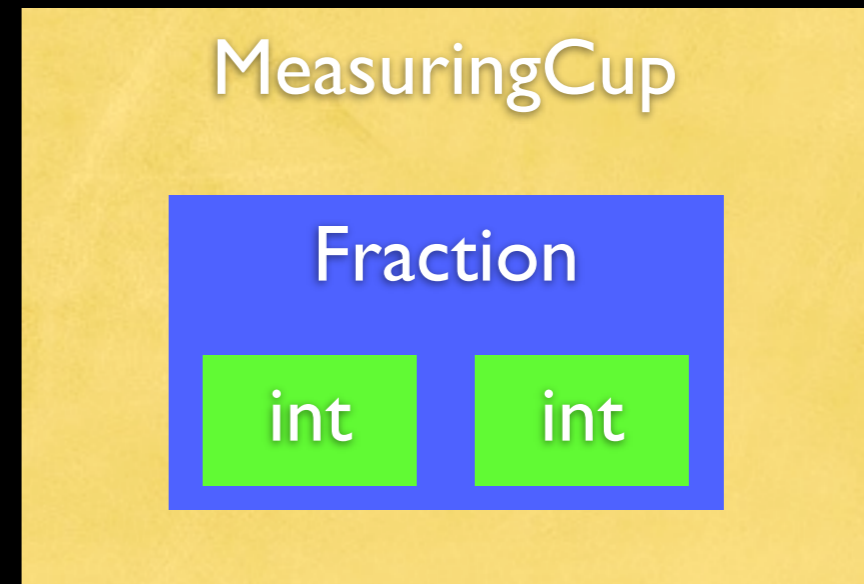


# Inheritance

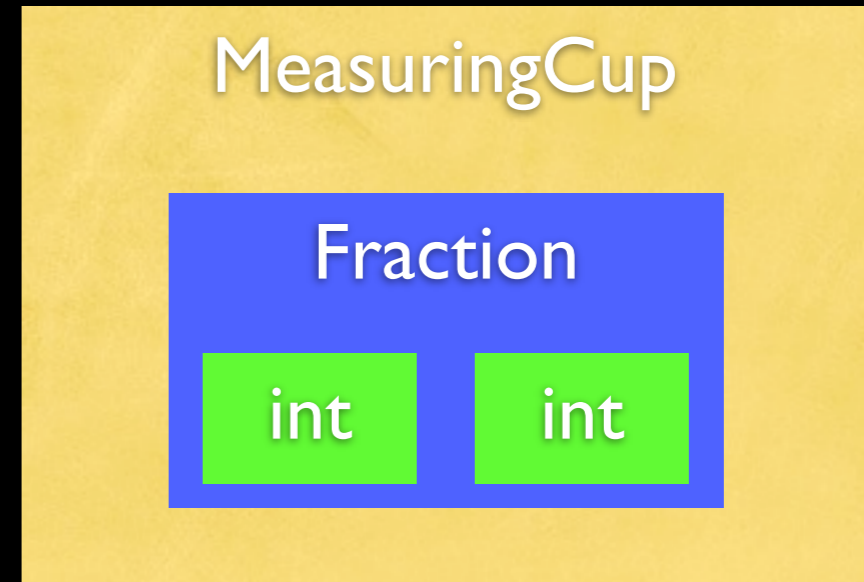
Not what you get from your parents

# Composition Recap



# Composition Recap

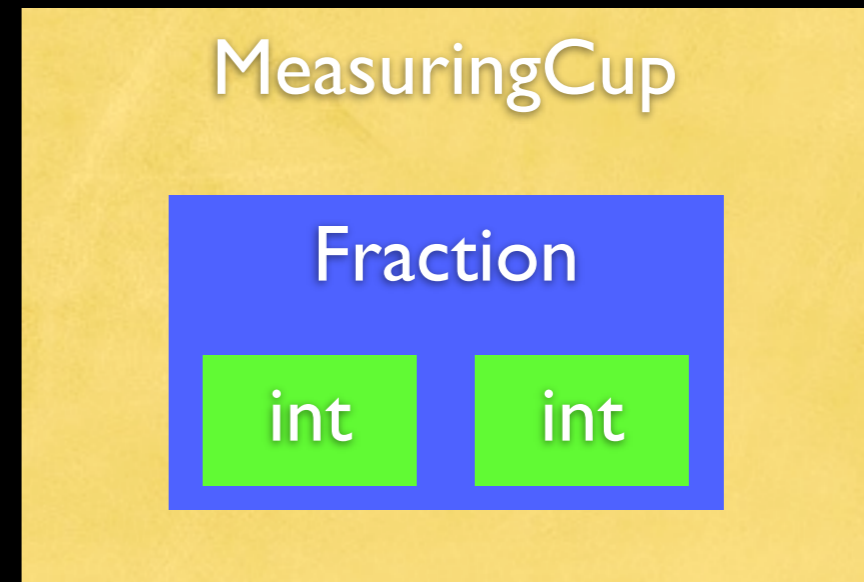
A Fraction has two integer's



# Composition Recap

A Fraction has two integer's

A MesuringCup has a  
Fraction(al) amount of liquid

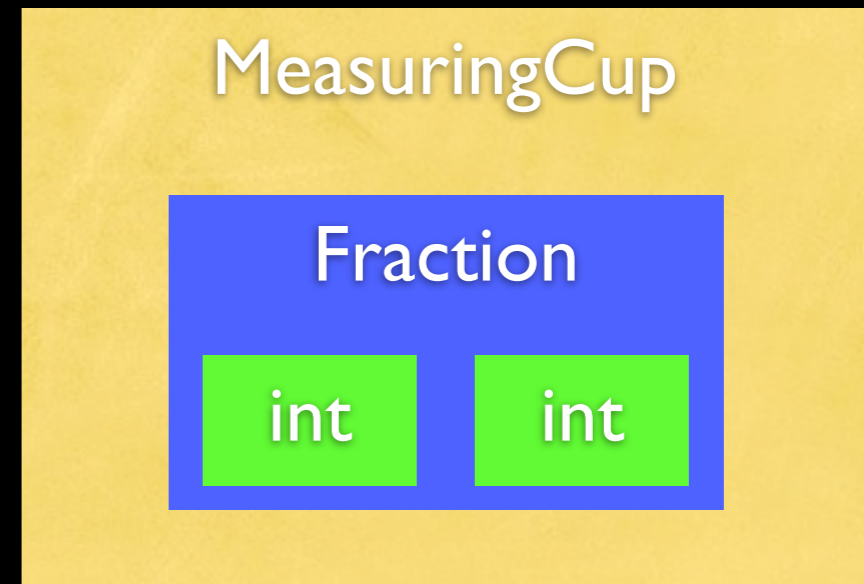


# Composition Recap

A Fraction has two integer's

A MesuringCup has a Fraction(al) amount of liquid

```
public class MeasuringCup
{
    private Fraction amount;
}
```

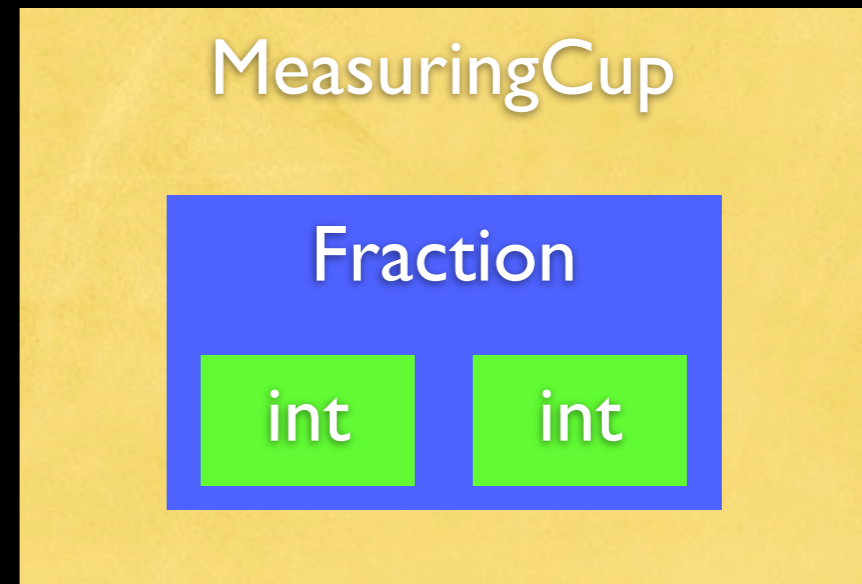


# Composition Recap

A Fraction has two integer's

A MesuringCup has a  
Fraction(al) amount of liquid

```
public class MeasuringCup  
{  
    private Fraction amount;  
}
```

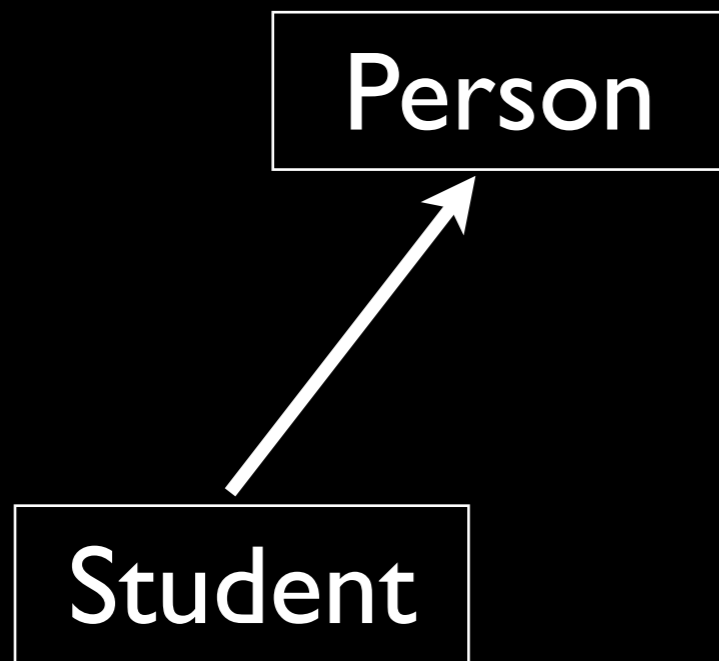


← null by default!

# Inheritance

- New type of relationship
- One class is called the **parent** class, the other the **child** class
- When a child class **extends** a parent class, the child **inherits** all of the public features of the parent
- We say that the child class "is-a" version of the parent class (but not vice-versa)

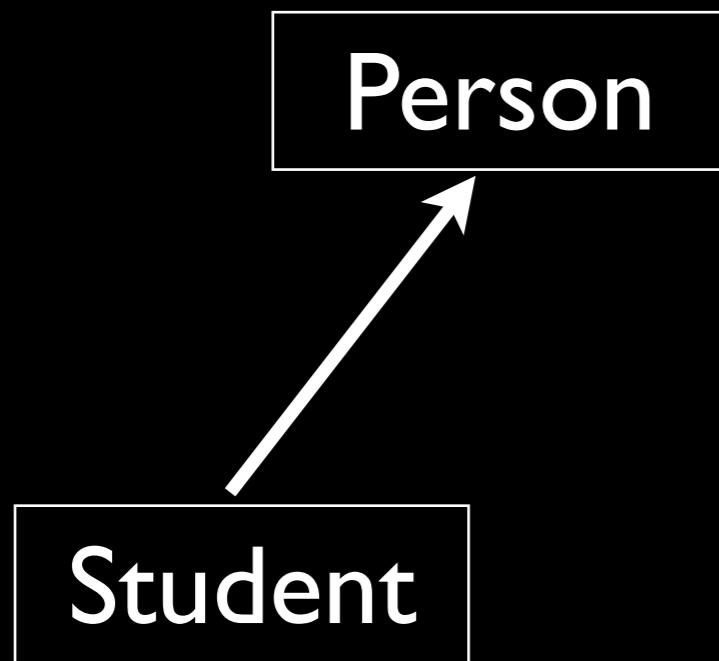
# Example



```
public class Person {
    private String name;

    public String getName() {
        return name;
    }
}
```

# Example



```
public class Person {
    private String name;

    public String getName() {
        return name;
    }
}
```

If Student extends Person, then the Student class will inherit the method **getName()** but not the variable name (directly)... why?

# Simple Example

```
Student s = ...;
```

```
String result = s.getName();
```

# Simple Example

```
Student s = ...;
```

```
String result = s.getName();
```

Student must have a `getName()` method (which it gets from `Person` class) **IF** `Student` is a subclass of `Parent`.

# How to extend

```
public class Student extends Person {  
  
}
```

# How to extend

```
public class Student extends Person {  
  
}
```

That's it!

# How to extend

```
public class Student extends Person {  
  
}
```

That's it!

- We also say that Student is a **subclass** of Person (Person is the **superclass**)

# How to extend

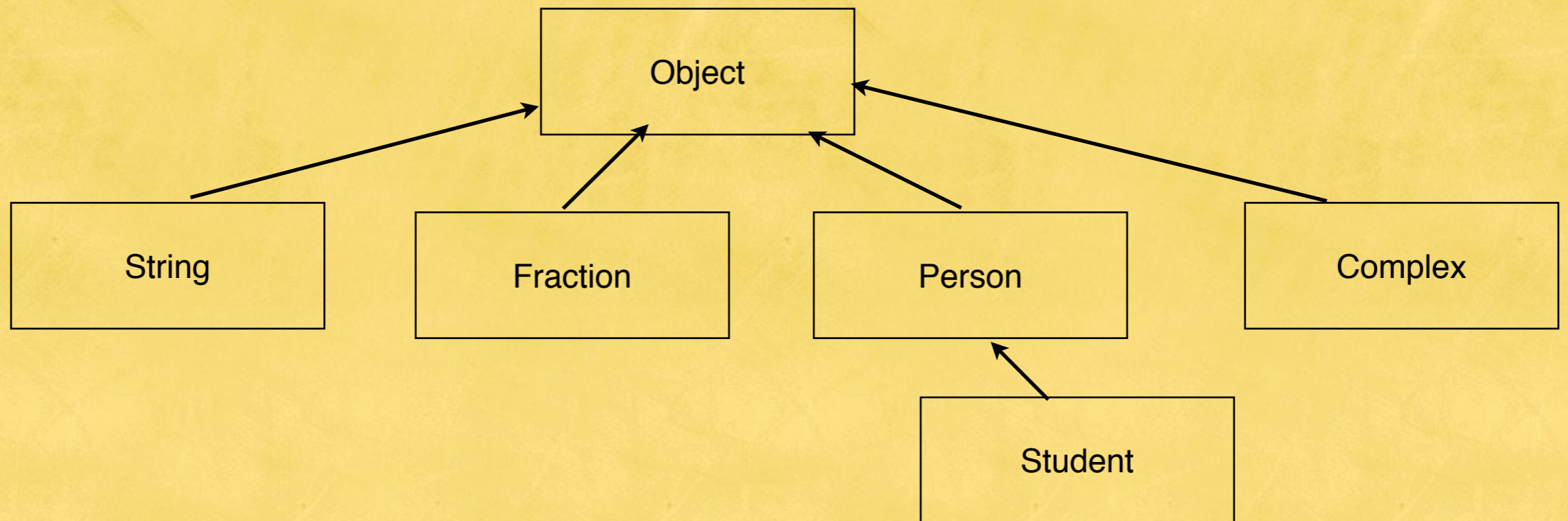
```
public class Student extends Person {  
  
}
```

That's it!

- We also say that Student is a **subclass** of Person (Person is the **superclass**)
- Person can be called the **base** class, and Student is the **derived** class

# Object Class

- Object is the base class for **ALL** classes in Java
- Every class is derived from Object



# Free Stuff!

- If every class is derived from Object, then every class inherits free stuff from Object!

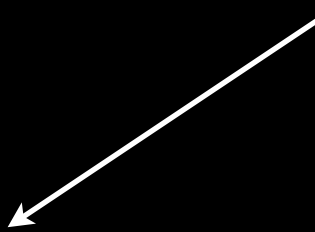
```
public class Object {  
    public String toString()...  
    public boolean equals(Object o) ...  
}
```

# Free Stuff!

- If every class is derived from Object, then every class inherits free stuff from Object!

```
public class Object {  
    public String toString()...  
    public boolean equals(Object o) ...  
}
```

Returns a virtual  
memory address



# Free Stuff!

- If every class is derived from Object, then every class inherits free stuff from Object!

```
public class Object {  
    public String toString()...  
    public boolean equals(Object o) ...  
}
```

Returns a virtual  
memory address

Checks if two  
objects have the  
same virtual  
memory address

# Code that works

```
public static void main(String[] args) {  
    Fraction f1 = new Fraction(3, 4);  
    Fraction f2 = new Fraction(6, 8);  
    System.out.println(f1.toString()); //@Fraction2A...  
    System.out.println(f1.equals(f2)); //false  
}
```

# Code that works

```
public static void main(String[] args) {  
    Fraction f1 = new Fraction(3, 4);  
    Fraction f2 = new Fraction(6, 8);  
    System.out.println(f1.toString()); //@Fraction2A...  
    System.out.println(f1.equals(f2)); //false  
}
```

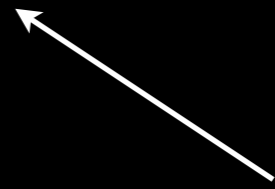
Bad results... we need to **override** these methods to make them more useful

# Overriding

- Overriding means a subclass inherits a superclass method and then changes how it behaves

```
public class Fraction extends Object
{
    public String toString() {
        return numer + "/" + denomer;
    }
}
```

optional



# Overriding

- Overriding means a subclass inherits a superclass method and then changes how it behaves

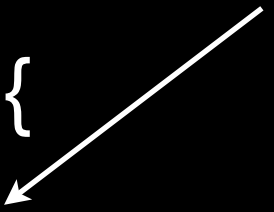
```
public class Fraction extends Object
{
    public boolean equals(Object o) {
        //trickier to do...
    }
}
```

# Overriding


```
public class Fraction extends Object
{
    public boolean equals(Object o) {
        Fraction other = (Fraction)o;

        return numer == other.numer && denomer == other.denomer;
    }
}
```

Cast the Object to  
a Fraction type



↑  
Compare the parts  
of the Fractions



# That seems weird...

- The following would NOT be overriding

```
public class Fraction extends Object
{
    public boolean equals(Fraction other)
    {
        return numer == other.numer && denomer == other.denomer;
    }
}
```

Why is it not?

What is it really?

# That seems weird...

- The following would NOT be overriding

```
public class Fraction extends Object
{
    public boolean equals(Fraction other)
    {
        return numer == other.numer && denomer == other.denomer;
    }
}
```

Why is it not?  
What is it really?

Different parameter  
list means this is  
overloading!

# Summary

- A class should **extend** another if there exists an "is-a" relationship
- A subclass inherits all of the properties of the superclass
- A subclass can change inherited methods by overriding them

# Practice

- Make your Complex be an **explicit** subclass of Object
- Override the toString() and equals(Object) methods so that they make sense [very similar to what I did for Fraction on the previous two slides]
- Test your overridden methods in your Driver
- Example test code on next page:

# Testing

```
Complex a = new Complex(2, 3);  
Complex b = new Complex(2, 3);  
Complex c = new Complex(2, 4);
```

```
System.out.println(a.toString()); //2 + 3i  
System.out.println(a.equals(b)); // true  
System.out.println(a.equals(c)); // false
```