Lecture 20

Inheritance and Polymorphism
Inheritance

- OOP lets you define new classes from existing classes.
- Very helpful for reusing code and we’re always trying to reuse our code!
Inheritance

- We use a class to model stuff of the same type.

- Sometimes, different classes have common properties and behaviors that are general, and can be shared by other classes.

- You can define a specialized class that inherits from the general class

- This is for “is-a” relationships
Inheritance

- The more generic class is the superclass
- the more specific is the subclass
- a subclass is not a *subset* of a class. It usually has more information than the superclass
- private data fields and methods are *only accessible inside the class*. That means they can’t be accessed from a subclass. Subclasses only inherit accessible data fields and methods
Inheritance

- When you create a subclass, you use the keyword `extends` in the class definition and which class you’re extending from.
Inheritance

- Animal example!
public class Animal {

    private String name;
    private String sound;
    private double height;
    private double weight;

    Animal(){}

    Animal(String name, String sound, double height, double weight){
        this.name = name;
        this.sound = sound;
        this.height = height;
        this.weight = weight;
    }

    public void setName(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }

    public void setSound(String sound){
        this.sound = sound;
    }

    public String getSound(){
        return sound;
    }
}

Inheritance
public class Bird extends Animal {
    private boolean flies;
    private double beakLength;

    Bird() {
    }

    Bird(boolean flies, double beakLength) {
        this.flies = flies;
        this.beakLength = beakLength;
    }

    Bird(boolean flies, double beakLength, String name, String sound) {
        this.flies = flies;
        this.beakLength = beakLength;
        //this.name = name;
        //this.sound = sound;
        setName(name);
        setSound(sound);
    }

    public void setFlies(boolean flies) {
        this.flies = flies;
    }

    public boolean doesItFly() {
        return flies;
    }

    public void setBeakLength(double beakLength) {
        this.beakLength = beakLength;
    }

    public double getBeakLength() {
        return beakLength;
    }
}
Inheritance

```java
public class Fish extends Animal{
    private boolean livesInOcean;

    Fish(){
    }

    Fish(boolean livesInOcean){
        this.livesInOcean = livesInOcean;
    }

    Fish(boolean livesInOcean, String name, String sound){
        this.livesInOcean = livesInOcean;
        setName(name);
        setSound(sound);
    }

    public void setLivesInOcean(boolean livesInOcean){
        this.livesInOcean = livesInOcean;
    }

    public boolean getLivesInOcean(){
        return livesInOcean;
    }
}
```
public class TestAnimal {

    public static void main(String[] args) {

        Fish myFish = new Fish(true, "Wanda", "Glub Glub");
        Bird myBird = new Bird(true, 3.0);

        System.out.println("My fish is named " + myFish.getName());
        System.out.println("My fish says " + myFish.getSound());
        // System.out.println("My fish weighs " + myFish.get...); Don’t have this yet!

        System.out.println();
        if (myBird.doesItFly()){
            System.out.println("My bird flies");
        } else {
            System.out.println("My bird does not fly");
        }

        System.out.println("My bird says " + myBird.getSound());
    }
}
• Not every “is-a” relationship should use inheritance. Example: a square is a rectangle, but having width and height as separate data fields isn’t necessary.

• Also, don’t use inheritance if the objects don’t have an “is-a” relationship. Example: a Butterfly class shouldn't extend from a Bird class, even if the data is very similar
Practice: Inheritance

• Create a subclass Bug that extends the Animal class and has the following properties:
  • a private variable legCount for how many legs it has
  • a constructor where you can pass in a leg count, name, sound and weight
  • a getter and setter for the legCount variable
Super!

• The constructors of a superclass are not inherited by the subclass

• To call constructors or methods of a superclass, you can access the class using super. (very similar to the this keyword)
Fish(boolean livesInOcean, String name, String sound, double height, double weight){
    super(name, sound, height, weight);
    this.livesInOcean = livesInOcean;
}

Super!

- The `super()` call must be the first line of subclass constructor, and it is executed before the current constructor does anything else.

- If you don’t put one in your subclass, the compiler puts one in for you behind the scenes.
Inheritance

• Note: if a class will be extended, make sure you have a no-arg constructor! Otherwise, you may run into an error

```java
public class Apple extends Fruit {
}

class Fruit {
    public Fruit(String name) {
        System.out.println("Fruit's constructor is invoked");
    }
}
```
Overriding Methods

- Sometimes, a subclass’s methods should perform a bit differently than the superclass.

- To do this, we can use *method overriding*.

- Method overriding is when you have a method in a subclass that has the same signature as in the superclass. If you call it from a subclass, the subclass method will be executed.

- In order to call the superclass’ method, you use `super.methodName()`.
Overriding Methods

- Private methods cannot be overridden because they aren’t accessible to the subclasses.

- Static methods can be inherited, but not overridden. If a static method is named the same, the superclass method is hidden but can be accessed by using `SuperClassName.StaticMethodName()`.
Overriding vs Overloading

public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overrides the method in B
    public void p(double i) {
        System.out.println(i);
    }
}

(a)

(b)
Overriding vs Overloading

• Note: You can use @Override annotation to make sure you don’t make a mistake!

```java
public class CircleFromSimpleGeometricObject
  extends SimpleGeometricObject {
    // Other methods are omitted

    @Override
    public String toString() {
      return super.toString() + "\nradius is " + radius;
    }
  }
```
Inheritance

- What happens if you don’t specify a superclass?
Object class

• Every class is java is descended from java.lang.object

• There is an implicit subclassing

• toString() method
Overriding Methods

- Let’s add a `toString()` overriding method to our animals