Outline

- Subtyping
  - Polymorphic Variables
  - The instanceof Operator
  - The Object Class

- Polymorphic Methods
  - Overriding Methods
  - The super Reference Revisited (in Methods)
  - Dynamic Dispatch

- Conclusion
The instanceof Operator

- Consider the following conundrum:
  - In the Farm class, write a method `exciteOnlyCows()` that makes the Cows speak

```java
public void exciteOnlyCows() {
    for (int i = 0; i < animals.size(); i++) {
        if (animals.get(i) is a Cow?) {
            animals.get(i).speak();
        }
    }
}
```

- Need a way to determine whether an object belongs to a certain class!
  - Enter: `instanceof` operator
The `instanceof` Operator (Cont.)

- **Java Syntax:** `obj instanceof Classname`
  - Results in a boolean
  - Determines if an object variable, `obj`, is of type `Classname`

---

**Example Usage:**

```java
Animal ani = new Bird();
if (ani instanceof Bird) {
    System.out.println("It's a Bird!");
}
if (ani instanceof Animal) {
    System.out.println("It's an Animal!");
}
if (ani instanceof Student) {
    System.out.println("It's a Student!");
}
if (ani instanceof Object) {
    System.out.println("It's an Object!");
}
```

**Output:**

It's a Bird!
It's an Animal!
It's an Object!
The instanceof Operator (Cont.)

- Solution:

```java
public class Farm {
    public void exciteOnlyCows() {
        for (int i = 0; i < animals.size(); i++) {
            Animal a = animals.get(i);
            if (a instanceof Cow) {
                a.speak();
            }
        }
    }
}
```

```
// (code omitted)
```
And we can also make all Birds lay an egg!

```java
public class Farm {
    public void makeBirdsLayEggs() {
        for (int i = 0; i < animals.size(); i++) {
            Animal a = animals.get(i);
            if (a instanceof Bird) {
                a.layEgg();
            }
        }
    }
    // (code omitted)
}
```

But this doesn't compile!! (Why?)
Problem: Animals generally don't have a `layEgg()` method

- Only *specific* Animals do
Solution: "Down"-Casting

- But *programmers* know inside this if-statement that it must be a Bird!
  - Need to tell Java! (Use a down-cast)

```java
public void makeBirdsLayEggs() {
    for (int i = 0; i < animals.size(); i++) {
        Animal a = animals.get(i);
        if (a instanceof Bird) {
            Bird b = (Bird) a; // Down cast to Bird!
            b.layEgg(); // Now this method exists for b!
        }
    }
}
```
Outline

- Subtyping
  - Polymorphic Variables
  - The instanceof Operator
    - The Object Class
- Polymorphic Methods
  - Overriding Methods
  - The super Reference Revisited (in Methods)
  - Dynamic Dispatch
- Conclusion
Important!! The Object Class in Java

- In truth all classes have a superclass*
  - **Important:** Any class that doesn't explicitly extend from a superclass inherits from the Object superclass.
  - Java inserts the `extends Object` statement when you don't!

```java
public class Student {
    ...
}
```

- Actually the same as writing:

```java
public class Student extends Object {
    ...
}
```

* Except for the Object class
Why Have a "Root" Object Class?

- Lets Java provide a good deal of *consistency* across all objects
  - The following methods pertain to the *Object* class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean equals(Object other)</td>
<td>Returns true if the given object is &quot;equal to&quot; this one</td>
</tr>
<tr>
<td>int hashCode()</td>
<td>Returns a distinct integer representation of this object. Useful for identifying an object within a Collection.</td>
</tr>
<tr>
<td>String toString()</td>
<td>Returns a string representation of this object</td>
</tr>
<tr>
<td>(others omitted)</td>
<td>(others omitted)</td>
</tr>
</tbody>
</table>

Java can now make assumptions, like:

- Every object has an `equals(..)` method
- Every object has a `toString(..)` method
Also, **Objects** are the *ultimate* polymorphic type:

- Example:
  
  - Not very useful though, as you always need to down-cast to call any specific method

```java
Object a = new Student(); // Works
Object b = new Bird();    // Works
...
```

- Another Example: This array list can hold anything

```java
ArrayList<Object> list = new ArrayList<>();
```

- Another Example: This method accepts anything as input

```java
public void method(Object obj) {
...}
```