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

Problem Solving 1

Problem Statement:

Shinji's is the most exclusive sushi shop in Rochester. It only serves premium quality unique sushi creations every day. To order, customers view the available items behind a glass display case. Each item is labeled with only a number. The customer picks between 3 and 8 unique items and hands their list to the cashier. The cashier then removes the items from the case and carefully places them into the to-go bag, heaviest item to lightest item. The customer then pays, takes the order home, and enjoys.

Read through the problem statement above. Then on the right list the classes you think are needed to implement it. For each class, include which data structures you think you will use within the class. **You are not allowed to use Lists**.

```
public class Sushi; //defines a sushi object

public class Shinji;
     HashMap<Integer, Sushi> choices
     HashSet<Integer> customer choices
     TreeSet<Sushi> to-go bag
```

```
public class Sushi implements Comparable<Sushi> {

     private int weight;
     private String name;

     public Sushi(int weight, String name) {
          this.weight = weight;
          this.name = name;
     }

     public String toString() {
          return name + " weighs " + weight;
     }

     public int compareTo(Sushi other) {
          return weight - other.weight;
     }
}
```

Problem Solving 2

Write the class declaration for the Sushi class. Include the class signature, all attributes, and constructor. You do not need to include accessors.

Think about how you can allow this class to be sorted by weight. Make sure to include everything necessary for that in the code to the left.

(we use a comparator and pass it to the data structure responsible for keeping sushi in order)

Problem Solving 3

When placing an order, the customer must select a unique series of numbers from the display case. There are only 25 numbers in the case from 0 - 24. Write the code necessary to create a unique collection of 5 items randomly selected from the case numbers.

You may not change the display case number in any way and you are not allowed to use a list. Each number must appear only once in your solution construct and there must be 5 of them.

```
import java.utils.Random;
HashMap<Integer, Sushi> display; // gets filled
HashSet<Integer> customerChoices = new
HashSet<>();
TreeSet<Sushi> toGoBag = new TreeSet<>(new
SushiWeightComparator())

while(customerChoices.size() < 5) {
      customerChoices.add(Random.nextInt(25));
}

while(customerChoices.size() > 0) {
      toGoBag.add(
      display.get(customerChoices.remove()));
}
// toGoBag is now a unique collection of five
Sushi that is sorted by weight
```

```
public static void main(String[] args) {
      Collection customerChoices =
takeOrder(numItems);
      Order order = bagOrder(customerChoices);
}
```

Problem Solving 4

Assume you already have methods for `takeOrder(numItems)`, which returns a customer's selection of desired items as a `Collection` and `bagOrder(wantedItems)` , which uses the result of `takeOrder` and returns a completed `Order`. Write a `main` method for the `Shop` based on the problem statement, the methods mentioned above, and the `Sushi` and `Order` classes.