The Object-Oriented Thought Process

Chapter 1

Introduction to Object-Oriented Concepts

Contents

- Procedural Versus OO Programming
- What Exactly Is an Object?
- What Exactly Is a Class?
- Using UML to Model a Class Diagram
- Encapsulation and Data Hiding
- Inheritance
- Polymorphism
- Composition

Procedural Versus OO Programming

- Procedural Programming
  - 절차적 프로그래밍

- Object-Oriented Programming
  - 객체지향 프로그래밍
  - 1960년경부터 시작되어 최근 15년간 꾸준히 비약적으로 성장
    - Java, .NET 등 기술의 성공
    - 전자상거래 관련 기술이 대부분 OO적 특징
    - Web이 OO를 mainstream으로 만드는데 기여

Object Wrappers

Even when there are legacy concerns, there is a trend to wrap the legacy systems in object wrappers.

- Object wrapper : object-oriented code that includes structured code inside.
- Wrappers are used to wrap:
  - Legacy code
  - Non-portable code
  - 3rd party libraries
  - etc,.

What is an Object?

In its basic definition, an object is an entity that contains both data and behavior.

- 속성과 행위
- This is the key difference between the more traditional programming methodology, procedural programming, and O-O programming.

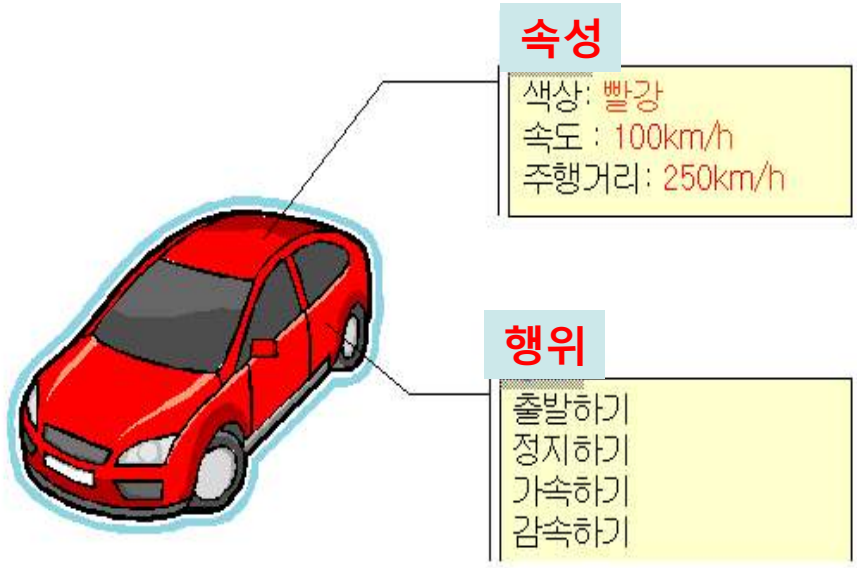


그림 7.3 자동차 객체의 예

OO Programming

- Object 중심으로 실세계를 모델링하여 소프트웨어를 개발하는 방법

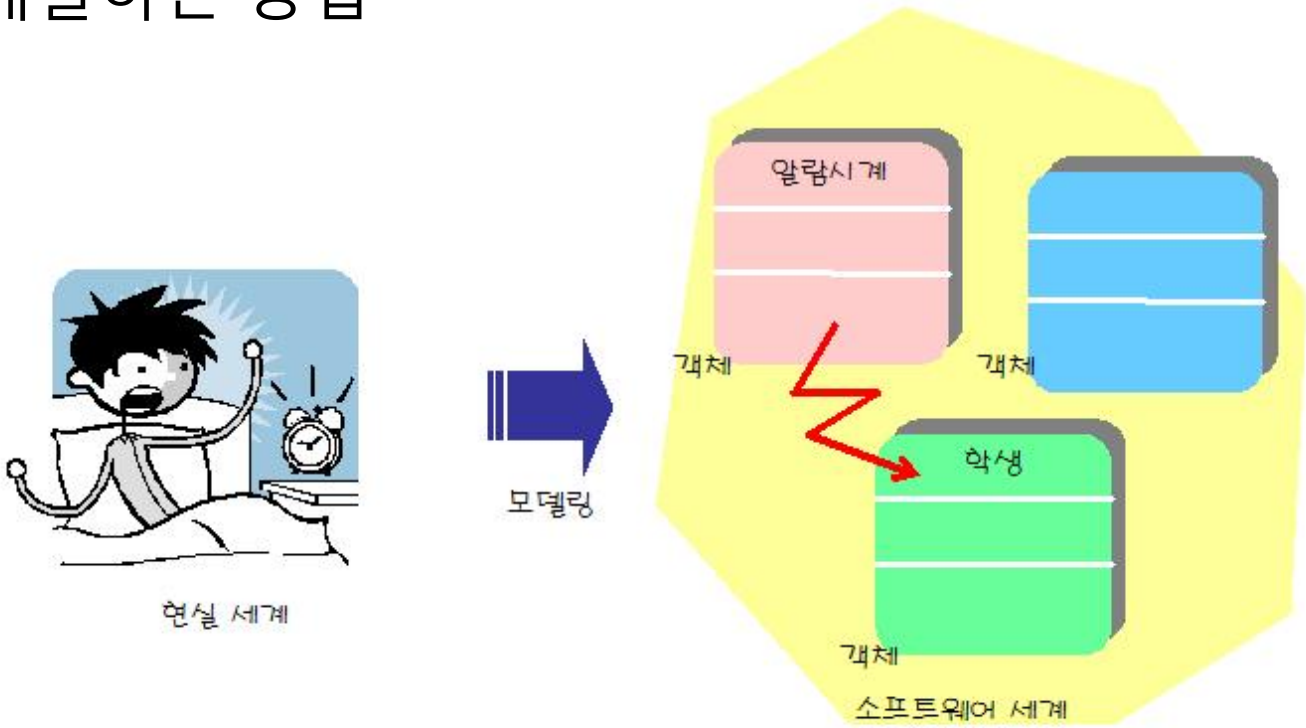


그림 7.1 객체 지향 방법은 현실 세계를 모델링하는 것

Procedural Programming

In procedural programming, code is placed into methods.

– Ideally these procedures then become "black boxes", inputs come in and outputs go out.

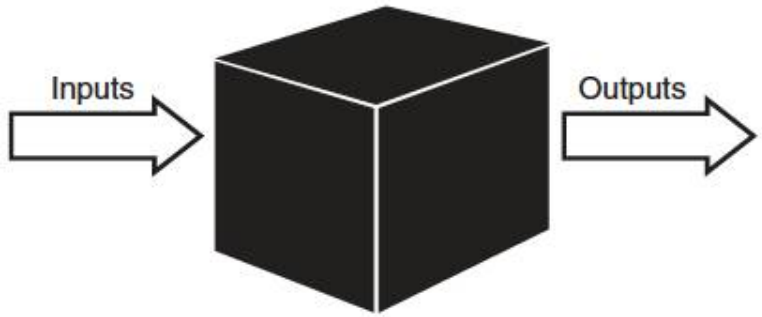
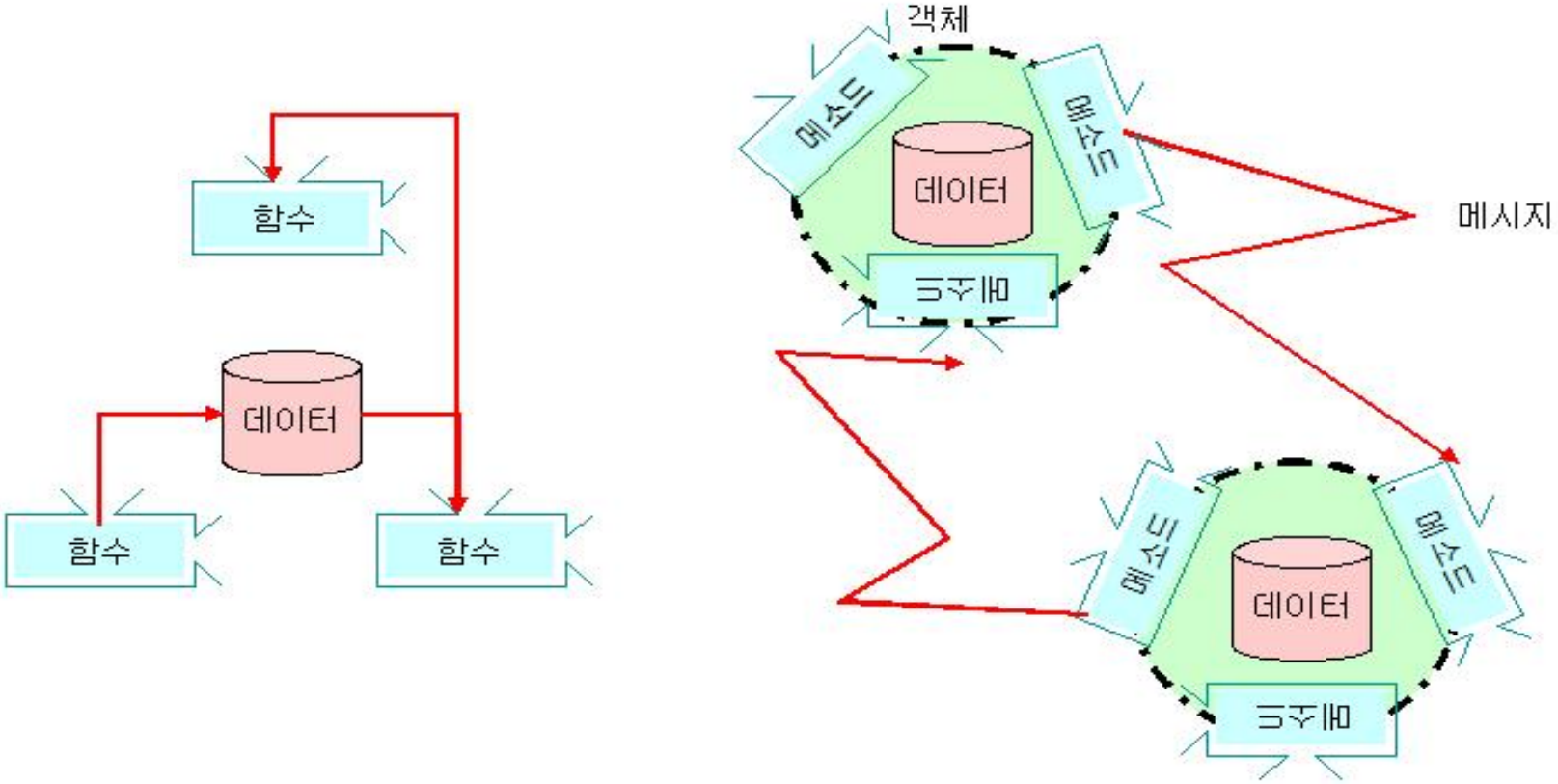


Figure 1.1 Black boxes.

- Data is placed into separate structures, and is manipulated by these methods.

- Difference Between OO and Procedural
  - In OO design, the attributes and behaviors are contained within a single object,
  - whereas in procedural, or structured design, the attributes and behaviors are normally separated.



절차 지향 프로그래밍에서는 데이터와 알고리즘이 묶여있지 않다.

객체 지향 프로그래밍에서는 데이터와 알고리즘이 묶여있다.

그림 7.2 절차 지향적인 방법과 객체 지향적인 방법의 비교

- Separation of data and procedures (functions)
- Causes “unpredictable”

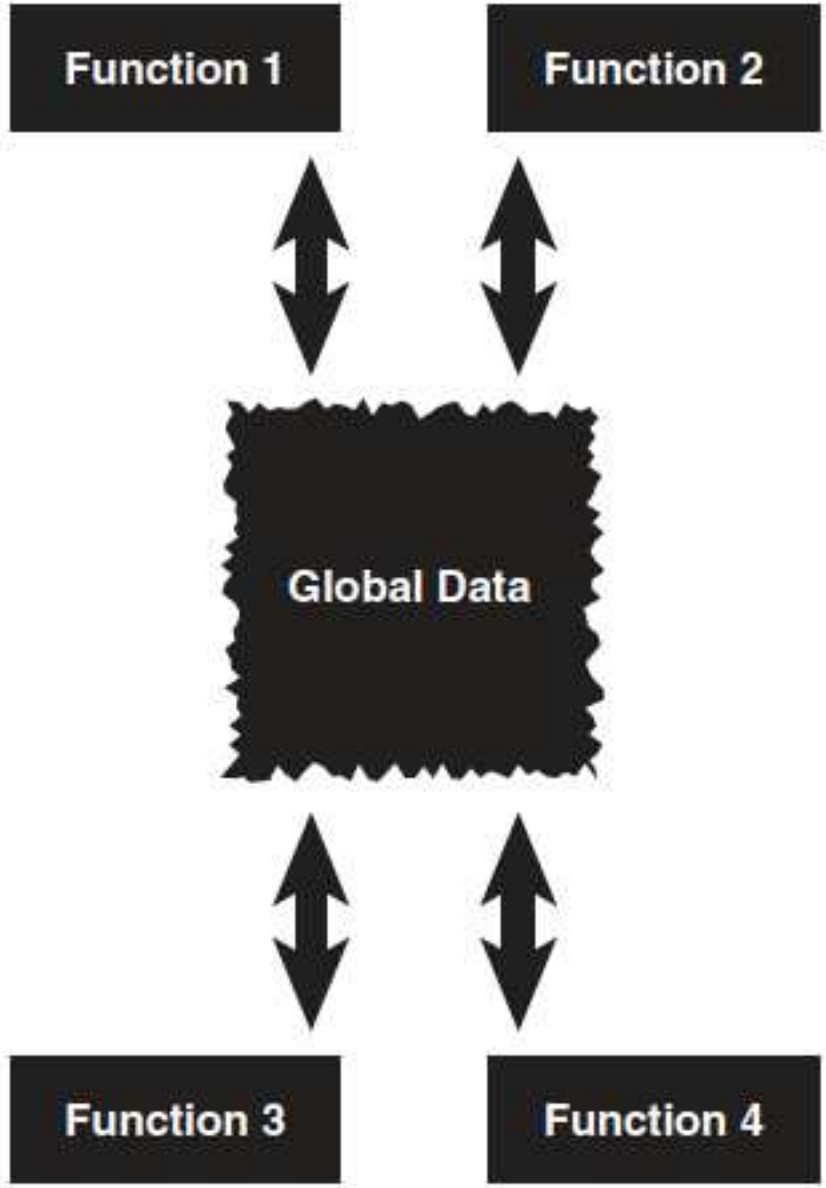


Figure 1.2 Using global data.

Unpredictable?

First, this means that access to data is uncontrolled and unpredictable.

- Second, because you have no control over who has access to the data, testing and debugging is much more difficult.

Nice Packages

Objects address these problems by combining data and behavior into a nice, complete package.

- Objects are not just primitive data types, like integers and strings.

What Exactly Is an Object?

- Objects are the building blocks of an OO program.
- A program that uses OO technology is basically a collection of objects.

Data and Behaviors

Procedural programming separates the data of the program from the operations that manipulate the data.

- For example, if you want to send information across a network, only the relevant data is sent with the expectation that the program at the other end of the pipe knows what to do with it.

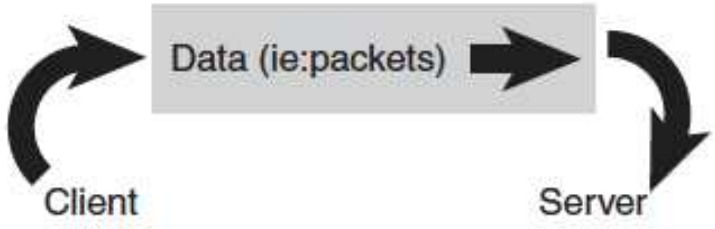


Figure 1.4 Data transmitted over a wire.

O-O Programming

The fundamental advantage of O-O programming is that the data and the operations that manipulate the data are both contained in the object.

– For example, when an object is transported across a network, the entire object, including the data and behavior, goes with it.

Figure 1.5 Objects transmitted over a wire.

Object Data

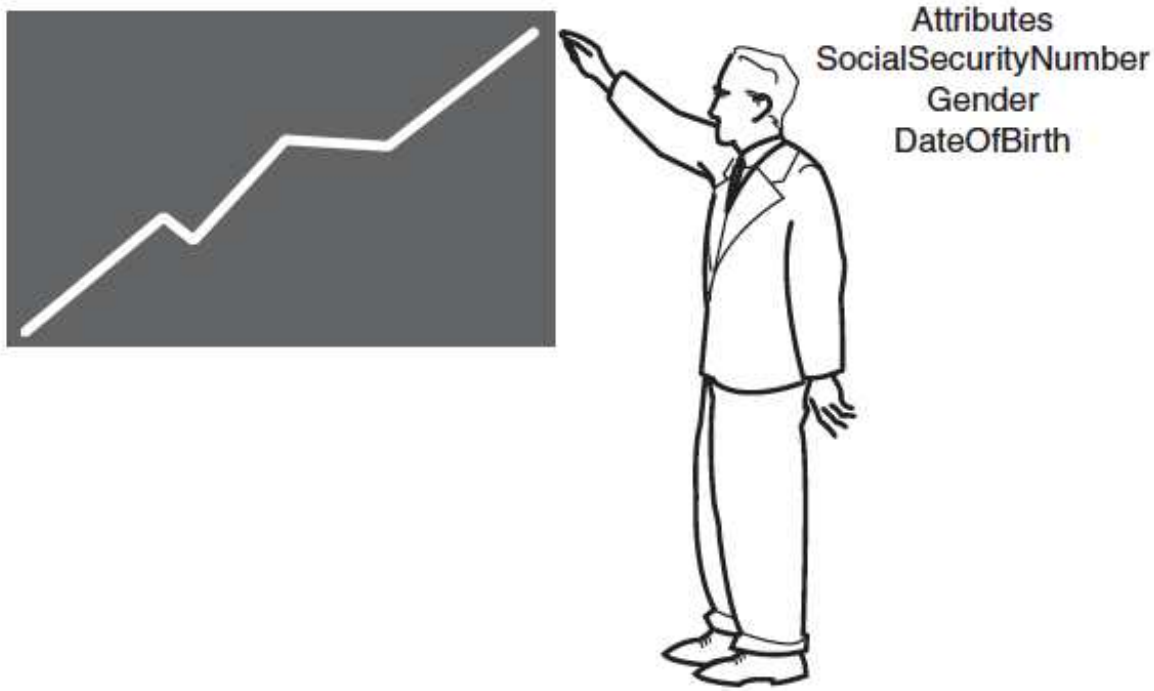


Figure 1.6 Employee attributes.

The data stored within an object represents the *state* of the object.

– In O-O programming terminology, this data is called *attributes*.

Object Behaviors

The *behavior* of an object is what the object can do.

- In procedural languages the behavior is defined by procedures, functions, and subroutines.

Object Behaviors

In O-O programming terminology these behaviors are contained in *methods,* and you invoke a method by sending a *message* to it.

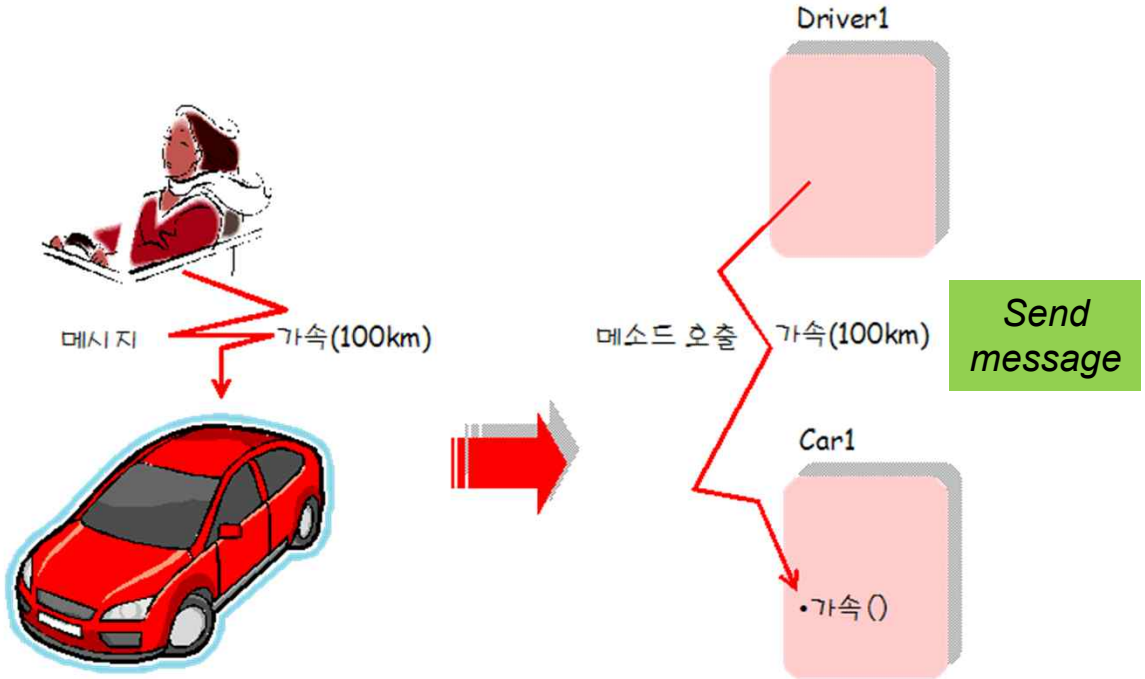


그림 7.5 메시지 전달

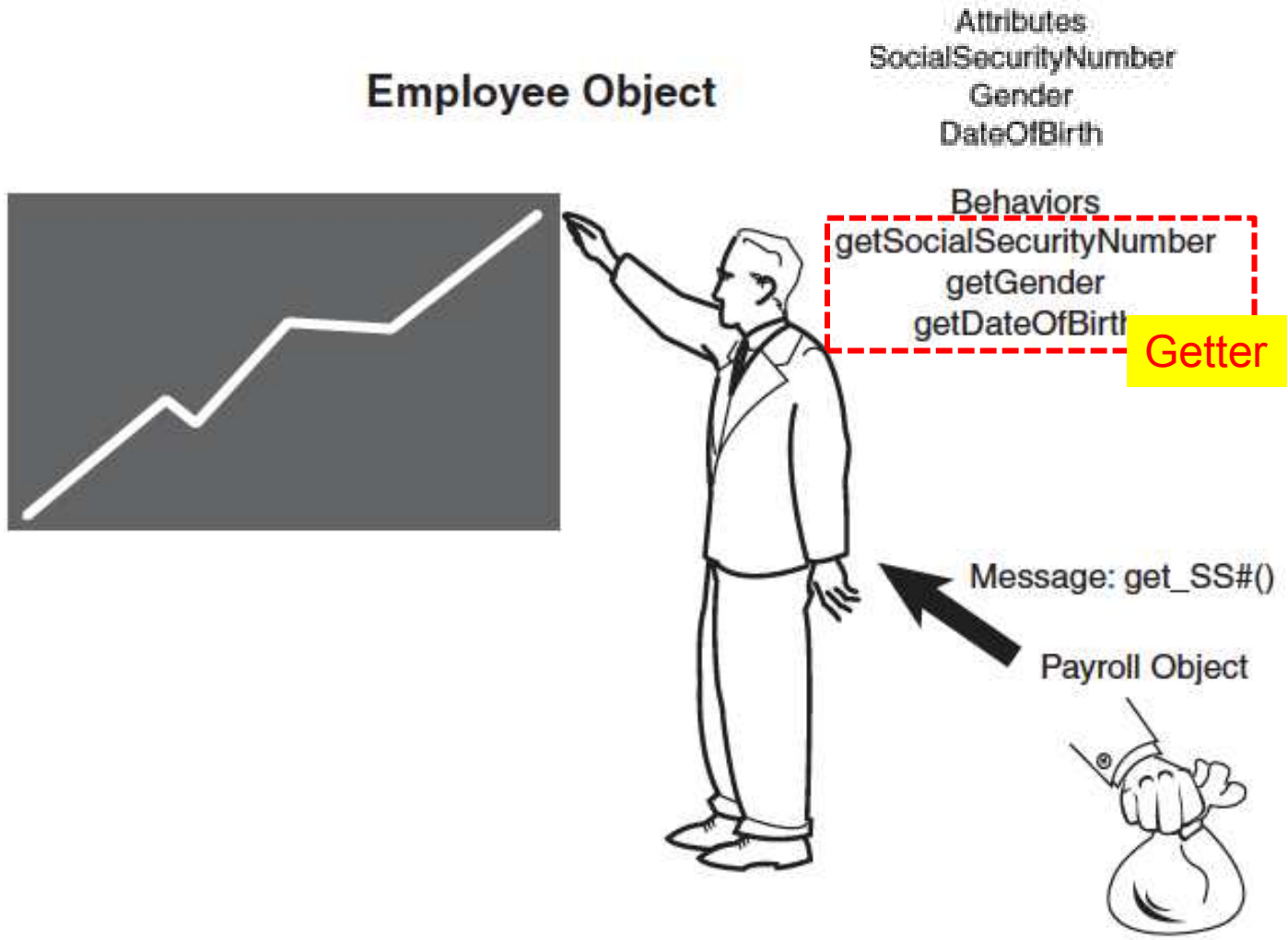


Figure 1.7 Employee behaviors.

Getter & Setter method

For example, consider an attribute called Name, using Java, that looks like the following:

```
private String name;
```

The corresponding getter and setter would look like this:

```
public void setName (String n) {name = n;};
public String getName() {return name;};
```

What Exactly Is a Class?

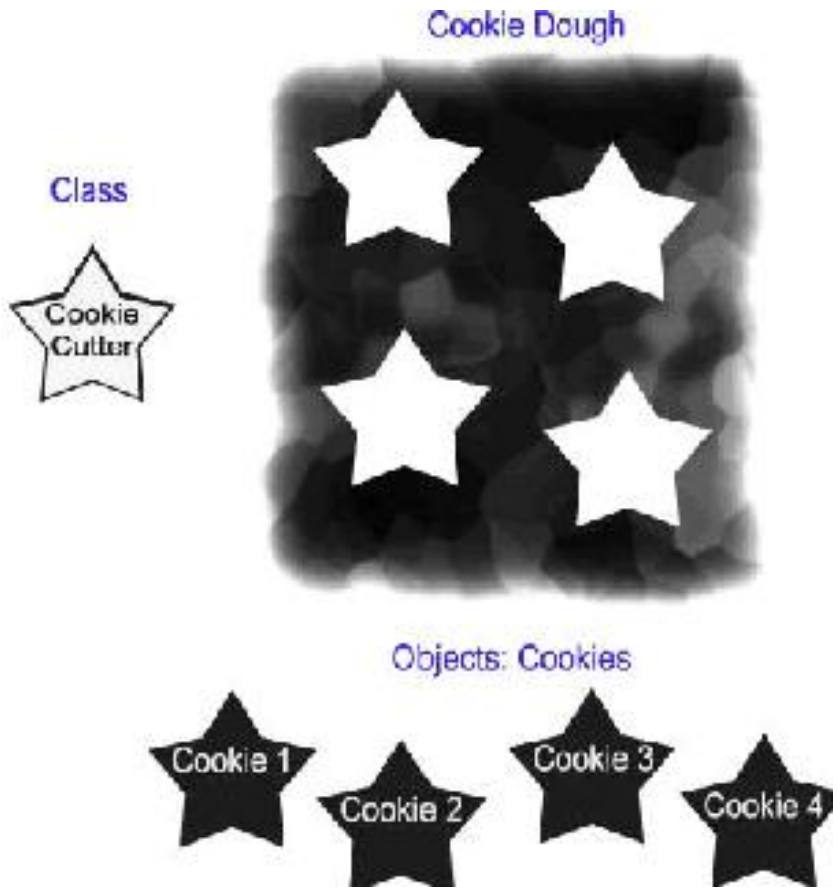
A *class* is a blueprint for an object.

- When you instantiate an object, you use a class as the basis for how the object is built.

What Exactly Is a Class?

An object cannot be instantiated without a class.

- Classes can be thought of as the templates, or cookie cutters, for objects



```
public class Person{
  //Attributes
  private String name;
  private String address;

  //Methods
  public String getName(){
    return name;
  }
  public void setName(String n){
    name = n;
  }
  public String getAddress(){
    return address;
  }
  public void setAddress(
        String adr){
    address = adr;
  }
}
```

Higher Level Data Types?

A class can be thought of as a sort of higher-level data type.

- For example, just as you create an integer or a float:

```
int x;
float y;
Person p1;
```

Attributes

- The data of a class is represented by attributes.
- Each class must define the attributes that will store the state of each object instantiated from that class.
- In the Person class example in the previous section, the Person class defines attributes for name and address.

Methods

- Methods implement the required behavior of a class.
- Methods may implement behaviors that are called from other objects (messages) or provide the fundamental, internal behavior (private) of the class.
- In the Person class, the behaviors are getName(), setName(), getAddress(), and setAddress().

Messages

- Messages are the communication mechanism between objects.
  - For example, when Object A invokes a method of Object B, Object A is sending a message to Object B.
  - Object B's response is defined by its return value.
- Only the public methods, not the private methods, of an object can be invoked by another object.

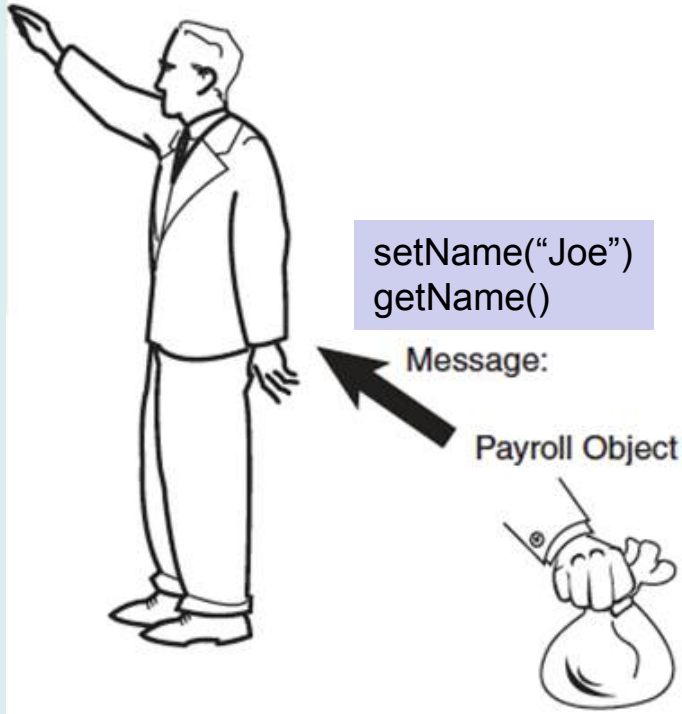
Method Signatures

The following is all the user needs to know to effectively use the methods:

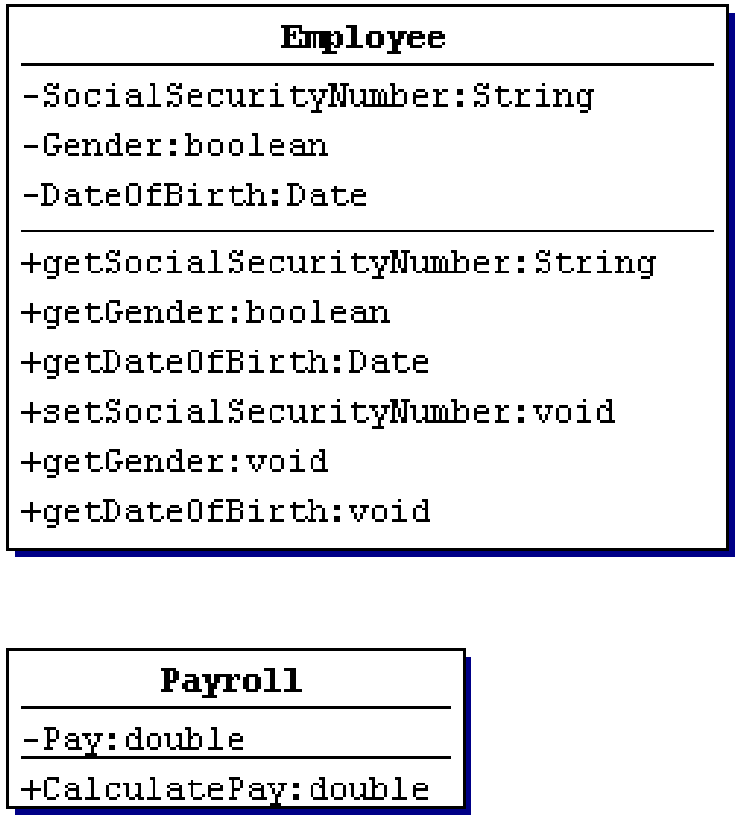
1. The name of the method
2. The parameters passed to the method
3. The return type of the method

- E.g.,

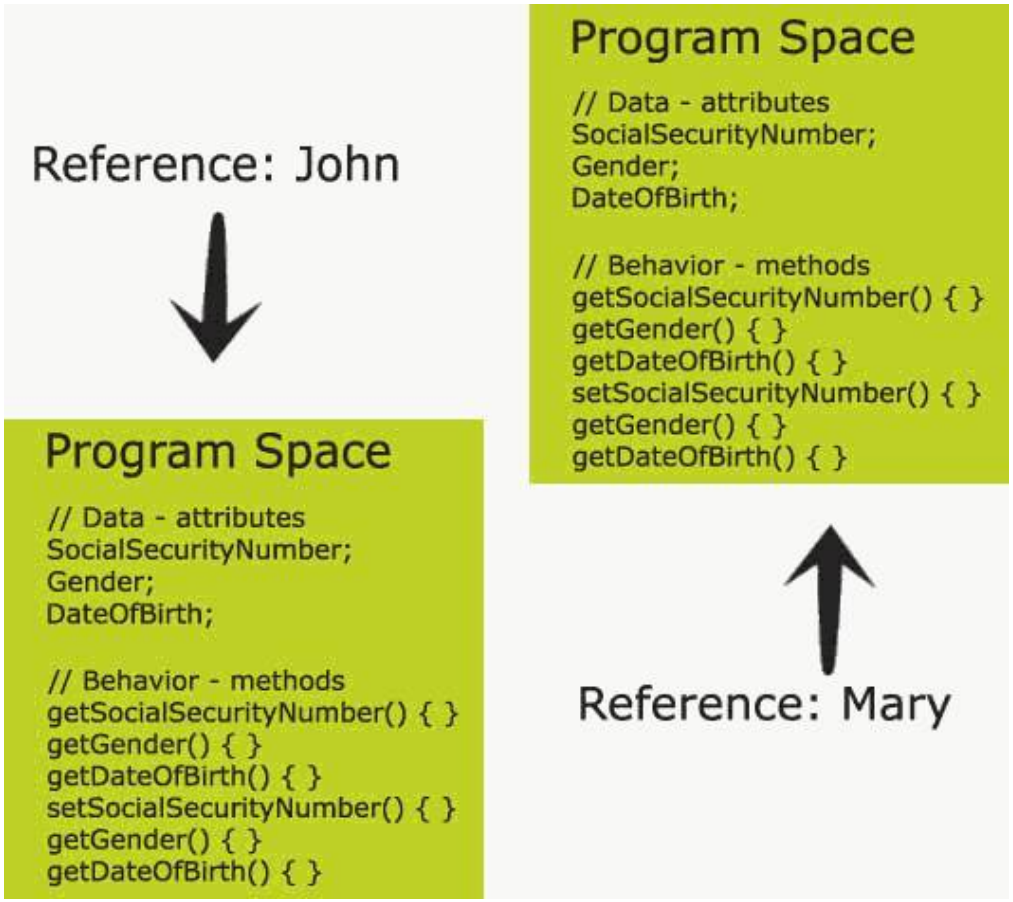
```
public class Payroll{
  public static void main(String[] args) {
          String name;
          Person p = new Person();
          p.setName("Joe");
           ... Other code
          name = p.getName();
  }
}
```



Modeling a Class in UML



Program Spaces




```
Employee John = new Employee();
Employee Mary = new Employee();
```

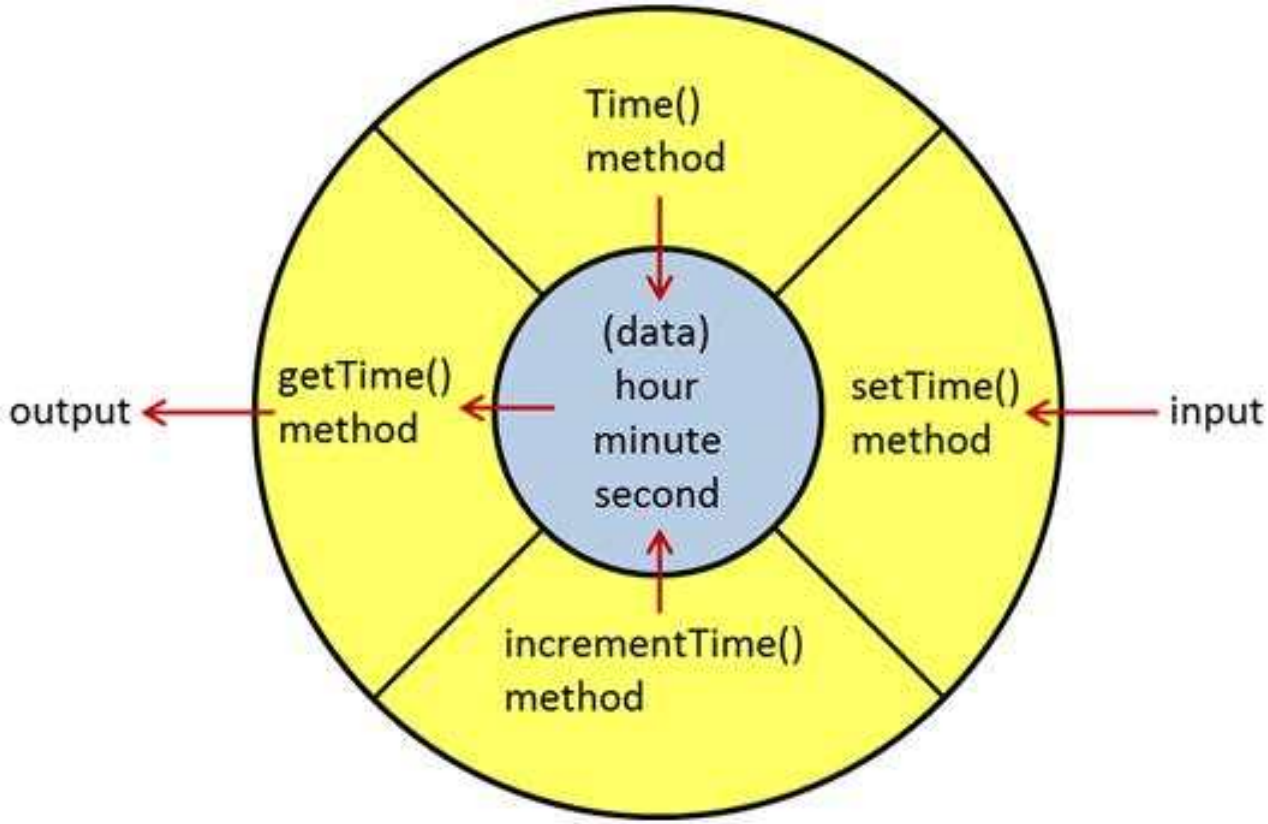
O-O Language Criteria

- Encapsulation - 캡슐화
- Inheritance - 상속
- Polymorphism - 다형성
- Composition - 조합

3대 특징

Encapsulation and Data Hiding

- 캡슐화와 데이터 은닉



Encapsulation

One of the primary advantages of using objects is that the object need not reveal all its attributes and behaviors.

- In good O-O design (at least what is generally accepted as good), an object should only reveal the interfaces needed to interact with it.

Encapsulation

Details not pertinent to the use of the object should be hidden from other objects.

- This is called *encapsulation*.

Interfaces

Any behavior that the object provides must be invoked by a message sent using one of the provided interfaces.

- The interface should completely describe how users of the class interact with the class.
- methods that are part of the interface are designated as public.

Interfaces

Interfaces do not normally include attributes only methods.

- As discussed earlier in this module, if a user needs access to an attribute, then a method is created to return the attribute.

Implementations

Only the public attributes and methods are considered the interface.

- The user should not see any part of the implementation interacting with an object solely through interfaces.
- In previous example, for instance the Employee class, only the attributes were hidden.
- Thus, the implementation can change and it will not affect the user's code.

A Real-World Example of the Interface/Implementation Paradigm

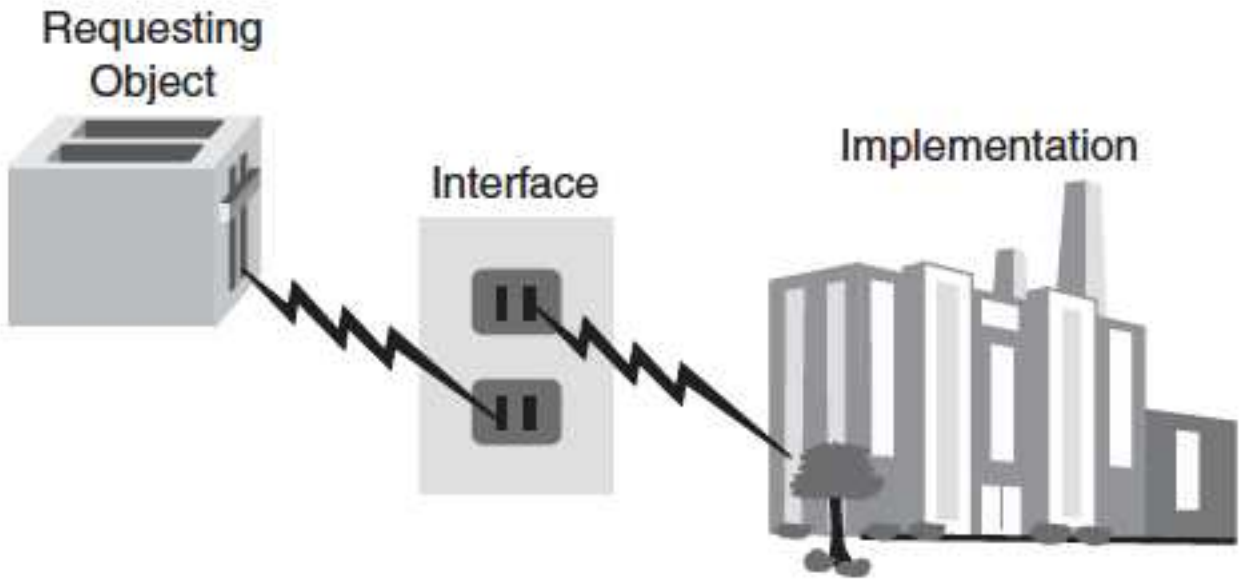


Figure 1.12 Power plant example.

A Model of the Interface/Implementation Paradigm

- Note that in the class diagram, the plus sign (+) designates public and the minus sign (-) designates private.

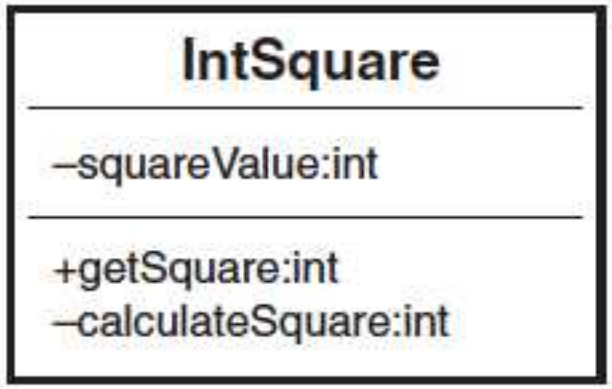


Figure 1.13 The square class.

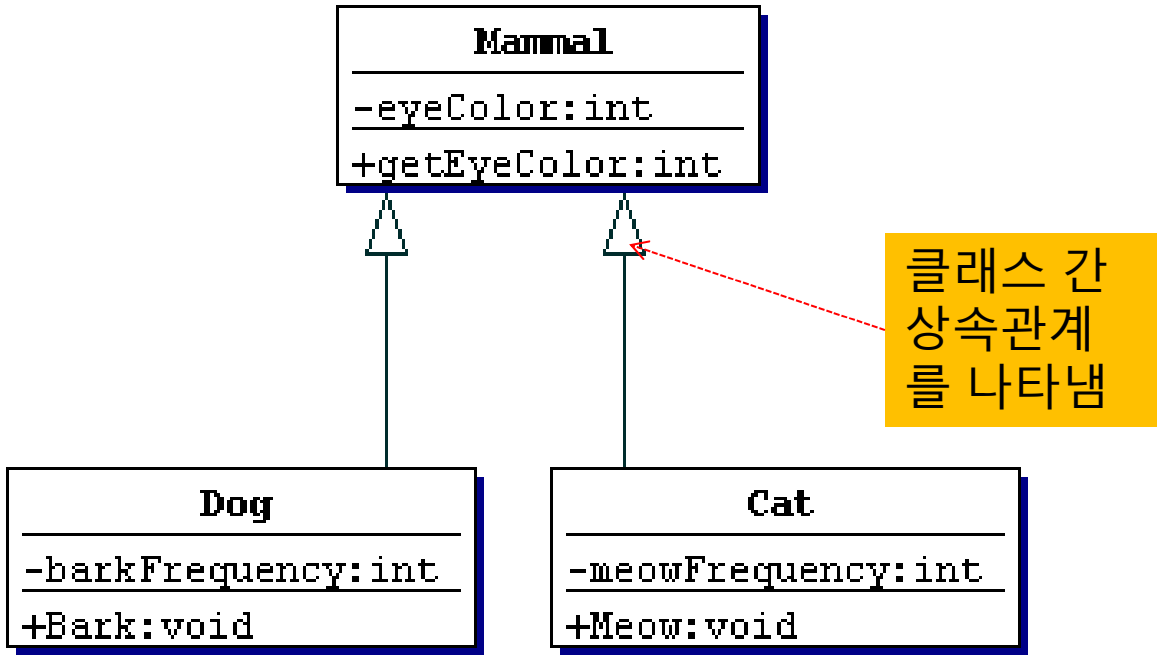
```
public class IntSquare {
  // private attribute
  private int squareValue;

  // public interface
  public int getSquare (int value) {
    squareValue =calculateSquare(value);
    return squareValue;
  }

  // private implementation
  private int calculateSquare (int value) {
    return value*value;
  }
}
```

Inheritance

Inheritance allows a class to inherit the attributes and methods of another class.

- This allows you to create brand new classes by abstracting out common attributes and behaviors.

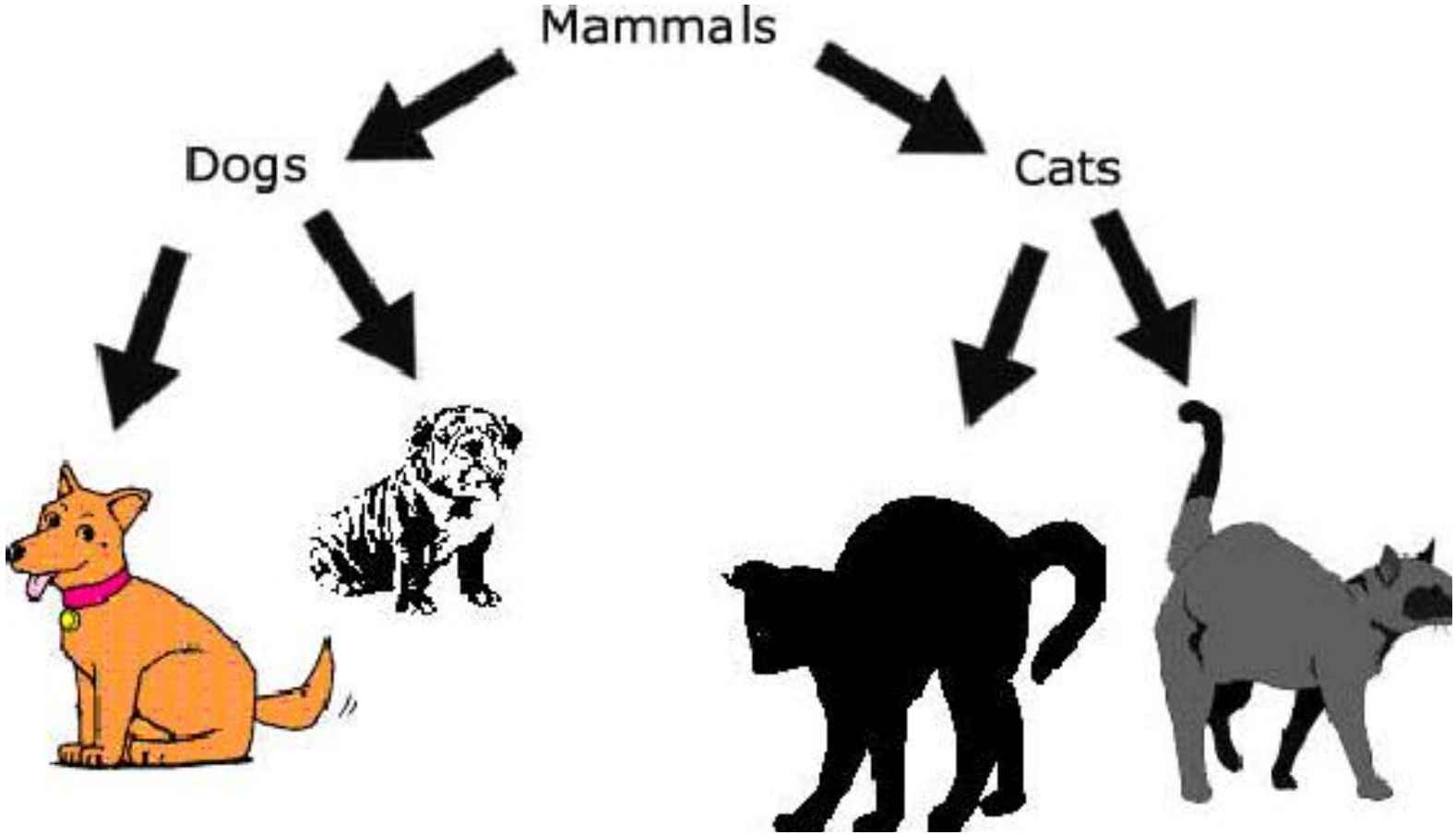
Mammal Hierarchy



Superclasses and Subclasses

- The superclass, or parent class, contains all the attributes and behaviors that are common to classes that inherit from it.

Mammal Hierarchy



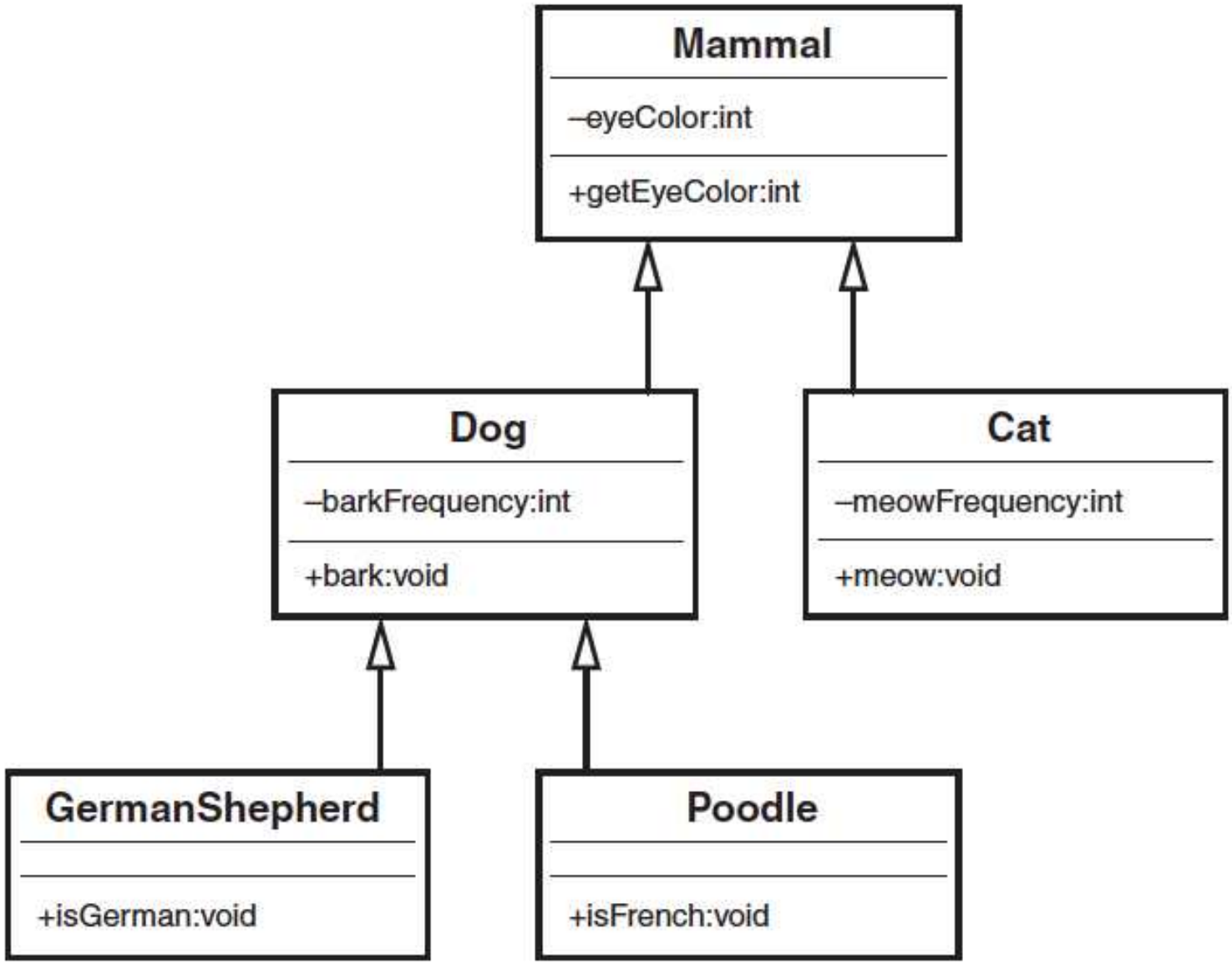


Figure 1.15 Mammal UML diagram.

Is-a Relationships

In the Shape example, Circle, Square, and Star all inherit directly from Shape.

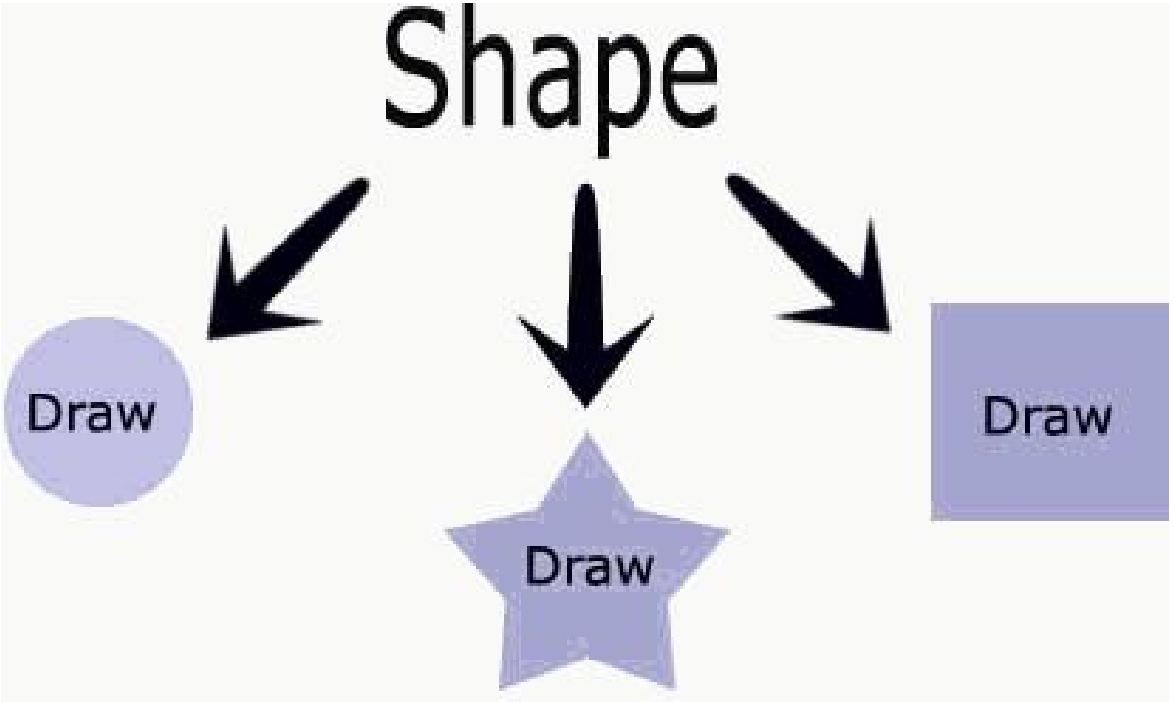
- This relationship is often referred to as an *is-a relationship* because a circle *is a* shape.

수퍼 클래스	서브 클래스
Animal	Lion, Dog,
Vehicle	Car, Bus, Truck, Boat, Motorcycle, Bicycle
Shape	Rectangle, Triangle, Circle

IS-A 관계

Polymorphism

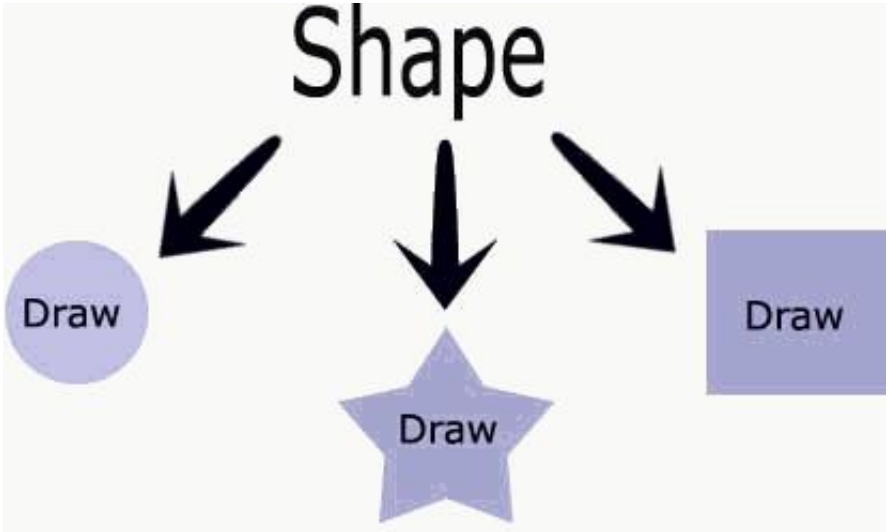
Polymorphism literally means many shapes.



Polymorphism

In short, each class is able to respond differently to the same Draw method and draw itself.

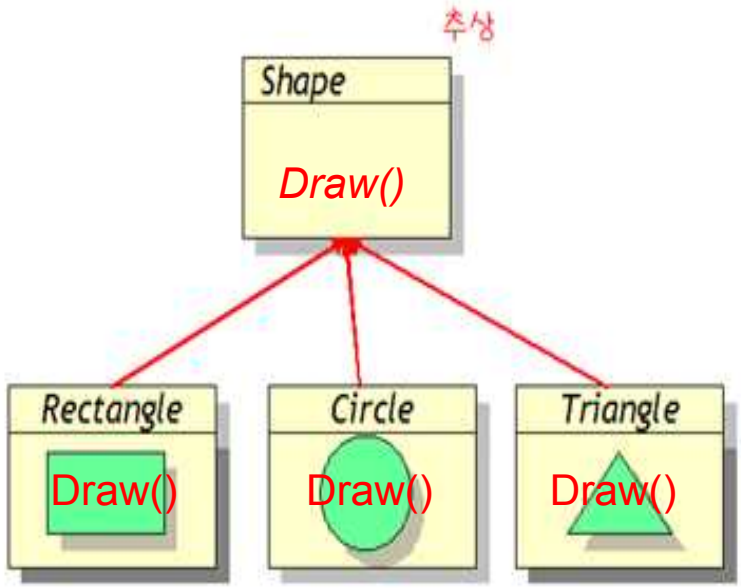
– This is what is meant by polymorphism.



Polymorphism

The Shape object cannot draw a shape, it is too abstract (in fact, the Draw() method in Shape contains no implementation).

- You must specify a concrete shape.
- To do this, you provide the actual implementation in Circle.



Polymorphism

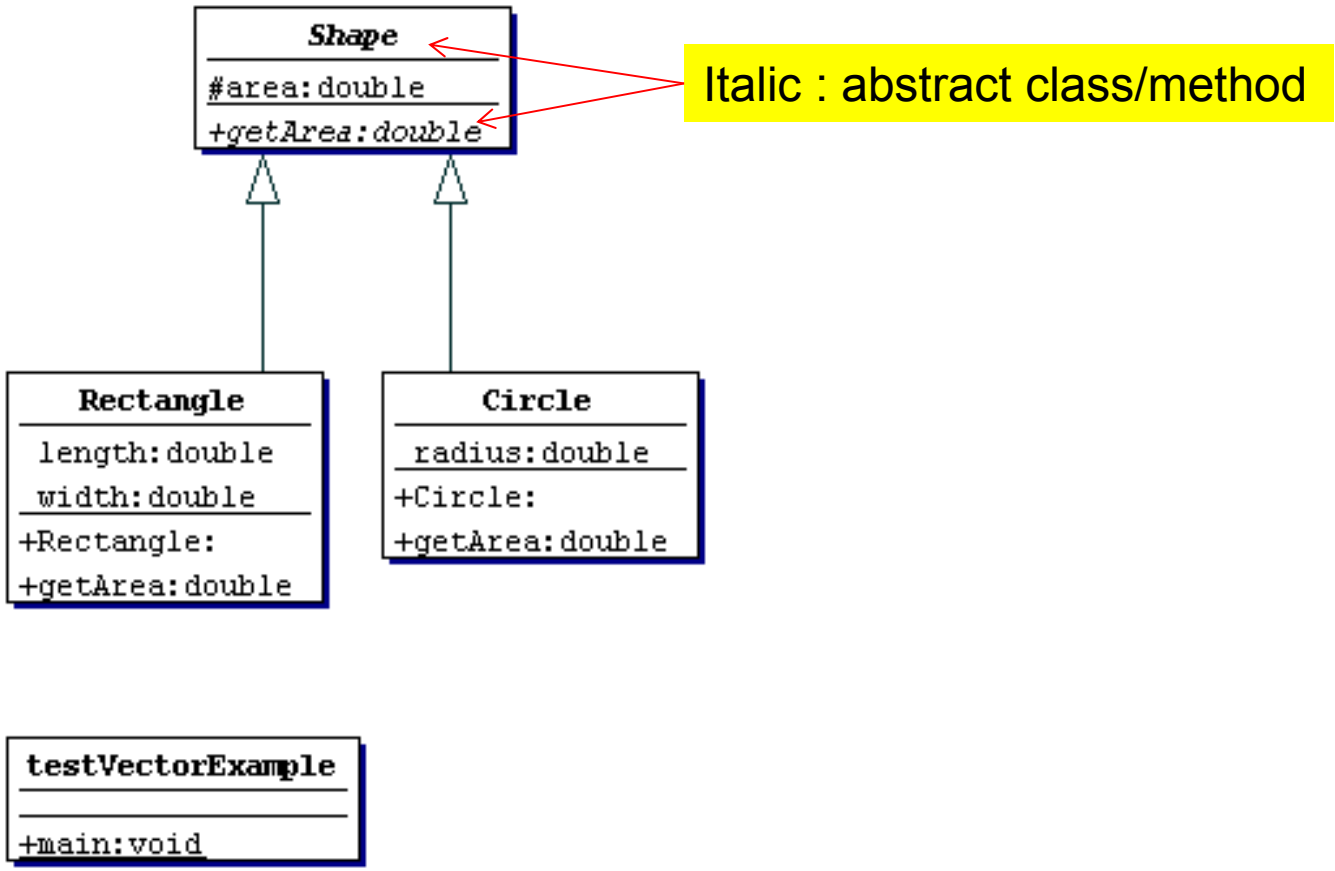
When a method is defined as abstract, a subclass must provide the implementation for this method.

- In this case, Shape is requiring subclasses to provide a Draw() implementation.

Polymorphism

- If a subclass inherits an abstract method from a superclass, it *must* provide a concrete implementation of that method, or else it will be an abstract class itself (see the following figure for a UML diagram).

Shape UML Diagram



Class Shape and Subclasses

```
public abstract class Shape{
  protected double area;
  public abstract double getArea();
}

public class Circle extends Shape{
  double radius;
  public Circle(double r) {
    radius = r;
  }
  public double getArea() {
    area = 3.14*(radius*radius);
    return (area);
  }
}
```

```
public class Rectangle extends Shape{
  double length;
  double width;
  public Rectangle(double l, double w){
    length = l;
    width = w;
  }
  public double getArea() {
    area = length*width;
    return (area);
  }
}
```

```
// testVectorExample.main() code fragment
Shape s = new Circle(5.0);
System.out.println(s.getArea());
s = new Rectangle(2.0, 5.0);
System.out.println(s.getArea());
```

- The Shape class has an attribute called area that holds the value for the area of the shape.
- The method getArea() includes an identifier called abstract.
- When a method is defined as abstract, a subclass must provide the implementation for this method

Composition

It is natural to think of objects as containing other objects.

- A television set contains a tuner and video display.

Composition

A computer contains video cards, keyboards, and drives.

- Although the computer can be considered an object unto itself, the drive is also considered a valid object.

Composition

In fact, you could open up the computer and remove the drive and hold it in your hand.

- Both the computer and the drive are considered objects. It is just that the computer contains other objects such as drives.
- In this way, objects are often built, or *composed*, from other objects: This is *composition*.

Has-a Relationships

While an inheritance relationship is considered an Is-a relationship for reasons already discussed, a composition relationship is termed a Has-a relationship.

Has-a Relationships

- Using the example in the previous section, a television Has-a a tuner and Has-a video display.
- A television is obviously not a tuner, so there is no inheritance relationship.

Has-a Relationships

In the same vein, a computer Has-a video card, Has-a keyboard, and Has-a disk drive.

- The topics of inheritance, composition and how they relate to each other is covered in great detail later in the course.

Conclusion

- Encapsulation (캡슐화)
  - Encapsulating the data and behavior into a single object is of primary importance in OO development.
  - A single object contains both its data and behaviors and can hide what it wants from other objects.

- Polymorphism (다형성)
  - Polymorphism means that similar objects can respond to the same message in different ways.
  - Using polymorphism, you can send each of these shapes the same message (for example, Draw).

- Inheritance (상속)
  - A class can inherit from another class and take advantage of the attributes and methods defined by the superclass.
- Composition (조합)
  - Composition means that an object is built from other objects.