CSCI 261
Computer Science II
Problem 1: Code duplication

- Re-writing instance variables and methods for each new animal
  - Goal: Just write common code once!
  - Solved through inheritance: the creation of the Animal superclass
Problem 2: Hard to update code

- Examples:
  - Adding a weight for all animals later
  - Updating the way a common method, `display()`, prints an Animal's report

- Goal: Want to make code updates in one place!
  - Also solved through inheritance
Problem 3: Hard to extend code

- Still has an ArrayList\(<\text{Cows}>\) and an ArrayList\(<\text{Birds}>\)
- Still has an add method for each type of animal: \text{addCow()}, \text{addBird()}
- \text{exciteAll()} and \text{summary()} still have to loop through both lists separately
- \text{(How to adding Goats to Farm?)}

We will solve through: polymorphism
What about Farm? (Still Lots of Duplications!)

```java
public class Farm {
    private ArrayList<Cow> cows;
    private ArrayList<Bird> birds;

    public Farm() {
        this.cows = new ArrayList<>();
        this.birds = new ArrayList<>();
    }

    public void addCow(Cow newCow) {
        this.cows.add(newCow);
    }

    public void addBird(Bird newBird) {
        this.birds.add(newBird);
    }

    public void exciteAll() {
        for (int i = 0; i < cows.size(); i++) {
            this.cows.get(i).speak();
        }
        for (int i = 0; i < birds.size(); i++) {
            this.birds.get(i).speak();
        }
    }

    public void summary() {
        System.out.println("-------------");
        System.out.println("Farm Summary");
        System.out.println("-------------");
        int numAnimals = 1;
        for (int i = 0; i < cows.size(); i++) {
            System.out.println("# " + numAnimals);
            this.cows.get(i).display();
            System.out.println();
            numAnimals++;
        }

        //display birds
        numAnimals = 1;
        for (int i = 0; i < birds.size(); i++) {
            System.out.println("# " + numAnimals);
            this.birds.get(i).display();
            System.out.println();
            numAnimals++;
        }
    }
}
```
Outline

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

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

- **Conclusion**
Our current understanding of object variables:

- A Cow variable is a “box” that can store *any* Cow object
- Let's say that the Cow box’s shape looks like:

Example:

```java
Cow c1, c2;
```
Our current understanding of object variables:

- A Cow variable is a “box” that can store any Cow object
- Let's say that the Cow box’s shape looks like:

Example:

```java
Cow c1, c2;
c1 = new Cow();
```
Our current understanding of object variables:

- A Cow variable is a “box” that can store any Cow object
- Let's say that the Cow box’s shape looks like:

Example:

```java
Cow c1, c2;
c1 = new Cow();
c2 = new Cow("Tuffy", "Mewww");
```
Object Variables: Think “Boxes”

- Our current understanding of object variables:
  - A Cow variable is a “box” that can store any Cow object
  - Let's say that the Cow box’s shape looks like:

- Example:

```java
Cow c1, c2;
c1 = new Cow();
c2 = new Cow("Tuffy", "Mewww");
c1 = c2;
```
Our current understanding of object variables:

- A Cow variable is a “box” that can store any Cow object
- Let's say that the Cow box’s shape looks like:

Example:

```java
Cow c1, c2;
    c1 = new Cow();
    c2 = new Cow("Tuffy", "Mewww");
    c1 = c2;
    c2 = null;
```
Subtyping and Polymorphism

- **Subtypes**: an object of a subclass is also an object of its superclass
  - *e.g.*, Cows and Birds are *subtypes* of the Animal class

- **Important**: *Object variables (i.e., “boxes”)* may hold objects…
  - Of their declared type, or
  - Of any of their subtype
Important: *Object variables* may hold objects of their declared type or of any subtype.

- What's the "shape" of the Animal box?
Important: *Object variables* may hold objects of their *declared type or* of any subtype.

What's the "shape" of the Animal box?

1. **Bird Type**
2. **others...**
3. **Animal Type**
(merge all subclass boxes!)
Subtyping and Polymorphism

- Important: *Object variables* may hold objects of their *declared type* or of any subtype.

- What's the "shape" of the Animal box?

  Animal Type
  (merge all subclass boxes!)
Important: *Object variables* may hold objects of their *declared type* or of any subtype.

What's the "shape" of the Animal box?

Animal Type
(merge all subclass boxes!)
Polymorphism literally means “many shapes”

- What's the "shape" of the Animal box? Take all subtypes, merge their boxes together.

Behold! The **polymorphic** (and **generic**) Animal type!
Polymorphic Variables (Important!)

- Java supports subtypes via *Polymorphic Variables (many shapes)*

```java
Bird b1 = new Bird(); // We always knew this works. Store the Bird object in a
// Bird variable
```

(Bird variable stores a Bird object)
Java supports subtypes via *Polymorphic Variables (many shapes)*

```java
Bird b1 = new Bird();   // We always knew this works. Store the Bird object in a
                        // Bird variable
Animal b2 = new Bird(); // THIS WORKS TOO! A Bird object stored in an Animal
                        // variable. This is called substitution.
```

(Animal variable can also store a Bird due to subtyping)
Does Reverse Substitution Work?

- Does the reverse work?
  - Remember that inheritance is a one-way street.

```java
Bird daffy = new Animal(); // The Bird box is more specialized than what
// you're trying to store inside it!
```

- Analogy:
  - We "molded" a box to store Birds.
  - It cannot be re-molded to store something more generic.
Sanity Check: Polymorphic Variables

- Given the class hierarchy, which of the following assignments are valid?

```java
Student s1 = new DoctoralStudent(); ✔️
GradStudent s2 = new PhDStudent(); ✔️
PhDStudent s3 = new GradStudent(); ❌
UgradStudent s4;
s4 = new AssocStudent(); ✔️
s4 = new BachelorStudent(); ✔️
s4 = new DentalStudent(); ❌
DentalStudent s5 = new MedStudent(); ❌
```
Trickier Still: Polymorphic Variables

- Given the class hierarchy, which of the following assignments are valid?

```java
Student s;
GradStudent g1 = new GradStudent();
GradStudent g2;
s = g1;  // valid
s = g2;  // won't compile, but shouldn't it?
g2 = (GradStudent) s;  // need to down-cast
```

![Class Hierarchy Diagram]

- Student
  - GradStudent
    - AssocStudent
    - BachelorStudent
    - MasterStudent
    - DoctoralStudent
      - PhDStudent
      - DentalStudent
      - MedStudent
Given the class hierarchy, which of the following assignments are valid?

GradStudent g;
MasterStudent m;
DoctoralStudent d;
m = new MasterStudent();
g = m; ✓
d = (DoctoralStudent) m; ✗ //won't compile, they don't share //subtype-supertype relationship!
d = (DoctoralStudent) g; ✗ //compiles, but runtime error when trying to call DoctoralStudent methods!
Fixing Problem 3 with Polymorphism!

- Problem 3: Hard to extend code
  - Farm now has a single ArrayList<Animal>
  - Farm has a single method for all types of animals: addAnimal()
  - exciteAll() and summary() are vastly simplified
  - Introduce another animal type now requires no change to Farm class!

- Insight: Animal is a polymorphic type!

- [Update our Farm code now!]