Week 3 - Wednesday

COMP 2000

Last time

- What did we talk about last time?
- Inheritance

Questions?

Project 1

Inheritance Examples

The Person class

- We can imagine a hierarchy of inheritance starting with a **`Person`** with the following members:
  - Name (final)
  - Age
- **`Student`** extends **`Person`** and adds:
  - Major
  - GPA
- **`Politician`** extends **`Person`** and adds:
  - Political party
- **`OtterbeinStudent`** extends **`Student`** and adds:
  - ID number (final)
- Members should have getters and setters as appropriate
- All classes should override the **`toString()`** and **`equals()`** methods

Overriding Methods

Adding to existing classes is nice…

- Sometimes you want to do more than add
- You want to change a method to do something different
- You can write a method in a child class that has the same name as a method in a parent class
- The child version of the method will always get called
- This is called **overriding** a method

Mammal example

- We can define the **Mammal** class as follows:

```java
public class Mammal {
  public void makeNoise() {
     System.out.println("Grunt!");
  }
}
```

`Mammal` subclasses

- From there, we can define the **`Dog`**, **`Cat`**, and **`Human`** subclasses, overriding the **`makeNoise()`** method appropriately

```
public class Dog extends Mammal {
  public void makeNoise() { System.out.println("Woof"); }
}
```

```
public class Cat extends Mammal {
  public void makeNoise() { System.out.println("Meow"); }
}
```

```
public class Human extends Mammal {
  public void makeNoise() { System.out.println("Hello"); }
}
```

Dynamic binding

- All normal Java methods use dynamic binding
- This means that the most up-to-date version of a method is always called
  - It also means that the method called by a reference is often not known until run-time
- Consider a class **`Wombat`** which extends **`Marsupial`** which extends **`Object`**
- Let's say that **`Wombat`**, **`Marsupial`**, and **`Object`** all implement the **`toString()`** method

Marsupial class

- Here's a simple **Marsupial** class:

```java
public class Marsupial {
   private final boolean pouch;

   public Marsupial(boolean pouch) {
      this.pouch = pouch;
   }

   public boolean hasPouch() {
      return pouch;
   }

   public String toString() {
      return "Marsupial " + (pouch ? "with" : "without") + " a pouch";
   }
}
```

Wombat class

- And the **`Wombat`** class extends the **`Marsupial`** class:

```
public class Wombat extends Marsupial {
   private final String name;

   public Wombat(String name) {
       super(true); // Wombats have pouches
       this.name = name;
   }

   public String getName() {
       return name;
   }

   public String toString() {
       return name + " the Wombat";
   }
}
```

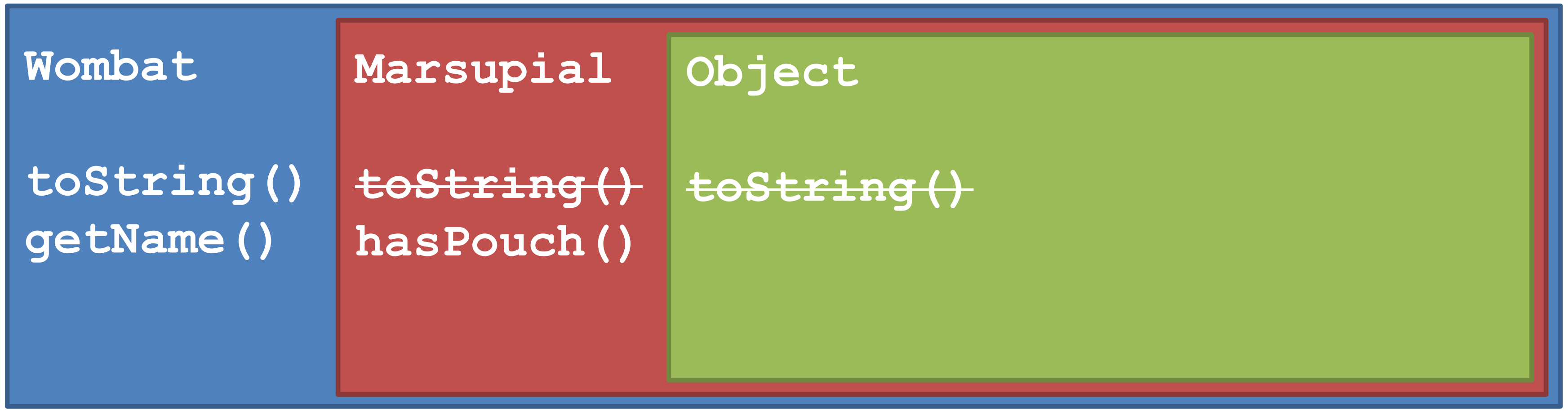
Wombat example

- What happens when we call **`toString()`** on an **`Object`**, a **`Marsupial`**, and a **`Wombat`**, all stored in **`Object`** references?

```
Object object = new Object();
Object marsupial = new Marsupial(false);
Object wombat = new Wombat("Winifred");
// Prints "java.lang.Object@7852e922"
System.out.println(object.toString());
// Prints "Marsupial without a pouch"
System.out.println(marsupial.toString());
// Prints "Winifred the Wombat"
System.out.println(wombat.toString());
```

How to think about inheritance

- Every object has a copy of its parent object inside (which has its parent inside, and so on)
- All methods from the class and parents are available, but the outermost methods are always chosen
  - If a class overrides its parent's method, you always get the overridden method



Using super to call parent methods

- In addition to using **`super`** to call parent constructors, you can use **`super`** to call parent methods
- You can only call methods "one level up", not methods that were overridden by parents

```
public class Wombat extends Marsupial {
   private final String name;

   public Wombat(String name) {
       super(true); // Wombats have pouches
       this.name = name;
   }

   public String getName() {
       return name;
   }

   // Prints "Name the Wombat (Marsupial with a pouch)"
   public String toString() {
       return name + " the Wombat (" + super.toString() + ")";
   }
}
```

Quiz

Upcoming

Next time…

- More on the **`final`** keyword
- Abstract classes
- More on the **`instanceof`** keyword and **`getClass()`** method
- UML class diagrams

Reminders

- Keep reading Chapter 17
- Keep working on Project 1
  - Due next Friday