Chapter 2
An Overview of Java

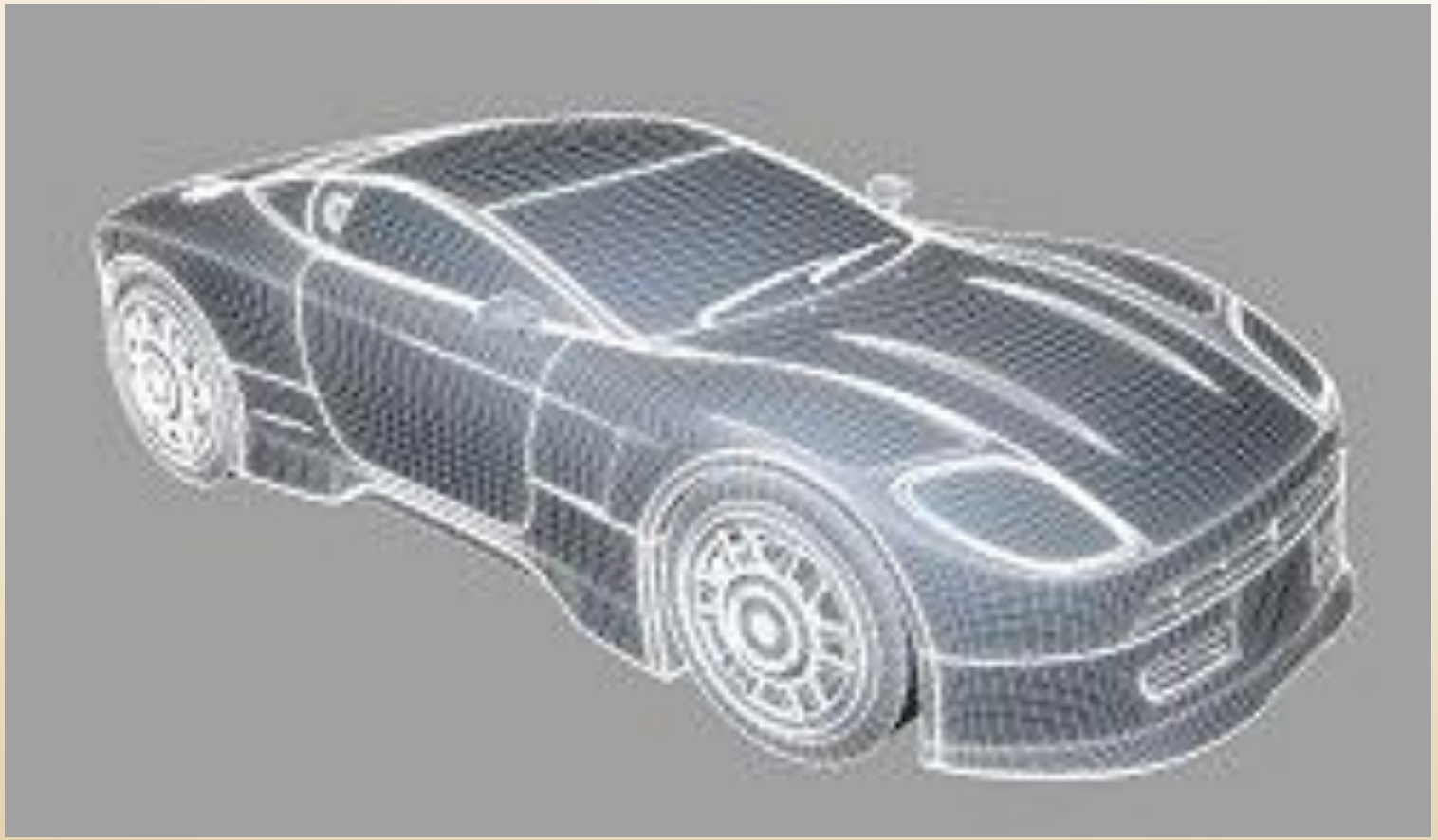
Object-Oriented Programming

- Object-oriented programming (OOP) is at the core of Java.
- **Two Paradigms**
  - All computer programs consist of two elements:
    - code
    - Data
  - some programs are written around "what is happening" and others are written around "who is being affected."
  - These are the two paradigms that govern how a program is constructed.
  - **first way** is called the *process-oriented model*
    - process-oriented model can be thought of as *code acting on data, like C language*
  - To manage increasing complexity, the **second approach**, called *object-oriented programming*, was conceived
    - Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data
    - An object-oriented program can be characterized as *data controlling access to code*.

Object-Oriented Programming - **Abstraction**

- Humans manage complexity through abstraction
- For example, people do not think of a car as a set of tens of thousands of individual parts
- to manage abstraction is through the use of **hierarchical classifications**
  - Break the complex system into more manageable pieces.
    - Like car is a single object and car consists of several subsystems
- **Hierarchical abstractions** of complex systems can also be applied to computer programs.
- The data from a traditional process-oriented program can be transformed by abstraction into its **component objects**.
- A sequence of process steps can become a collection of **messages** between these objects.

Object-Oriented Programming - **Abstraction**



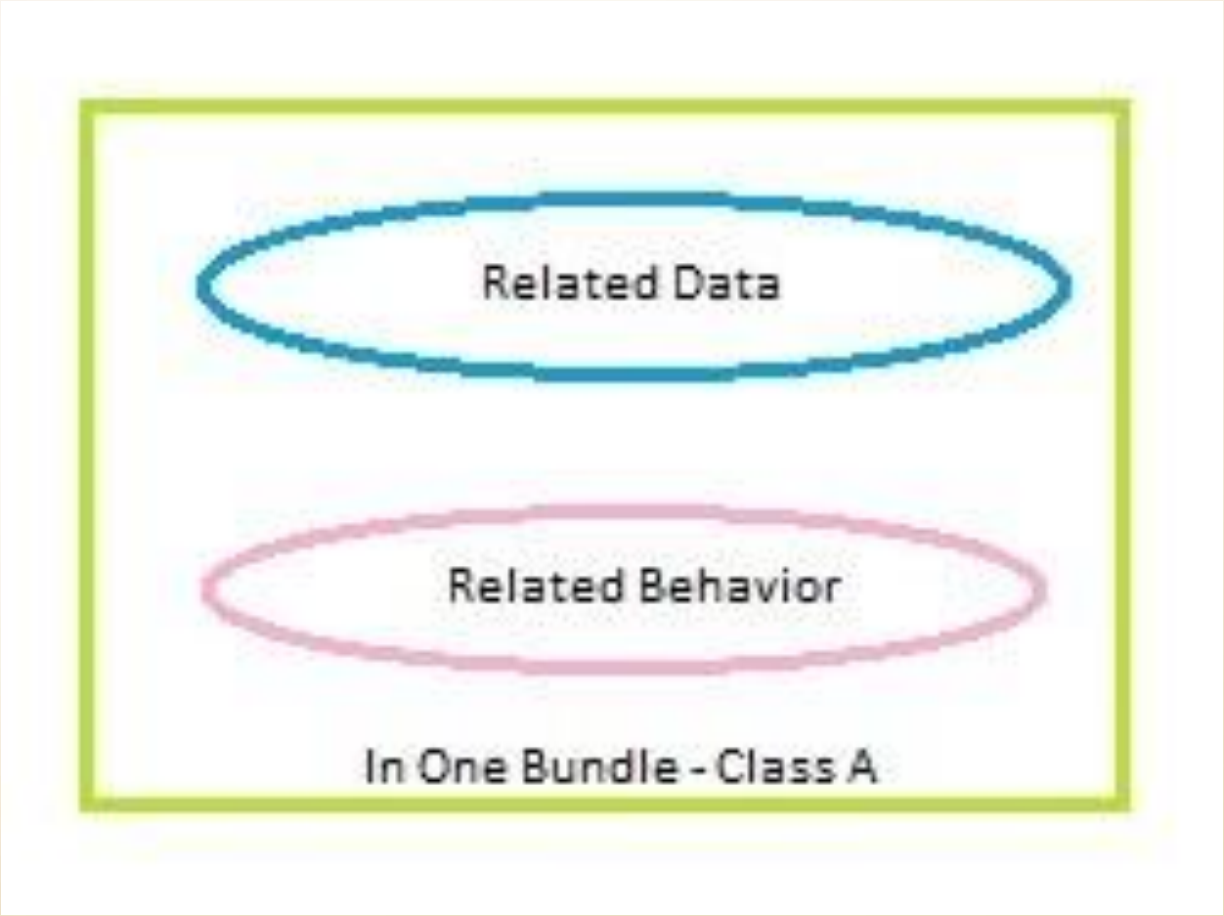
OOP - **Three OOP Principles**

- Object-oriented languages provide ***mechanisms*** to implement object-oriented model
  - Encapsulation
  - Inheritance
  - Polymorphism

1. Encapsulation

- *Encapsulation* is the mechanism that ***binds together code and the data*** it manipulates, and keeps both safe from outside interference and misuse.
- Example, automatic transmission on an automobile
- The power of *encapsulated code* is that ***everyone knows how to access it*** and thus can use it regardless of the implementation details—and without fear of unexpected side effects.
- In ***Java***, the basis of encapsulation is the ***class***.
- A ***class*** defines the structure and behavior (data and code) that will be shared by a set of objects.
- each object of a given class contains the structure and behavior defined by the class
- objects are sometimes referred to as *instances of a class*

1. Encapsulation

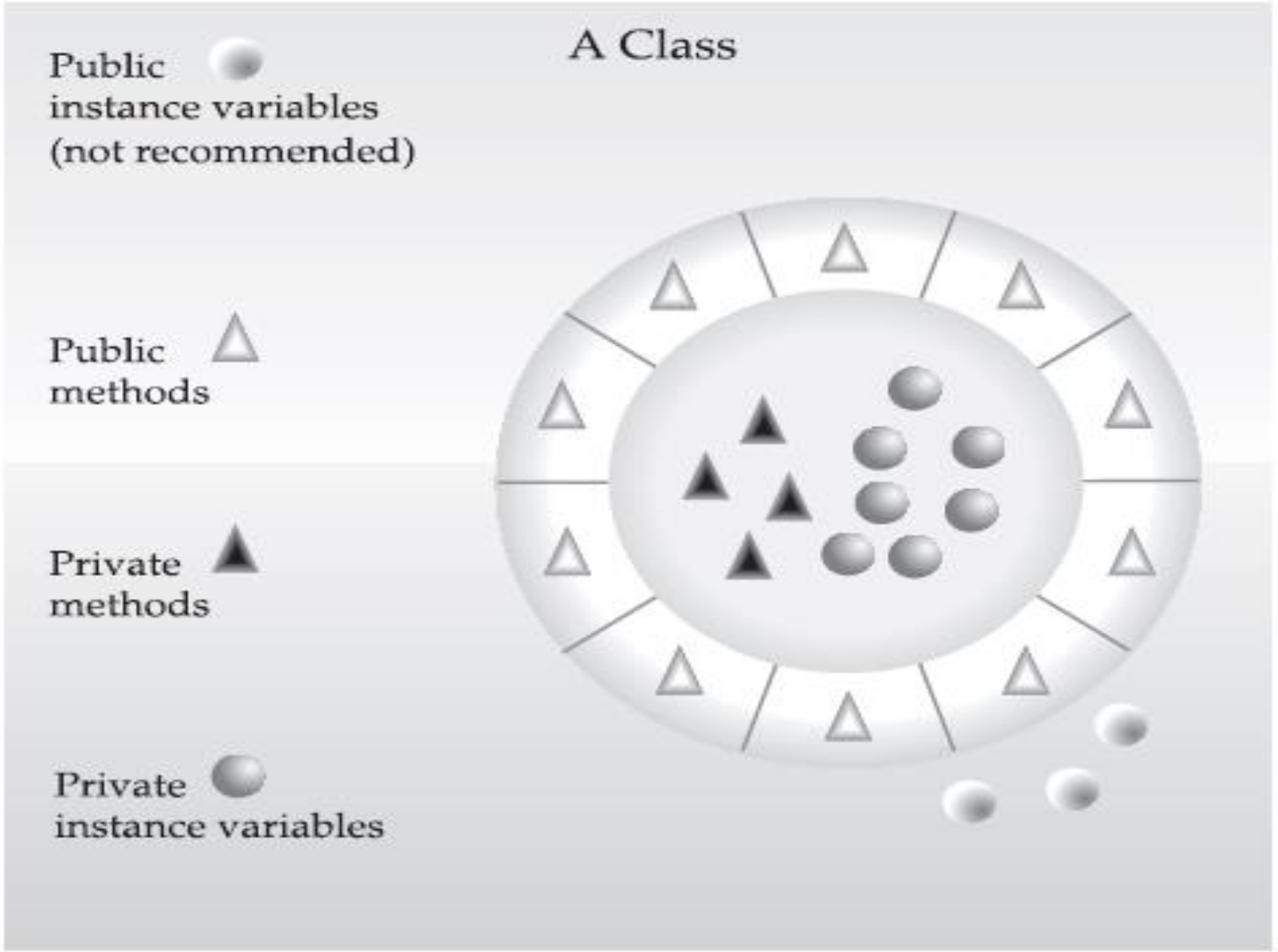


1. Encapsulation

- A class is a logical construct and an object has physical reality.
- Elements / Members of the class:
  - Data → *member variables* or *instance variables*
  - Code → *member methods* or *methods*
- Each method or variable in a class may be marked ***private or public.***
- *public* interface of a class represents everything that external users of the class need to know, or may know
- *private* methods and data can only be accessed by code that is a member of the class

1. Encapsulation

FIGURE 2-1
Encapsulation: public methods can be used to protect private data

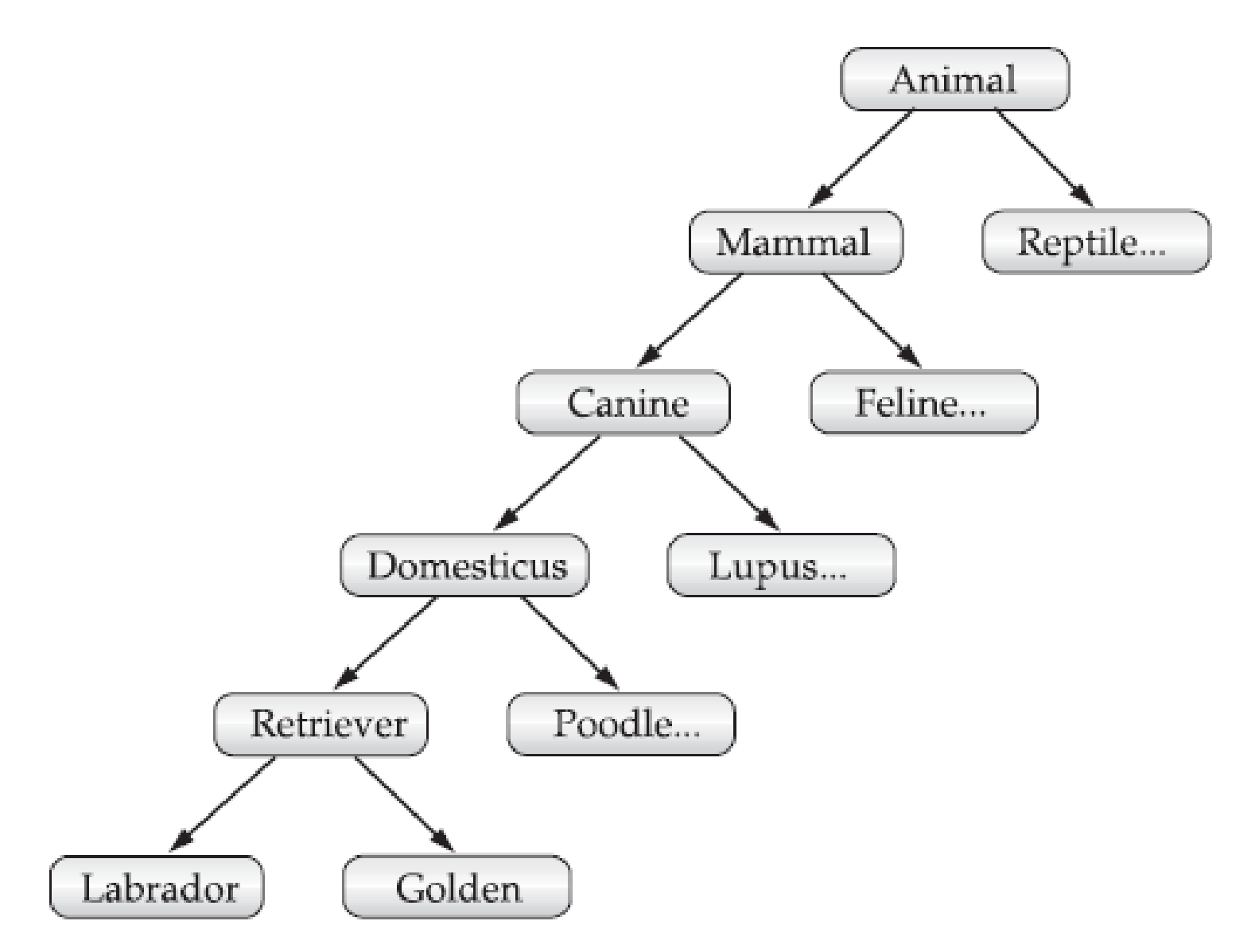


2. Inheritance

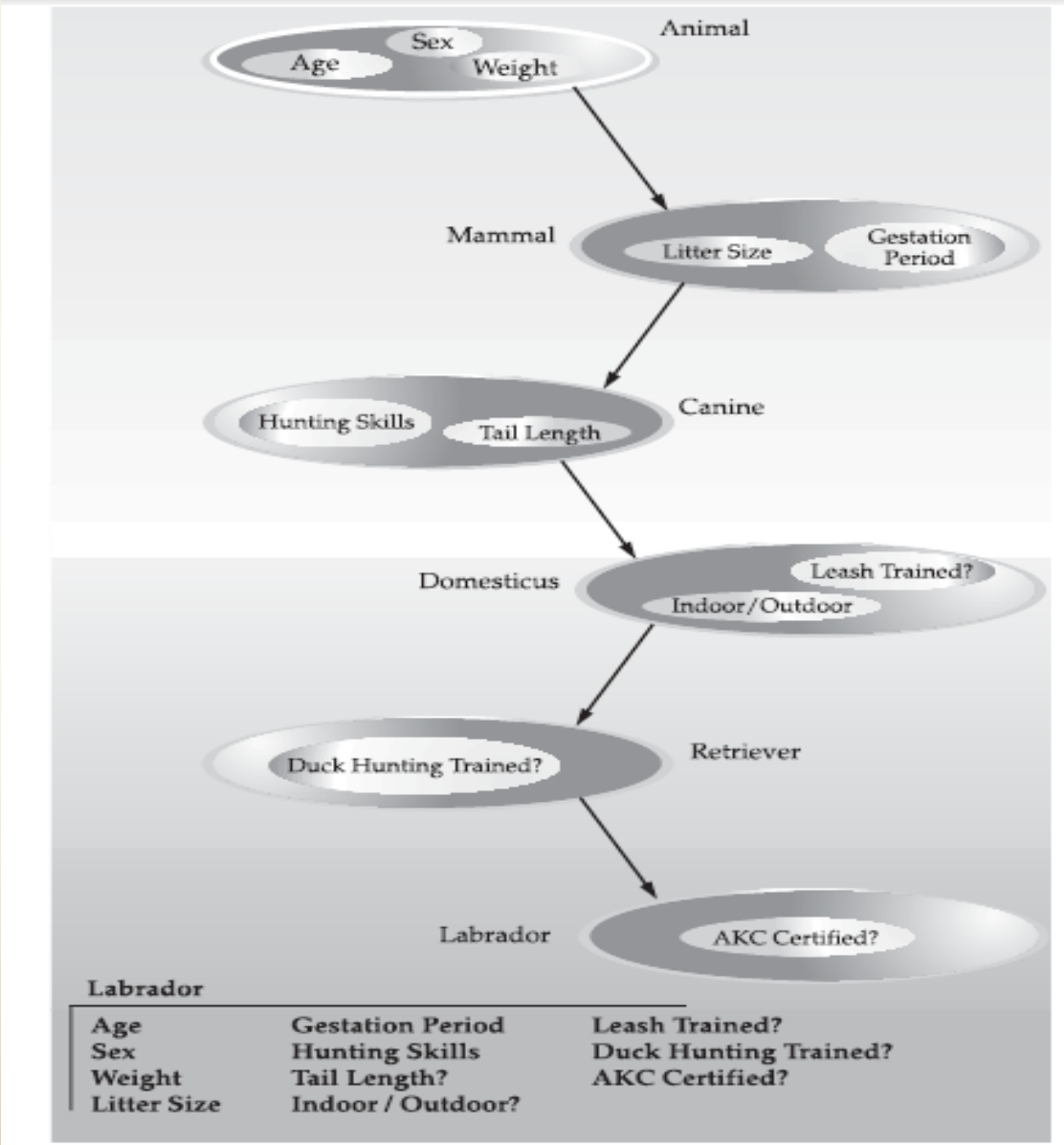
- *Inheritance* is the process by which one object acquires the properties of another object.
- most knowledge is made manageable by hierarchical (that is, **top-down**) classification
- By use of inheritance, an object need only define those qualities that make it unique within its class
- It can inherit its general attributes from its parent.

- Inheritance interacts with encapsulation as well.

2. Inheritance



2. Inheritance



Labrador inherits the encapsulation of all its superclasses

3. Polymorphism

- *Polymorphism* from Greek meaning "**many forms**"
- *Polymorphism* is a feature that allows one interface to be used for a general class of actions.
- One interface, multiple methods
- E.g. both a Plane and a Car could respond to a 'turnLeft' message,
  - however the means of responding to that message (turning wheels, or banking wings) is very different for each.
- **Advantages of Polymorphism**
  - Generics: Enables generic programming.
  - Extensibility: Extending an already existing system is made simple.
  - De-clutters the object interface and simplifies the class blueprint.

Polymorphism, Encapsulation, and Inheritance Work Together

A First Simple Program

- *Create* the program by typing it into a text editor and saving it to a file named, say, Example.java.

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
class Example {
   // Your program begins with a call to main().
   public static void main(String args[]) {
   System.out.println("This is a simple Java program.");
   }
}
```

A First Simple Program

- *Keep in mind that Java is case-sensitive.*
- In Java, a source file is officially called a *compilation unit*.
- In Java, all code must reside inside a class.

- ***Compile*** it by typing "javac Example.java" in the terminal window.

  `% **javac Example.java**

- ***Run*** (or *execute*) it by typing "java Example" in the terminal window.

  % **java Example**

Output :

This is a simple Java program.

A First Simple Program

public static void main(String args[]) {

- All Java applications begin execution by calling **main( ).**
- **public** keyword is an *access specifier*
- **public**, then that member may be accessed by code outside the class in which it is declared
- The keyword **static** allows **main()** to be called without having to instantiate a particular instance of the class (/object).
- **void** – return type
- **String args[ ]** declares a parameter named **args**, which is an array of instances of the class **String**. Used in command line arguments.

A First Simple Program

System.out.println("This is a simple Java program.");

- **System** is a predefined class that provides access to the system
- **out** is the output stream that is connected to the console
- **println()** - displays the string which is passed to it followed by a new line on the screen

A second short program

```
/*
Here is another short example.
Call this file "Example2.java".
*/
class Example2 {
public static void main(String args[]) {
int num;              // this declares a variable called num
num = 100;            // this assigns num the value 100
System.out.println("This is num: " + num);
num = num * 2;
System.out.print("The value of num * 2 is ");
System.out.println(num);
}
}
```

Output of the program:

This is num: 100

The value of num * 2 is 200

Two Control Statements

if Statement

- Simplest form:

  if(*condition*)

  statement;

- Example:

  if(num < 100)

  System.out.println("num is less than 100");

if Statement

```
/* Demonstrate the if.
Call this file "IfSample.java". */


class IfSample {
public static void main(String args[]) {
int x, y;
x = 10;
y = 20;
if(x < y) System.out.println("x is less than y");
x = x * 2;
if(x == y) System.out.println("x now equal to y");
x = x * 2;
if(x > y) System.out.println("x now greater than y");
// this won't display anything
if(x == y) System.out.println("you won't see this");
 }
}
```

The output generated by this program is shown here:

```
x is less than y
x now equal to y
x now greater than y
```

for Loop

- Simplest form of for loop

  for(*initialization; condition; iteration*)

  statement;

for Loop

```
/*
Demonstrate the for loop.
Call this file "ForTest.java".
*/
class ForTest {
    public static void main(String args[]) {
        int x;
        for(x = 0; x<10; x = x+1)
            System.out.println("This is x: " + x);
    }
}
```

This program generates the following output:

```
This is x: 0
This is x: 1
This is x: 2
This is x: 3
This is x: 4
This is x: 5
This is x: 6
This is x: 7
This is x: 8
This is x: 9
```

Using Blocks of code

- two or more statements to be grouped into *blocks of code*, also called *code blocks*
- Done by enclosing the statements between opening and closing curly braces
- Example:

```
if(x < y) { // begin a block
    x = y;
    y = 0;
} // end of block
```

Lexical Issues

- Java programs are a collection of
  - Whitespace
  - Identifiers
  - Literals
  - Comments
  - Operators
  - Separators
  - keywords

Lexical Issues - Whitespace

- Java is a free-form language.
- whitespace is
  - Space
  - Tab
  - newline

Lexical Issues - Identifiers

- Identifiers are used for class names, method names, and variable names.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- They must **not** begin with a number.
- Example:
  - AvgTemp
  - $test

Lexical Issues - Literals

- A ***constant value*** in Java is created by using a *literal* representation of it.
- For Example,
  - 100
  - 98.6
  - 'x'
  - "Hello"

Lexical Issues - Comments

- **Three types:**
  - Single-line comment → //
  - Multiline comment → /* */
  - Documentation comment
    - used to produce an HTML file that documents your program
    - begins with a **/**** and ends with a ***/**

Lexical Issues - Separators

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Lexical Issues - Java Keywords

- 50 keywords currently
- keywords cannot be used as names for a variable, class, or method
- Java reserves the following: **true**, **false**, and **null** for the values.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Java Class Libraries

- Java environment relies on several built-in class libraries that contain many built-in methods that provide support for such things as I/O, string handling, networking, and graphics

Disclaimer

- These slides are not original and have been prepared from various sources for teaching purpose.

Sources:

- Herbert Schildt, Java™: The Complete Reference

Thank You….