CSM 61B Spring 2017

Exceptions & Iterators

Week 7: February 27, 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);
} finally {
    System.out.println("in finally block");
}
```

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

```
caught an exception
in finally block
NullPointerException
```

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

```
caught array index exception
in finally block
```

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

```
in finally block
```

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);
    }
}
```

```
Oops
I love trading
Done making 1 apple pie
Traded 1 apple pie
I love trading
All in a day's work
ArrayIndexOutOfBoundsException
```

3 VoteIterator

3.1 Define `VoteIterator`, an iterator 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 };
```

Given the input above, calls to `next()` would eventually return `1` *twice*, `2` *once*, and `4` *once*. Make sure your iterator adheres to standard iterator rules.

```
public class VoteIterator implements Iterator<Integer> {
    private int[] votes;
    private int index, current;
    public VoteIterator(int[] votes) {
        this.votes = votes;
        this.index = this.current = 0;
    }
    public boolean hasNext() {
        if (current < votes[index]) {
            return true;
        }
        for (int i = index + 1; i < votes.length; i++) {
            if (votes[i] > 0) {
                return true;
            }
        }
        return false;
    }
    public Integer next() {
        while (current == votes[index]) {
            index += 1;
            current = 0;
        }
        current += 1;
        return index;
    }
    public void remove() {
        votes[index] -= 1;
        current -= 1;
    }
}
```