

> Lesson Summary and Outcomes

Students will learn to model and implement inheritance through the zoo activity. They will begin with a general animal class with common instance variables (properties) and methods (functionalities). Then they will “extend” the general class to build more specific classes – examples are given for cow and camel. Students may then add other classes to the Animal hierarchy, testing as they do so.

> Student Assessment

How well do students perform each task:

- Add at least three more subclasses of animal.
- Figure out a way to enhance each new class with at least two new instance variables and overriding `toString()` methods.
- Create at least one instance of each class and ensure it is inheriting everything properly.
- Complete the Zoo driver class as described above
- Comment everything, and save, compile, and test OFTEN!

> Prerequisite Knowledge and Skills

BlueJ:

- Creating projects
- Adding new classes
- Using BlueJ's built-in object bench for instantiating objects
- Testing methods from instantiated objects

Java:

- Class structure including methods
- Inheritance
- Adding new classes and subclasses
- Driver programs for testing
- Arraylists

> This Lesson Targets the Following Subject Areas(s):

Pre-Java	Java Programming
<input type="checkbox"/> Hardware Basics	<input type="checkbox"/> Applet Programming
<input type="checkbox"/> Software Basics	<input type="checkbox"/> Subroutine Programming
<input type="checkbox"/> Networks and Servers	<input checked="" type="checkbox"/> Full Scale Programming
<input type="checkbox"/> HTML	
<input type="checkbox"/> Action Scripting	
<input type="checkbox"/> Java Scripting	

> Curriculum-Framing Questions

Essential Question

- What is Inheritance?
- What exactly is a driver program?

Activity Questions

- Why do you use inheritance?
- What other subclasses might make sense here?
- How do you group all the “animals” together ?

Sample Content Questions

- How do you implement inheritance?
- What does a subclass get from its parent?

> Targeted Content Standards, Benchmarks, or State Frameworks

Colorado State Standards:

Science:

- Standard #3 - Students know and understand the characteristics and structure of living things, the processes of life and how living things interact with each other and their environment.
 - 3.1 Students know and understand the characteristics of living things, the diversity of life, and how living things interact with each other and with their environment
- 3.1.1 As students in grades 9-10 extend their knowledge, what they know and are able to do includes: Using and producing a variety of classification systems for organisms (for example, the five-kingdom classification, classification based on behavior).

International Society of Technology Education (ISTE):

1. Information Literacy Standards:

- Standard 1: The student who is information literate accesses information efficiently and effectively.
- Standard 2: The student who is information literate evaluates information critically and competently.

2. Technology Foundation Standards for Students:

- **Standard 3:** Technology productivity tools
 - Students use technology tools to enhance learning, increase productivity, and promote creativity.
 - Students use productivity tools to collaborate in constructing technology-enhanced models, prepare publications, and produce other creative works.

- 3. Science Standards for Students Grades 9-12
- Content Standard C: Life Science
 - C1. The cell
 - C2. Molecular basis of heredity
 - C3. Biological evolution
 - C4. Interdependence of organisms
 - C5. Matter, energy, and organization in living systems
 - C6. Behavior of organisms

> Materials and Resources Required for Project

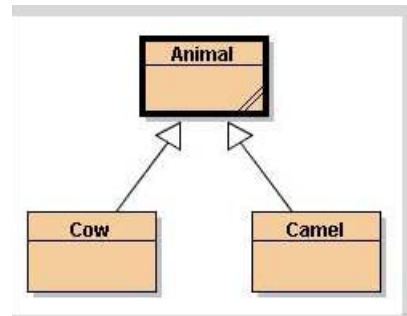
Equipment	<ul style="list-style-type: none"> • Computer for each student with BlueJ and Java SDK installed on it
Consumable Supplies	<ul style="list-style-type: none"> • Diskettes and CD-ROMs
Textbooks/Lesson Guides	<ul style="list-style-type: none"> • Handouts of lessons
Technology	<ul style="list-style-type: none"> • Windows machines running Windows 98 SE or higher
Internet Resources	<ul style="list-style-type: none"> • Numerous websites on the internet show applets in action

> Lesson Plan Outline

In this activity, students will model the animal hierarchy. They should begin with the base class – animal – a generic class from which all others will ultimately inherit. This is the superclass that all the other classes will “extend”: When creating a model using inheritance, start by defining a more general model with common instance variables and methods. The more specific classes will be told to extend or inherit from the general class. The general class then becomes the superclass and the others become the subclasses. The subclasses automatically inherit the instance variables and methods from the superclass.

To accomplish this, students should create a new project called Zoo and then within that project create a class called Animal. There is a sample in the handout. Once that is keyed in and tested – using the Object Bench on BlueJ – students should go ahead and create the other classes – cow and camel. These are also illustrated in the handout. Make sure each class has a `toString()` method geared to the specifics of the class – this is the vehicle for displaying information about a specific animal.

The result is an inheritance hierarchy; a simple diagram of such a hierarchy is illustrated here. In this case the Animal class is the superclass. It has two subclasses, Cow and Camel. This doesn't work in reverse. A subclass can extend only one class.



When all of the animal classes have been coded and tested, students should create the zoo “driver” class. The zoo class will allow for all of the animals to be contained in a single `ArrayList`. The zoo class will need at least the following:

- one single property or instance variable, an `ArrayList` named `residents`
- a `Zoo()` constructor method which
 - creates the `residents ArrayList`
 - creates at least one instance of each animal subclass you customized

earlier

- adds each instance to the `ArrayList`
- a `getResidents()` method which
 - uses an iterator to pull each animal from residents
 - calls the `toString()` method to show each animal's specific information
- a `main()` method that creates the `Zoo` object and calls its `getResidents()` method

Tips, Hints and Tricks –. Get creative and make the program more robust – add classes that inherit from `Cow` or `Camel` or perhaps new classes on the same inheritance level as `Cow` and `Camel` – maybe a `Dog` class with subclasses for specific types of dogs

> **Pacing/Timeline**

This lesson will require 4-7 hours

- Basic animal hierarchy 2-3 hours
- Driver class coded and tested 1-2 hours
- Three new classes added 1-2 hours

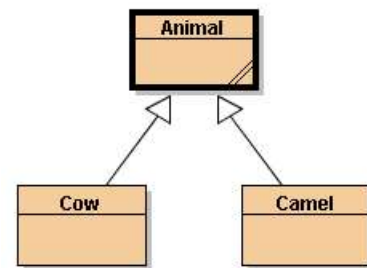
> **Teacher Reflection** (For example, what worked well in this lesson? What would you change if you were to teach it again?)

- Do students understand the basic zoological hierarchy concepts introduced here?
- Do students understand the concept of inheritance?
- Are they able to suggest and expansion or refinement of the hierarchy?

> Student Handout - Java Programming Assignment - Inheritance

It's good programming style to keep the amount of code to a minimum and to reuse existing code as much as possible. Code that is economical is easy to maintain. The concept of inheritance does just that: it illustrates a way to make code both economical and extendable.

When creating a model using inheritance, you start by defining a more general class with common instance variables and methods. The more specific classes are told to “extend” the general class. The general class then becomes the *superclass* and the others become the *subclasses*. The subclasses automatically inherit the instance variables and methods from the superclass.



The result is called the *inheritance hierarchy*; a simple such hierarchy is shown here. In this case, the Animal class is the superclass. It has two subclasses, Cow and Camel. (This doesn't work in reverse. A subclass can extend only one class.)

Let's go ahead and start to model this animal hierarchy. Create a new project in BlueJ named Zoo, and a class called Animal. Enter the following code into the Animal class. This is the superclass that all specific animals will extend:

```

/**
 * Animal.java is the superclass. All specific animals
 * inherit their * base properties and methods from here.
 * @author (your name)
 * @version (a version number or a date)
 */
public class Animal { // instance variables
    private String myName; // subclasses use getName() to access
    private String mySound;
    /**
     * Constructor for objects of class Animal
     */
    Animal() // default (no argument) constructor blanks
        // everything out
  
```

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

```

    {
        myName = ""; // initialize instance variables
        mySound = "";
    }
    Animal(String newName, String newSound) // constructor
        // takes name and sound
    {
        myName = newName; // initialize instance variables
        mySound = newSound;
    }
    /**
    * Accessor methods
    * getName() returns this animal object's name
    */
    String getName()
    {
        return myName;
    }
    /**
    * Accessor methods
    * getSound() returns this animal object's sound
    */
    String getSound()
    {
        return mySound;
    }
    /**
    * toString() returns this animal object's information as a
    * String
    */
    public String toString()
    {
        return myName + " says " + mySound; // send name,
            // sound to the caller
    } // end of toString()
} // end of Animal superclass

```

Now you have the superclass defined; let's define a subclass. Create a new class named Cow and enter this code:

```

/**
* Cow.java is a subclass of animal.
* @author (your name)
* @version (a version number or a date)
*/
public class Cow extends Animal // keyword "extends" makes this
    // a subclass

```

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

```

{ // no instance variables; all are inherited from Animal

    /**
     * Constructor for objects of class Cow
     * Takes two parameters for the cow's name and sound.
     */

    Cow(String newName, String newSound)
    { // initialize inherited instance variables using the
      // superclass' constructor
      super(newName, newSound);
    }
    // no methods needed; all are inherited from Animal
}

```

That was a very easy class. It inherited everything. Try the Camel class now, which needs a little enhancing. We will add an instance variable for the number of humps, and fix `toString()` to display that number, both of which are specific to the Camel class.

```

/**
 * Camel.java is a subclass of animal with modifications.
 * @author (your name)
 * @version (a version number or a date)
 */
public class Camel extends Animal
{
    // one instance variable; all others are inherited from
    // Animal
    int myHumps;
    /**
     * Constructor for objects of class Camel
     * Takes three parameters, the camel's name, sound, and #
     * of humps.
     */
    Camel(String newName, String newSound, int newHumps)
    {
        // initialize the inherited instance variables
        super(newName, newSound); // call the superclass'
        // constructor
        myHumps = newHumps; // and set the Camel specific
        // property
    }
    // One method, which is the overridden toString() method
    // All others are inherited from Animal
    public String toString()
    {
        String message = ("The camel " + getName() + " says ");

```

Sun, Sun Microsystems, the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. This Java activity was developed for The Sun Foundation by Boulder High School 2004.

```

        message += getSound() + " and has " + myHumps + "
                    humps.";
        return message;
    }
    //end of toString()
} // end of Camel class

```

You should have a good collection of working subclasses of the `Animal` class. You should also have tested an instance of each subclass to make sure its specific properties and `toString()` methods work correctly. Once that part is done, all that's left is to create a driver class that will take care of creating the animals in the zoo and displaying a list.

With the Zoo project still open add a class called `Zoo`. Your `Zoo` class will need at least the following:

- one single property or instance variable, an `ArrayList` named `residents`
- a `Zoo()` constructor method which
 - creates the `residents ArrayList`
 - creates at least one instance of each animal subclass you customized earlier
 - adds each instance to the `ArrayList`
- a `getResidents()` method which
 - uses an iterator to pull each animal from `residents`
 - calls the `toString()` method to show each animal's specific information
- a `main()` method that creates the `Zoo` object and calls its `getResidents()` method.

Here is a possible outline of the Zoo class.

```

/**
 * Zoo.java is the driver class. This class creates all
 * animal instances.
 * @author (your name)
 * @version (a version number or a date)
 */
public class Zoo
{
    // instance variables
    private ArrayList residents; // holds all animal instances
    /**
     * Constructor for objects of class Zoo
     */
    public Zoo()
    {
        // initialize instance variable
        // create animals
        // add to residents
    } // end of constructor
    /**
     * getResidents() displays the info of each animal instance
     */
    public void getResidents()
    {
        // use an iterator, to move through the ArrayList
    } // end of getResidents()
} // end of Zoo class

```

> YOUR JOB:

- Add at least three more subclasses of Animal.
- Find a way to enhance each new class with at least two more instance variables and overriding `toString()` methods.
- Test creating at least one instance of each class and ensure it is inheriting everything properly.
- Complete the zoo class as described above
- Comment everything, and save, compile, and test OFTEN!



> EXTRAS YOU CAN ADD:

- Get creative and make the program better. For instance, inheritance doesn't need to stop with one super and subclass. Any class can have subclasses. Try making a subclass of one of the existing specific classes.