CSM 61B Fall 2017

# Exceptions & More Interfaces

Mentoring 5: September 25, 2017

## 1 Catch Blocks, Exceptions, and Try-s, Oh My

1.1
```
try {
    doSomething();
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("caught array index exception");
} catch (Exception e) {
    System.out.println("caught an exception");
    throw e;
} catch (NullPointerException e) {
    System.out.println("caught null pointer exception");
} finally {
    System.out.println("in finally block");
}
```

(a) What will print if `doSomething()` throws a `NullPointerException`?

(b) What if `doSomething()` throws an `ArrayIndexOutOfBoundsException`?

(c) What if `doSomething()` doesn't error?

# 2 Mercantilism

2.1 Let's model a feudal society where managers, merchants, and workers cooperate to deliver goods. What happens when we call **new** `Manager().work()`?

```
class Manager {
    Merchant merchant = new Merchant();
    Worker worker = new Worker();
    String[] goods = new String[1];
    void work() {
        try {
            merchant.trade(goods);
        } catch (RuntimeException e) {
            worker.produce("apple pie", goods);
            merchant.trade(goods);
            worker.produce("cornbread", goods);
        } catch (Exception e) {
            merchant.trade(goods);
        } finally {
            System.out.println("All in a day's work");
        }
    }
}
class Merchant {
    void trade(String[] goods) {
        try {
            for (String good : goods) {
                if (good != null) {
                    System.out.println("Traded 1 " + good);
                    return;
                }
            }
            throw new RuntimeException("Not enough goods");
        } catch (Exception e) {
            System.out.println("Oops");
            throw e;
        } finally {
            System.out.println("I love trading");
        }
    }
}
class Worker {
    void produce(String item, String[] goods) {
        int i = 0;
        while (goods[i] != null) {
            i += 1;
        }
        goods[i] = item;
        System.out.println("Done making 1 " + item);
    }
}
```

# 3 VoteIterator

3.1 Define `VoteIterator`, an `IntIterator` that takes in an **`int`**`[] array` of vote counts and iterates over the votes. The input array contains the number of votes each candidate received.

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

Each candidate, represented by their index, `i`, should be returned from each call to `next()` `array[i]` times in total. Given the input above, calls to `next()` would eventually return `1` *twice*, `2` *once*, and `4` *once*.