



Recitation 5 - February 6th

A4/GR

William and Isaac

Object Oriented Programming



Objects

They put the "O" in OOP.

What are they?

What do they represent?

What do they contain?

Instance - a single object



Classes

What is a Class?

- High Level: Classes are a blueprints for objects!
- Essentially a Class describes what an object can do and what kind of components it can have.

Creating a Class



me chillin' with a snow
monkey

In case you forgot:

```
public class Monkey {  
  
  
}
```

- This creates a class named Monkey that is public



Why?

The reasons why OOP is so widely used

- Modularity
- Information-hiding
- Code re-use
- Pluggability/Debug-ability

What do we do in real life? Do you know what's inside your car? Computer? What do we do to fix them?


Visibility Modifiers



Visibility Modifiers

Controlling who sees what

- `public`
 - Any other class
- `package private` (no modifier)
 - Any class in the package
- `private`
 - Only in the class
- `protected`
 - Very similar to private, we'll get there



So what do visibility modifiers mean?

So which one of these calls in MonkeyManager work? C/R?

What if Monkey Manager isn't in the same package?

What if it was?

```
public class Monkey {  
    private int age;  
    public SmartLevel intellect;  
    Softness softness;  
}  
  
public class MonkeyManager {  
    Monkey isaac = new Monkey();  
    isaac.age = 21(?);  
    isaac.intellect = SmartLevel.HIGH;  
    isaac.softness = Softness.ULTRASOFT;  
}  
  
public enum SmartLevel { LOW, MEDIUM, HIGH }  
  
public enum Softness {NOT SOFT, SOFT, VERYSOFT  
ULTRASOFT}
```

P.S. Isaac is so going to kill me for this

P.P.S. Isaac is ultra soft

Instance Data



Instance Data

Storing what defines a particular object

- Most objects have some information that defines them (properties of the object)
- We call these instance data
- Should really be defined first thing in the class
- You should always define as `private/protected` unless absolutely necessary



Instance Data

Storing what defines a particular object

```
public class Student {  
  
    private String name;  
  
    private int gtID;  
  
  
    public void doStuff(){}  
  
}
```

Constructors



Constructors!


- a special method of a class or structure that **initializes** an object of that type
- A constructor is an instance method that has the same name as the class
- Should be used to **initialize** data
- By default, a class comes with a constructor that has no parameters and does nothing.

Creating a constructor



```
public class Monkey {  
    private int age;  
    private SmartLevel intellect;  
    private Softness softness;  
  
    public Monkey(int age) {  
        this.age = age;  
    }  
  
}  
  
public enum SmartLevel { LOW, MEDIUM, HIGH }  
public enum Softness {NOT SOFT, SOFT, VERYSOFT ULTRASOFT}
```

this.age refers to the instance variable age, we are thus setting the instance variable age to the parameter also named age. This makes your code explicit, so the compiler knows which variable is which





- As you can see a constructor is a great place to initialize instance variables (instance data), since it is guaranteed to be called when the object is created.

Creating a constructor (and Constructor Chaining)

```
public class Monkey {
    private int age;
    private SmartLevel intellect;
    private Softness softness;
    public Monkey(int age) {
        this.age = age;
    }
    public Monkey(SmartLevel smartness, int age) {
        this.intellect = smartness;
        this(age);
    }
    public Monkey(SmartLevel smartness, int age, Softness softness) {
        this.softness = softness;
        this(smartness, age);
    }
}

public enum SmartLevel { LOW, MEDIUM, HIGH }
public enum Softness {NOT SOFT, SOFT, VERYSOFT ULTRASOFT}
```

 **this(age)** is calling the previous constructor (Monkey(int age)). This is java's notation for "chaining constructors". Sorta equivalent to someone calling the constructor Monkey(age).

 Same idea for constructor chaining here, except it is now calling Monkey(SmartLevel smartness, int age)



Methods



Methods

Some Basics

- Represent things that a class can do! Ex. a class's behavior.
- Equivalent to functions/procedures
- Public methods are known as the **“client interface”**
 - Clients don't need to know the details of a method, just what the method does! This is the basis of abstraction

Methods



```
public class Monkey {  
    private int age;  
    private SmartLevel intellect;  
    private Softness softness;  
  
    public Monkey(int age) {  
        this.age = age;  
    }  
    public int getAge() {  
        return this.age;  
    }  
    public SmartLevel intellect() {  
        Return this.intellect;  
    }  
    public int calculateIntelligenceByAge() {  
        return age/(intellect.ordinal() + 1);  
    }  
}
```

Encapsulation



Encapsulation

- Keeping things private and controlling values through getters/setters
- Particularly, setting restrictions in the setter methods allows for greater flexibility

```
public void setGrade(int grade) {  
  
    if (grade < 0) this.grade = 0;  
  
    else this.grade = grade;  
  
}
```