Objects! Finally!

6.S092 Lecture 3
Reminders

Assignment 1 is due TONIGHT (at 11:59 pm)!
OH tonight from 7-10 in 32-044
Look at the bug report on Piazza!
Last time...

● ALL the control flow
● The do...while block and the switch statement
● Functions and function signatures
● Recursion (because I wanted to, that’s why.)
Feedback

“I liked how you incorporated the feedback into lecture 2. I could tell that you were really trying to speak slower, and it's just nice that you guys took everything to heart. With that said I liked lecture 2, but maybe next time whenever you have examples of code, if you could stay on that slide for just a few seconds longer that would be great. I know that if we wanted to have the exact code we could look at lecture notes later, but sometimes I found myself trying to type down a few lines of the essential parts of the code just to have it in my notes for reference but when I looked up to get a new part of it, you had already moved on in your powerpoint.”

Fair enough.
Feedback, cont.

“Can we do a quick review of the control flow of a recursion method?”

*Uh, sure? (Not entirely sure what you mean here?)*
public static String reverseString(String s){
    if(s.length() == 0)
        return "";
    String l = s.substring(0, 1);
    return reverseString(s.substring(1)) + l;
}

reverseString("test") → “tset”
    return reverseString("est") + “t”
    return reverseString("st") + “e”
    return reverseString("t") + “s”
    return reverseString(“”) + “t”
    return “”
int roll = diceRoll();
System.out.println(roll);
switch(roll){
    case 1:
    case 2:
    case 3: System.out.println("You rolled a 3 or smaller");
            break;
    case 4:
    case 5:
    case 6: System.out.println("You rolled a 4 or greater");
            break;
int roll = diceRoll();
System.out.println(roll);
switch(roll){
case 1: 
case 2: System.out.println("You also rolled a 2");
case 3: System.out.println("You rolled a 3 or smaller");
break;
case 4: System.out.println("You also rolled a 4");
case 5:
case 6: System.out.println("You rolled a 4 or greater");
break;
FAQ

Main? No.
Unit test? No.
Scanner? No.
Functions? Yes!
And now, objects.
class Queue:
    def __init__(self):
        self.queue = []

    def push(self, elem):
        self.queue.append(elem)

    def pop(self):
        top = self.queue[0]
        self.queue = self.queue[1:]
        return top

    def isEmpty(self):
        return len(self.queue) == 0

The object-oriented concepts are the same, Java has some different buzzwords for them!
Real quick: a class in Python

class Queue:
    def __init__(self):
        self.queue = []

    def push(self, elem):
        self.queue.append(elem)

    def pop(self):
        top = self.queue[0]
        self.queue = self.queue[1:]
        return top

    def isEmpty(self):
        return len(self.queue) == 0
class Queue:
    def __init__(self):
        self.queue = []

    def push(self, elem):
        self.queue.append(elem)

    def pop(self):
        top = self.queue[0]
        self.queue = self.queue[1:]
        return top

    def isEmpty(self):
        return len(self.queue) == 0
Real quick: a class in Python

class Queue:
    def __init__(self):
        self.queue = []

    def push(self, elem):
        self.queue.append(elem)

    def pop(self):
        top = self.queue[0]
        self.queue = self.queue[1:]
        return top

    def isEmpty(self):
        return len(self.queue) == 0
Syntax of a class

class ClassName {
    private Type instanceProp;

    public ClassName() {
        this.instanceProp = someValue;
    }
}
Syntax of a class

class ClassName {
    private Type instanceProp;

    public ClassName() {
        this.instanceProp = someValue;
    }
}
Overloading the constructor

But first, an example with functions. In Java, this is legal.

```java
public static int function(int num){
    return num;
}
```

```java
public static int function(double num){
    return (int) num;
}
```
Overloading the constructor

But first, an example with functions.
In Java, this is legal.

```java
public static int function(int num){
    return num;
}
```

```java
public static double function(double num){
    return num;
}
```

Thank you, type-safe languages!
Overloading the constructor

```java
public class Thing {
    private int val;

    public Thing(){
        this.val = 5;
    }

    public Thing(int val){
        this.val = val;
    }
}
```
toString()

```java
Thing t = new Thing(4);
System.out.println(t);
// prints Lecture3Code$Thing@22998b08

...yeah that’s super helpful.
```
**toString()**

```java
public String toString(){
    // code that returns some String here
}
```

```java
public String toString(){
    return String.valueOf(this.val);
}
```
equals()

Thing t1 = new Thing(4);
Thing t2 = new Thing(4);

System.out.println(t1 == t2); // this prints false...?

Well, sadface for all involved.
.equals()

== for primitives does value equality. For objects, however, it’s going to check if these objects share the same memory location.

Objects are checked through .equals().
.equals()

Thing t1 = new Thing(4);
Thing t2 = new Thing(4);

System.out.println(t1.equals(t2));
// this STILL prints false!

#annoying.
.equals()

From Oracle:

“The equals() method compares two objects for equality and returns true if they are equal. The equals() method provided in the Object class uses the identity operator (==) to determine whether two objects are equal.”

Java does == by default! #evenmoreannoying
.equals()

So we overwrite it!

// MUST TAKE IN AN OBJECT
public boolean equals(Object obj) {
    // code to return some boolean here
}

...but that object though...
Conventions for an `.equals()` method

1. Test if it’s `null`
2. Make sure the object is the same type of `object as this`
3. Cast it!
4. And now, test your instance parameters.

Let’s write it for `Thing`!
public boolean equals(Object o) {
    if (o == null) //1
        return false;
    if (! (o instanceof Thing)) //2
        return false;
    Thing t = (Thing) o; //3
    return this.val == t.val; //4
}
Inheritance

Because that’s, like, a thing, yo.
This is probably best by example.

```java
public class Bicycle {
    public int cadence;
    public int gear;
    public int speed;

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }
    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
    public void speedUp(int increment) {
        speed += increment;
    }
}
```
public class MountainBike extends Bicycle {
    public int seatHeight;

    public MountainBike(int startHeight, int startCadence,
                          int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
Case study: The **ArrayList**

This is Java’s dynamically sized array object.

```java
ArrayList<E> list = new ArrayList<E>();
ArrayList<E> list = new ArrayList<E>(7);
```

...what is this `<E>` business?
Case study: The **ArrayList**

public boolean add(E element);
public void add(int index, E element);
public E get(int index);
public int indexOf(Object o);
public E remove(int index);
public boolean remove(Object o);
public int size();
Next time: abstract classes & interfaces

Let’s code!