Lesson 1 Introduction to inheritance

# Learning goals

1. Design and write classes that inherit from superclasses.
2. Override methods inherited from a superclass.
3. Write subclass constructors that call superclass constructors.
4. Write applications that make use of class-inheritance, e.g. an animal taxonomy program.

# Agenda

1. We’ve seen inheritance before:
   1. JFrame 🡪 GOLapp
   2. GOLapp inherits all fields and methods of JFrame but can add new ones.
2. Extending the Dog class in Java (on white board):
   1. Recap of the Dog and DogTester classes
   2. Making a Chihuahua class using the Java keyword *extends*
   3. How Chihuahua inherits fields from Dog
   4. Adding fields that are unique to Chihuahuas
   5. The Chihuahua constructor
   6. How Chihuahua inherits methods from Dog
   7. Inherited method (describe()) vs. overridden methods (bark())
3. Practice: Students code the Chihuahua class and modify main() in DogTester class
4. Take up the working program
5. Practice Exercises #5-1

## Recap of Dog class

class **Dog** {

String name, gender, breed, emoState;

**Dog**( String n, String g, String b, String es ) {

this.name = n;

this.gender = g;

this.breed = b;

this.emoState = es;

}

void **describe**() {

println( name + “ is a ” + gender + “ ” + breed + “ and is feeling ” +   
 emoState + “ today.”);

}

void **bark**() {

sout( “Woof!” );

}

}

void setup () {

Dog patrick = new Dog(“Patrick”, “male”, “lab”, “happy”);

patrick.describe(); //prints *Patrick is a male lab and is feeling happy today*

patrick.bark();

}

## Let’s add a Chihuahua class to this package.

class Chihuahua **extends Dog** { //Making Chihuahua a *subclass* of Dog.

//This means Chihuahua has the same fields and //methods of Dog, plus any others we add below.

int numSweaters; //A field unique to Chihuahuas. Other Dog objects don’t have this.

//Thus, every Chihuahua object has 5 fields: numSweaters plus the 4

//fields it inherits from Dog, its superclass.

Chihuahua( String n, String g, int nS ) {

**super**( n, g, “chihuahua”, “nervous” ); //Calls the Dog constructor with 4 arguments,

//which sets *name* to n, *gender* to g, *breed* to

//“chihuahua” and *emoState* to “nervous”

this.numSweaters = nS; //Setting the 5th field value

}

//More methods to come. For now, we’ll just have a constructor.

}

## Making and using subclass objects in main()

void setup() {

Chihuahua luana = new Chihuahua( “Luana”, “female”, 3); //Calls the Chihuahua constructor,

// which in turn calls the Dog constructor

// with 4 arguments: “Luana”, “female”,

// “chihuahua” and “nervous”.

// and then sets r.numSweaters to 3.

luana.describe(); //Since luana is a Chihuahua object, it is also a Dog object, which means

//it has a describe() method. When Java sees this line, it first looks in the

//Chihuahua class for a describe() method. If it finds none, it looks in the superclass.

//If it finds a describe() method there, Java will use that one.

//Thus, Java will print *“Luana is a female chihuahua and is feeling nervous today.”*

## Overriding a superclass method & adding new methods

Now, chihuahuas don’t really bark. They yipe. So if we called luana.bark() inside setup(), the “woof” Luana makes would not be realistic. We can solve this problem with an OOP feature called *overriding*.

In Processing/Java, we can *override* a superclass method inside a subclass. For example, let’s override the bark() method from the Dog class inside the Chihuahua class, so that it makes chihuahuas yipe instead of woof.

public class Chihuahua extends Dog {

int numSweaters;

Chihuahua( String n, String g, int nS ) {

super( n, g, “chihuahua”, “nervous” );

this.numSweaters = nS;

}

**void** **bark**() { //Overriding the bark() method from the Dog class

println( “Yipe!” );

}

**void** **describe**() { //Overriding the describe() method from the Dog class

**super.describe();** //Calls the describe() method from the Dog class

println( name + “ owns ” + numSweaters ); //Does more stuff that Dog.describe() doesn’t

}

**void chewSweater**() { //Adding a method that’s unique to Chihuahua objects.

numSweaters = max( numSweaters – 1, 0 );

}

}

Let’s use these new methods inside setup()

void setup() {

Dog Patrick = new Dog( “Patrick”, “male”, “lab”, “happy”);

Patrick.describe(); //prints “*Patrick is a male lab and is feeling happy today”*

Patrick.bark(); //prints *“Woof!”*

Chihuahua luana = new Chihuahua( “Luana”, “female”, 3);

luana.bark(); // prints “*Yipe*!”

luana.describe(); // prints *“Luana is a female chihuahua and is feeling nervous today. Luana owns 3 sweaters*.”

luana.chewSweater();

luana.describe(); // prints *“Luana is a female chihuahua and is feeling nervous today. Luana owns 2 sweaters*.”

Patrick.chewSweater(); // ERROR! The Dog object Patrick is not a Chihuahua object, so it does not

// have a chewSweater() method.

## More examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Superclass** | **Subclass** | **Inherited fields & methods** | **New fields & methods** |
| Dog | Chihuahua | name  emoState  bark() | numSweaters  bark() (overridden)  chewSweater() |
| School | HighSchool | principal  registerStudent() | universityAcceptanceRate  expel() |
| Book | Textbook | numPages  publisher  openToPage()  search() | studentUser  issueTo()  returnToSchool() |