<interface>> Range
-------------------------------


+iterator(): Iterator<Integer>
+size(): int
+get(index: int)int ;
```

```
Public class IntRange implants Range{
       Private final int start;
       Private final int stop;
       Private final int step;

       Public IntRange(int stop){
              this(0, stop, 1);
       }
       Public IntRange(int start, int stop){
              this(start, stop, 1);
       }

       Public IntRange(int start, int stop, int step){
              if(step == 0){
              Throw new IllegalArgumentExpection(“step cant
be zero”)
              This.start = start;
              This.stop = stop;
              This.step = step;
              }

       @Override
       Public int size(){
              if((step > 0 && start >= stop || step < 0 && start <=stop)){
              Return 0; }
              Int diff = stop - start;

              Int stepAbs = step;
              If (step < 0){
                     stepAbs = -step;
              }

              If (diff < 0){
                     Diff = -diff
              }
```

Problem Solving 4

An `IntRange` is an implementation of the Range ADT that is created with some combination of start, stop, and step values. Implement the `IntRange` class. How many constructors will you need?

For now, just stub out the `iterator()` method.

If you are working digitally and need more space, duplicate this slide.

```
Int count = diff / stepAbs;
if(diff % stepAbs !=0){
Count = conut +1
}
Return count;
```

Problem Solving 4

An `IntRange` is an implementation of the Range ADT that is created with some combination of start, stop, and step values. Implement the `IntRange` class. How many constructors will you need?

For now, just stub out the `iterator()` method.

If you are working digitally and need more space, duplicate this slide.