CS 61BL — Small Group Tutoring
Summer 2021 — Section 5 Exceptions, Iterators, Iterables

Worksheet 5

1 Pusheen Exceptions

Below is a class that represents a Pusheen. Pusheen cares about two things: happiness and food. Her happiness is directly proportional to how much she is fed.

```
public class Pusheen {
    public int happiness;

    public Pusheen() {
        happiness = 0;
    }

    public void feed(int amount) {
        happiness = 14 * amount;
    }
}
```

Unfortunately, some Pusheen haters have decided to try and feed Pusheen a negative amount! Obviously, we must prevent this from happening.

a). Modify the `feed` method to throw an `InvalidPusheenException` if Pusheen is fed with a negative amount. Being fed a negative amount should **NOT** change Pusheen's happiness.

```
public void feed(int amount) {
    if (amount < 0) {
        throw new InvalidPusheenException();
    }
    happiness = 14 * amount;
}
```

Now suppose we change the method signature of `feed` to:

```
public void feed(int amount) throws InvalidPusheenException {
    //Assume we have implemented everything correctly
}
```

b). Does this change the behavior of `feed`? Does this change require us to change the behavior of other functions that call `feed` as a subroutine?

Adding `throws InvalidPusheenException` does not change the behavior of `feed`. Instead, it requires that any function which calls `feed` calls it within a try catch loop. Basically it just tells the calling function ”hey this method could error and it is your job to make sure that error is always caught”.

Note: In this class you should never have to write a method which has `thorws Exception` in the signature! If you add this you might end up with code that compiles locally but breaks the AG! When error handling, always use a try catch loop instead.

2 Exceptions

What does Java display when the main method of Test is run?
abdec

```java
public class Test {
    String str = "a";

    public void A() {
        try {
            str += "b";
            B();
        } catch (Exception e) {
            str += "c";
        }
    }

    public void B() throws Exception {
        try {
            str += "d";
            C();
        } catch(Exception e) {
            throw new Exception();
        } finally {
            str += "e";
        }
        str += "f";
    }

    public void C() throws Exception {
        throw new Exception();
    }

    public void display() {
        System.out.println(str);
    }

    public static void main(String[] args) {
        Test object = new Test();
        object.A();
        object.display();
    }
}
```

3 Pizza Iterator

Artichoke's is overwhelmed by the number of hungry students in line at 12am. To make things more efficient, the owner has asked you to build a custom iterator that will aggregate all orders and print out the number of slices that should made for each kind of pizza.

The static menu array declared inside `MenuIterator` contains the three types of pizza offered that night.

```
static String[] menu = {"Artichoke", "Margherita", "Meatball"};
```

The input array passed into the constructor contains the list of orders.

```
int [] orders = { 0 , 2 , 1 , 0 , 1 , 0 };
```

Each order is represented by an integer that corresponds to the pizza's index in the menu array. For example, 0 represents an order of Artichoke pizza.

Fill in the code for `MenuIterator`, an iterator that takes in an `int[]` array representing orders at the restaurant and iterates over the aggregated results.

Given the input above, calls to `next()` would eventually return "Artichoke 3", "Margherita 2", "Meatball 1". Make sure your iterator adheres to standard iterator rules.

```
public class MenuIterator implements Iterator {
    private static String[] menu = {"Artichoke", "Margherita", "Meatball"};
    private int[] order_counts = new int[3];
    private int index;

    public MenuIterator(Integer[] orders){
        // Initialize index and order_counts.
        for (int i=0; i < orders.length; i++){
            order_counts[orders[i]] += 1;
        }
        index = 0;
    }

    public boolean hasNext() {
        return index < menu.length;
    }

    public String next() {
        // Should return a string in the format "Artichoke 3".
        String order = menu[index] + "␣" + order_counts[index];
        index += 1;
        return order;
    }
}
```