Object Oriented Programming

Topics To Be Covered Today

- Objects versus Class
- Three main concepts of OOP
  - Encapsulation
  - Inheritance
  - Polymorphism
- Method
  - Parameterized
  - Value-Returning

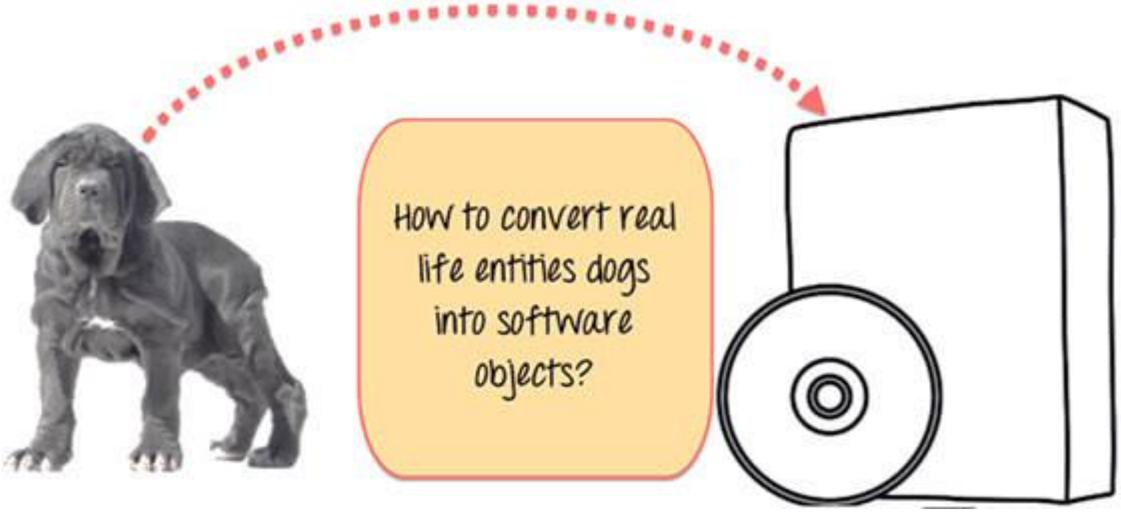
Shift Operator

A class can contain any of the following variable types.

- Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- Instance variables: Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

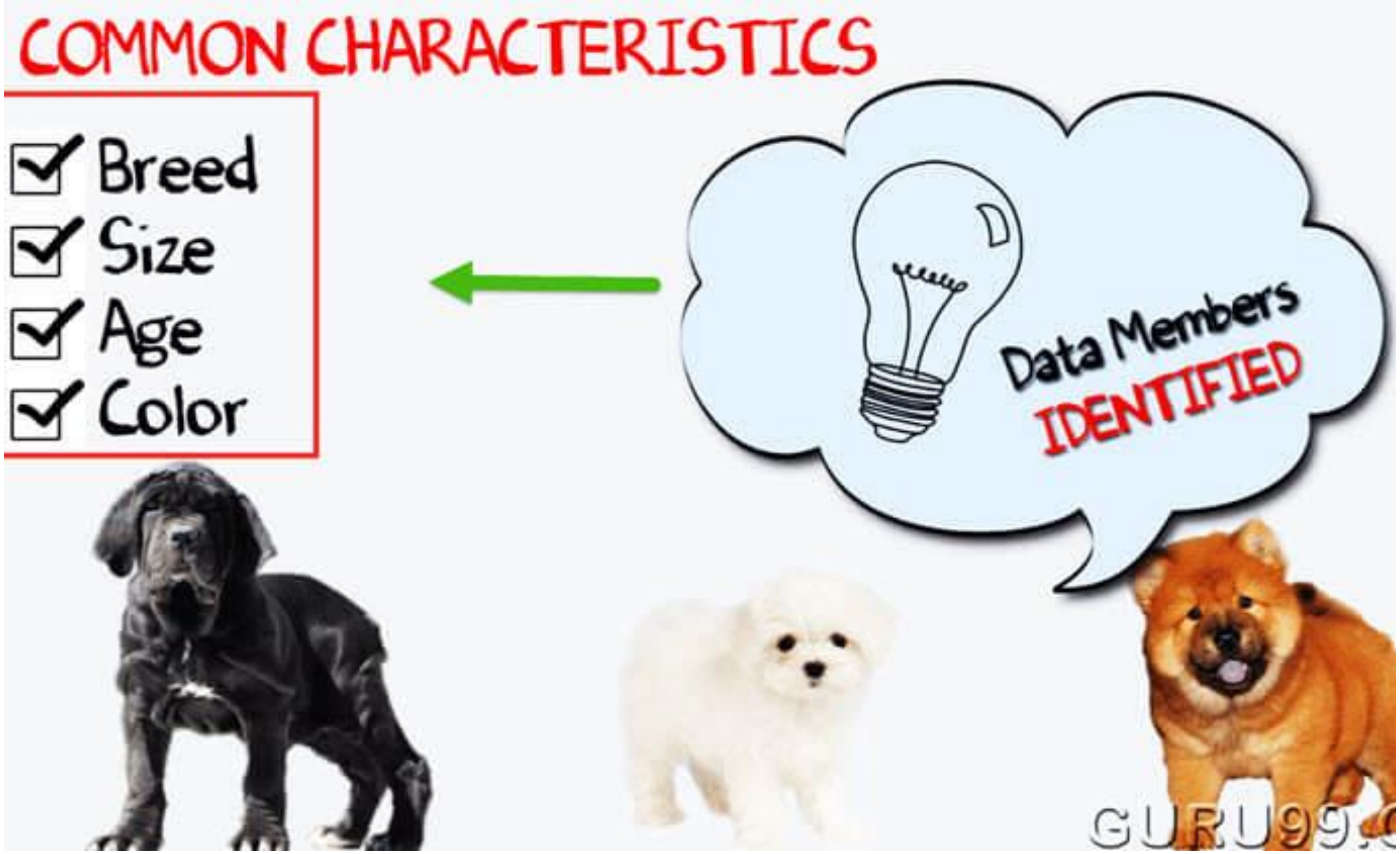
- Class variables: Class variables are variables declared within a class, outside any method, with the static keyword.

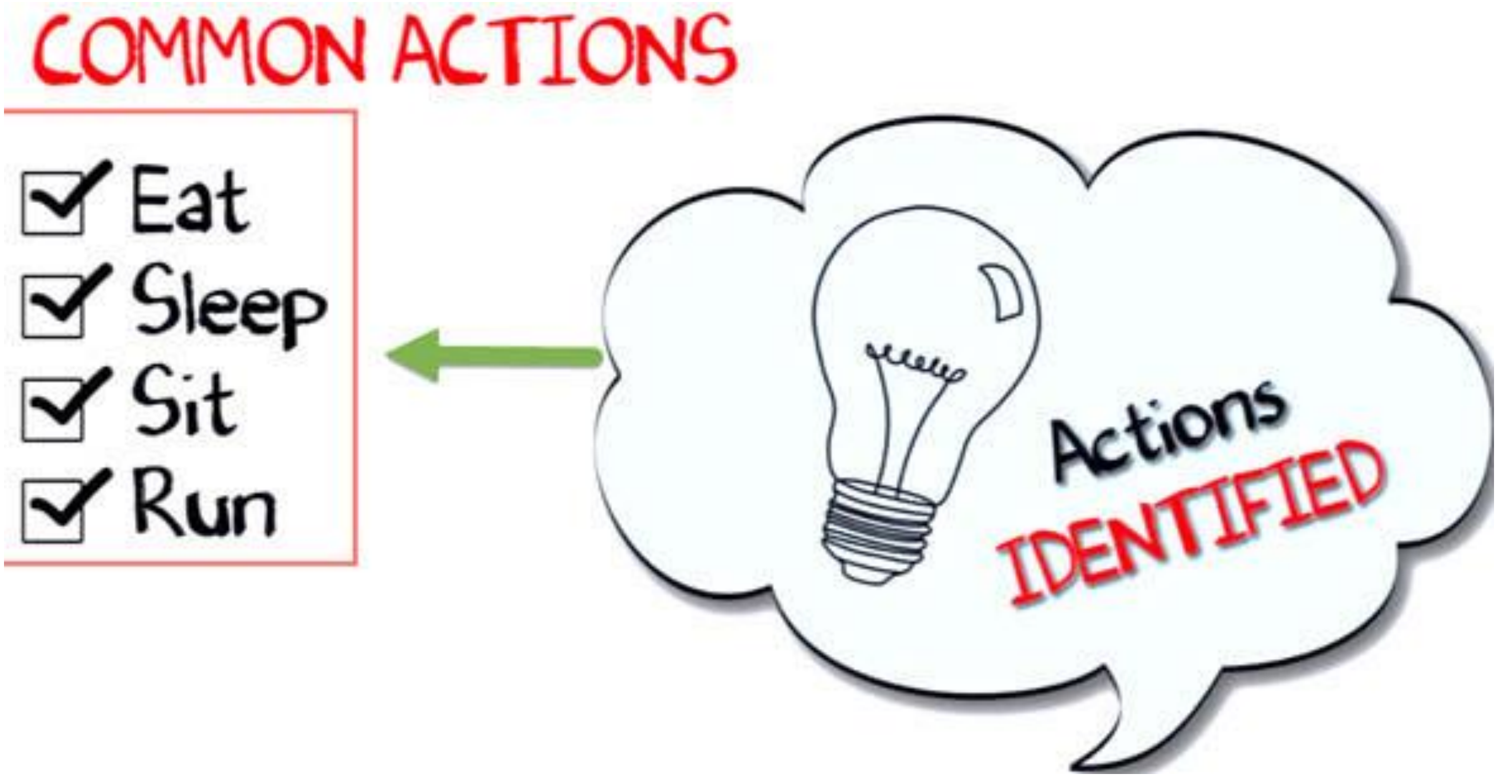
- Let's take an example of developing a pet management system, specially meant for dogs. You will need various information about the dogs like different breeds of the dogs, the age, size, etc.



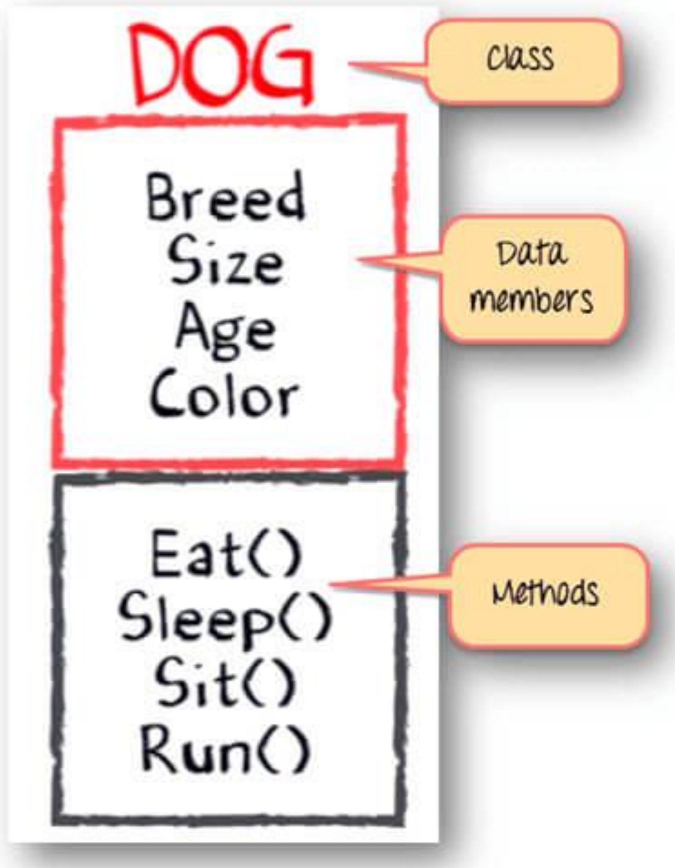
How to convert real life entities dogs into software objects?

It has differences

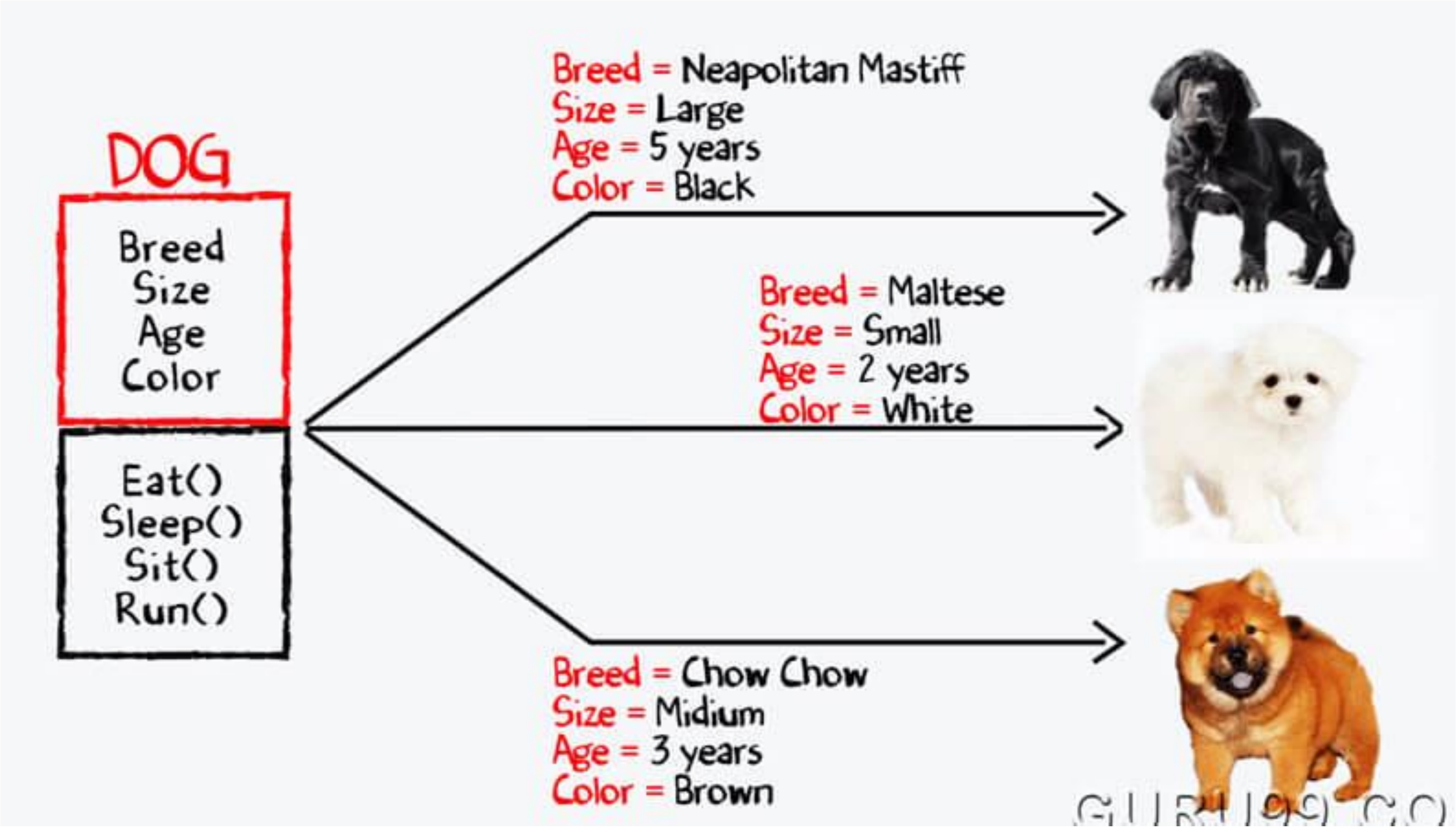
COMMON CHARACTERISTICS
☑ Breed
☑ Size
☑ Age
☑ Color
Data Members
IDENTIFIED
GURU99.C


COMMON ACTIONS
☑ Eat
☑ Sleep
☑ Sit
☑ Run
Actions
IDENTIFIED

Class template



objects


DOG
Breed
Size
Age
Color
Eat()
Sleep()
Sit()
Run()
Breed = Neapolitan Mastiff
Size = Large
Age = 5 years
Color = Black
Breed = Maltese
Size = Small
Age = 2 years
Color = White
Breed = Chow Chow
Size = Midium
Age = 3 years
Color = Brown
GURU99.CO

What is an Object

- Object: is a bundle of data and its behavior(often known as methods).
- Objects have two characteristics: They have states and behaviors.
- Examples of states and behaviors

Example 1:

Object: House

State: Address, Color, Area

Behavior: Open door, close door

- So if I had to write a class based on states and behaviours of House. I can do it like this: States can be represented as instance variables and behaviours as methods of the class. We will see how to create classes in the next section of this guide.

```
class House {
   String address;
   String color;
   double are;
   void openDoor() {
}
   void closeDoor() {
}
}
```

- **Example 2:**
  Let’s take another example.
  Object: Car
  State: Color, Brand, Weight, Model
  Behavior: Break, Accelerate, Slow Down, Gear change.
- **Note:** As we have seen above, the states and behaviors of an object, can be represented by variables and methods in the class respectively.

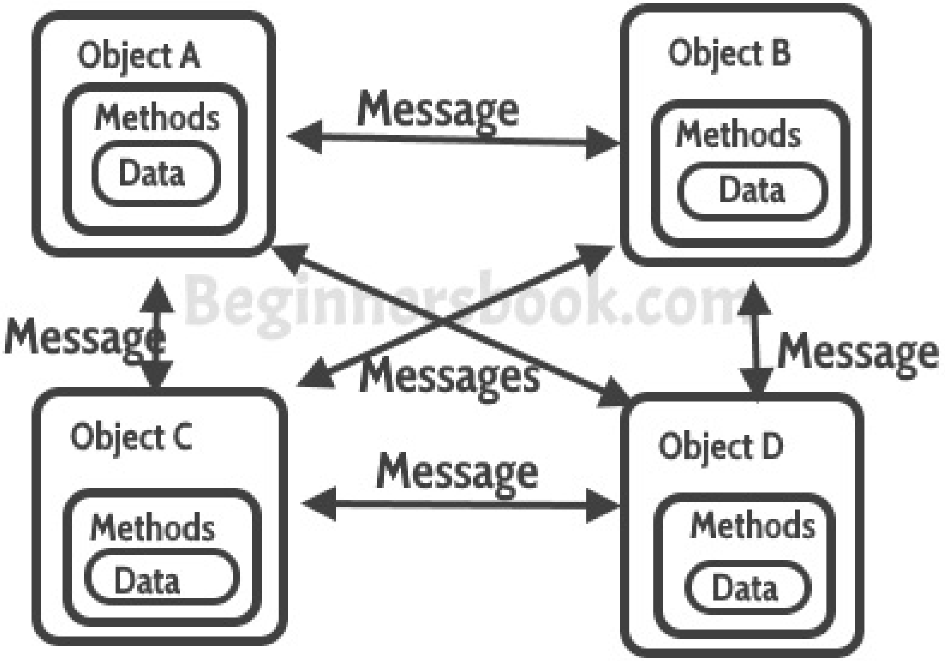
What is an Object?

- Real world objects are things that have:
  - state
  - Behavior
- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.
- Example: your dog:
  - state – name, color
  - behavior – sitting, barking, waging tail, running
- A software object is a bundle of variables (state) and methods (operations).

- For Example, Pen is an object.
- Its name is Reynolds; color is white, known as its state.
- It is used to write, so writing is its behavior.

Message passing

A single object by itself may not be very useful. An application contains many objects. One object interacts with another object by invoking methods on that object. It is also referred to as **Method Invocation**. See the diagram below.



CLASS??

- A class is an entity that determines how an object will behave and what the object will contain.
- So

Difference between object and class

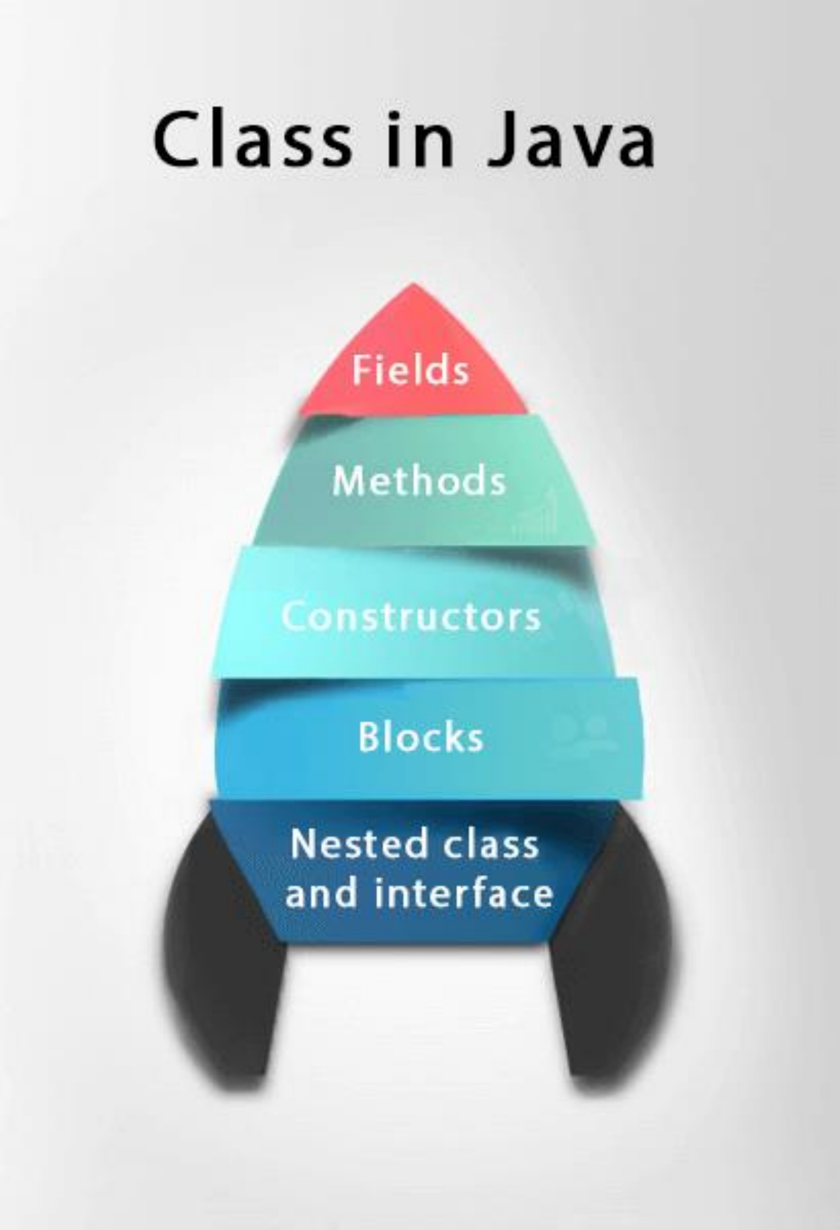
- A **class** is a **blueprint or prototype** that defines the variables and the methods (functions) common to all objects of a certain kind.

- An **object** is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life.

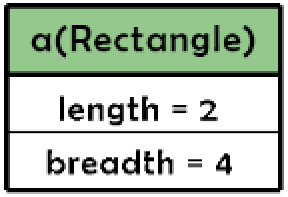
What is a Class

- **An object is an instance of a class.**
- A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.
- A class is a blueprint that defines the variables and methods common to all objects of a certain kind.
- Example: ‘your dog’ is a object of the class Dog.
- An object holds values for the variables defines in the class.
- An object is called

  an instance of the Class


Class in Java
Fields
Methods
Constructors
Blocks
Nested class and interface

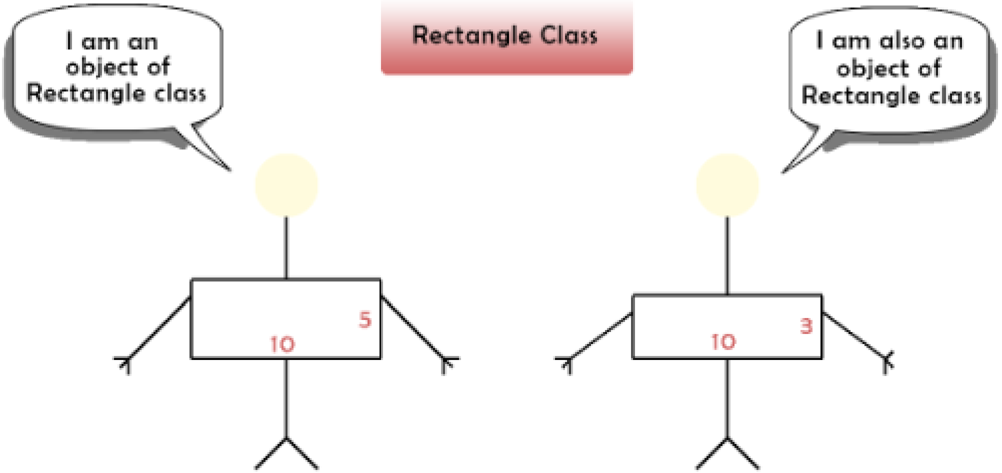
Mention state class object behavior for diagram



a(Rectangle)

length = 2

breadth = 4


I am an object of Rectangle class
Rectangle Class
I am also an object of Rectangle class
5
10
3
10

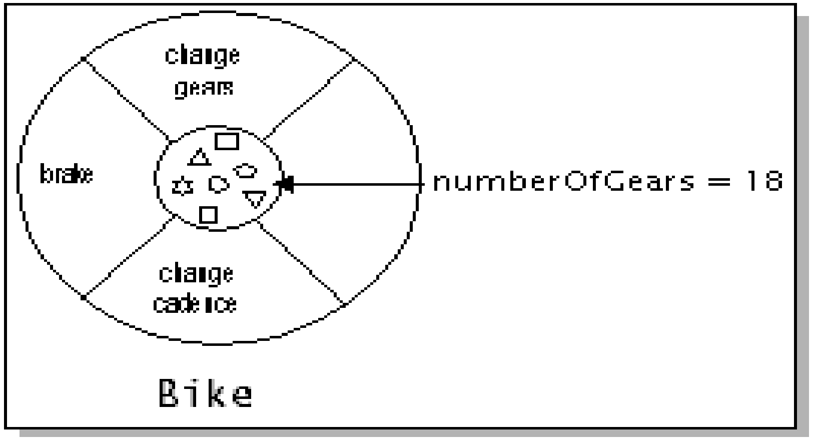
new keyword in Java

- The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

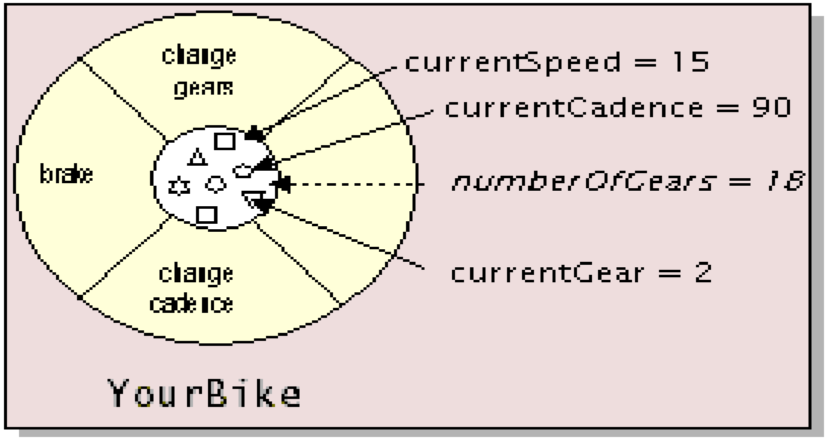
```
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

```java
//Java Program to demonstrate having the main method in
//another class
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Objects versus Class



Class

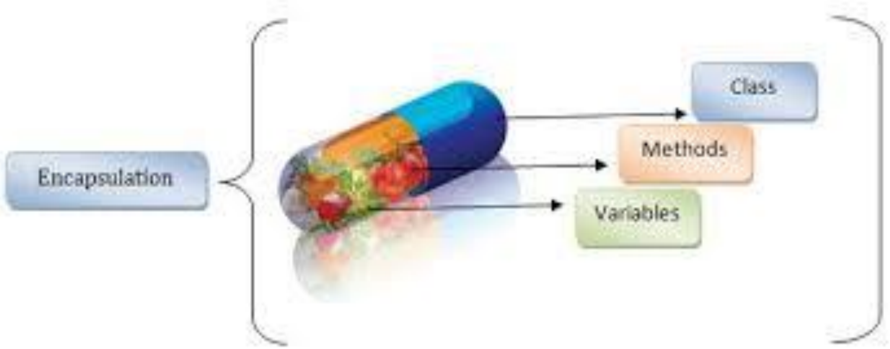


Instance of a Class

- operations: change Gears, brake
- variables: current Speed, current Gear

Three main concepts

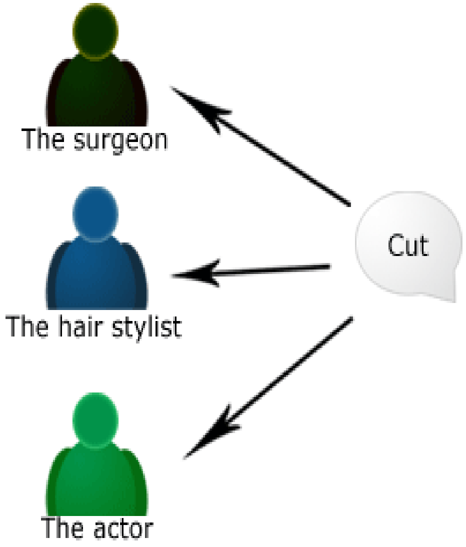
Encapsulation



Inheritance

Polymorphism

iF ANY BODY SAYS "CUT" TO THESE PEOPLE

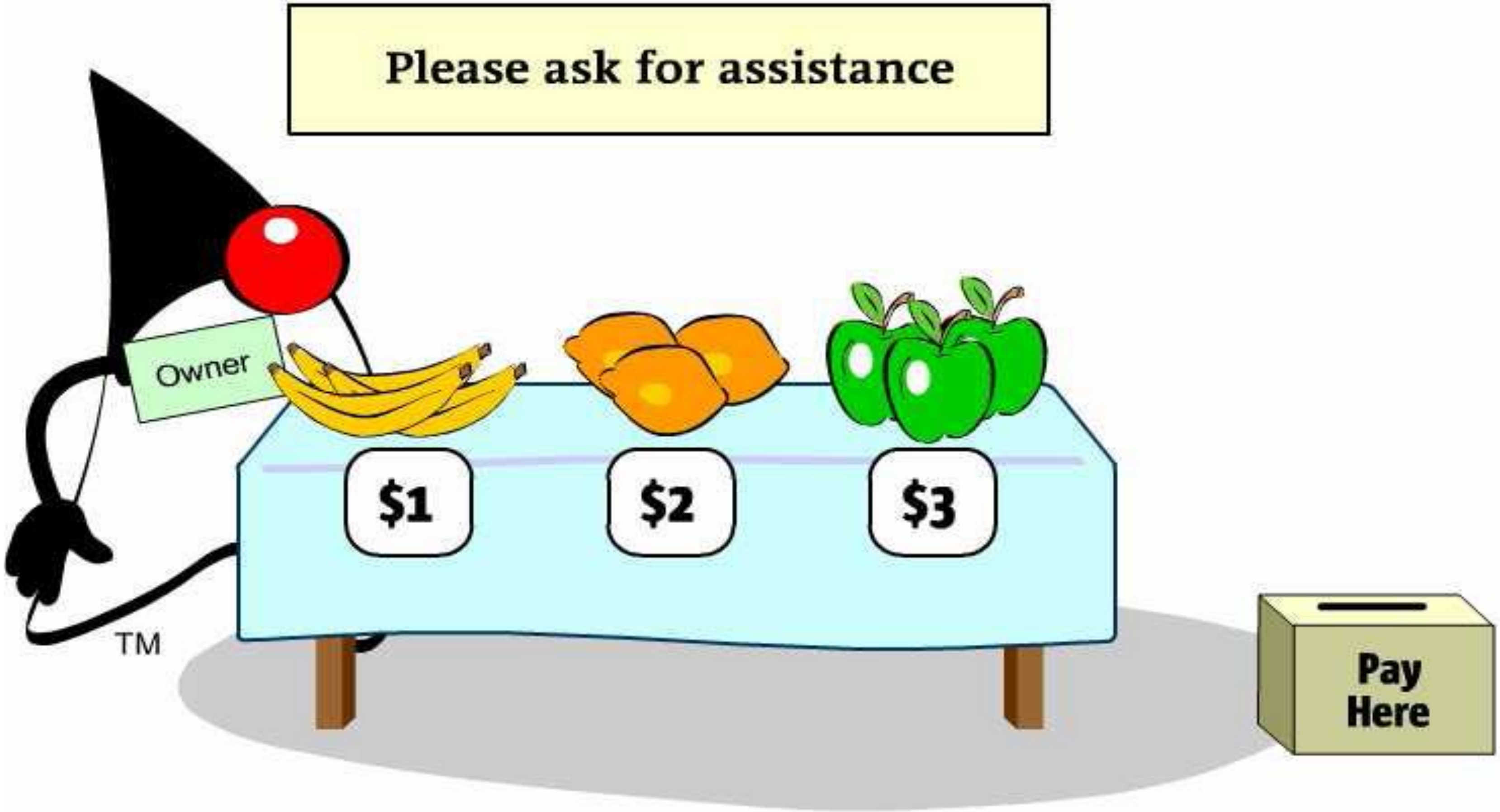


The surgeon would begin to make an incision.

The hair stylist would begin to cut someone's hair.

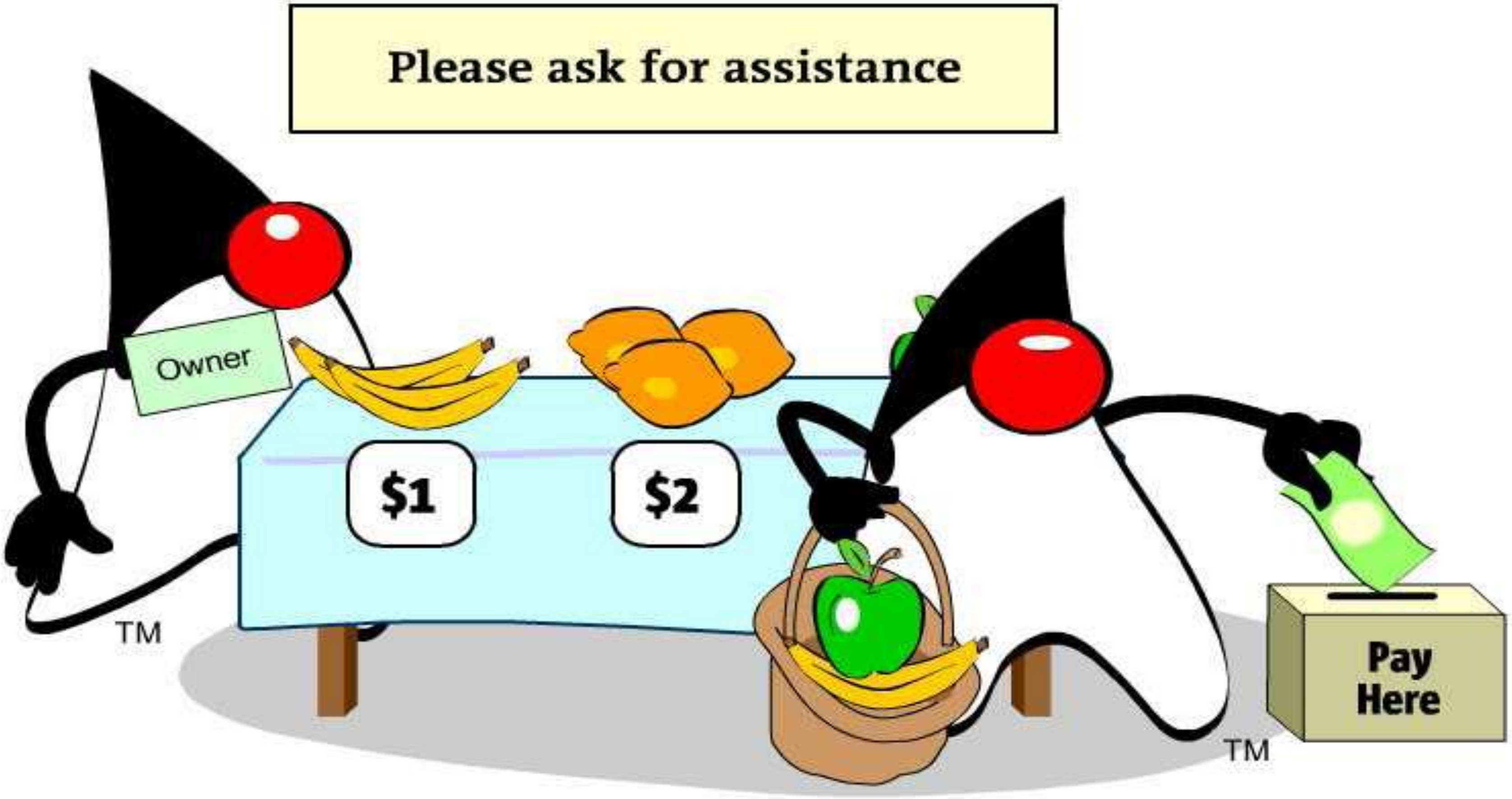
The actor would abruptly stop acting out the current scene, awaiting directorial guidance.

Encapsulation



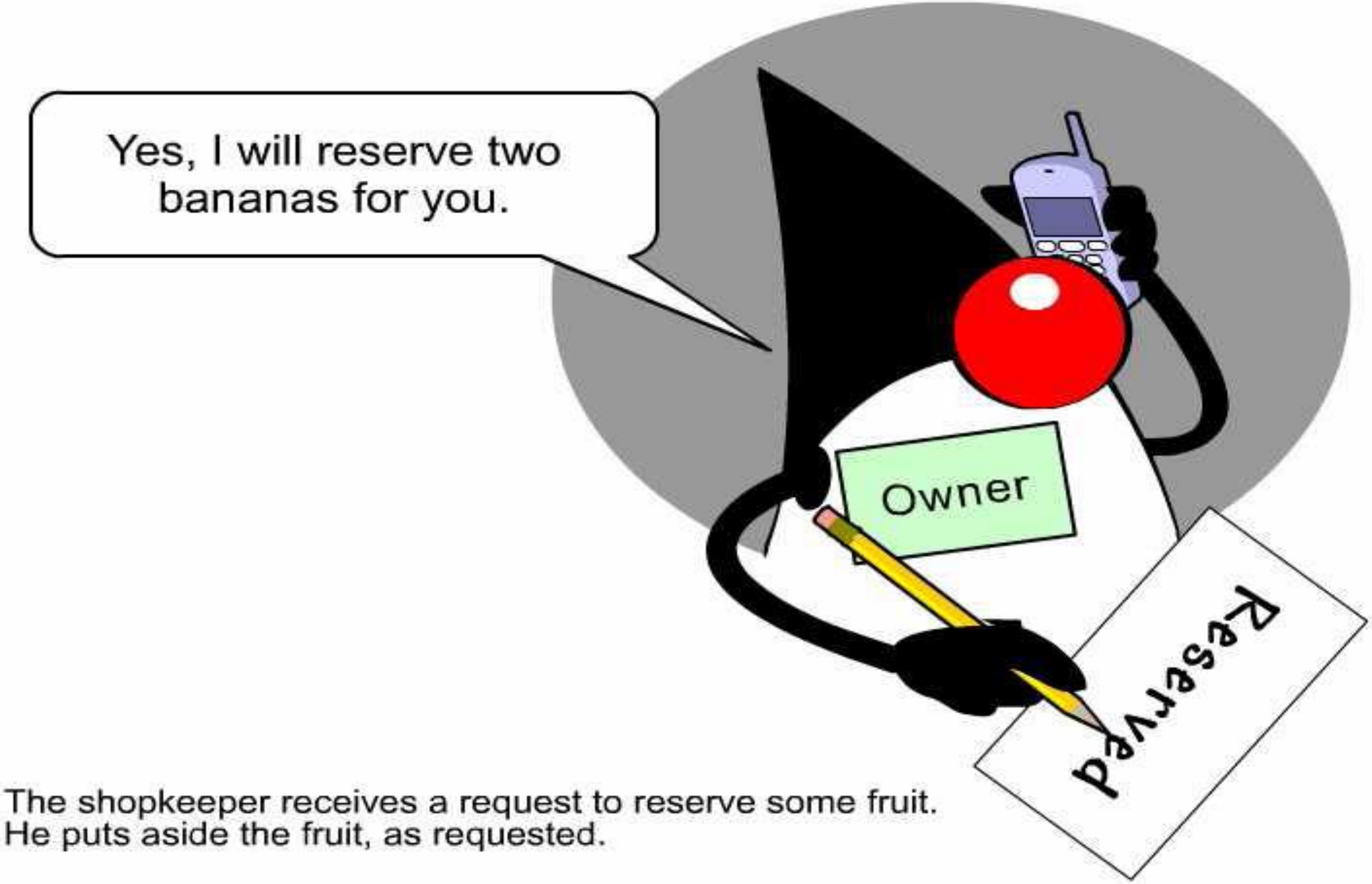
The shopkeeper's fruit stand is designed to serve all customers. However, the design of the fruit stand does not prevent self-service. The fruit is freely accessible to all customers.

Encapsulation



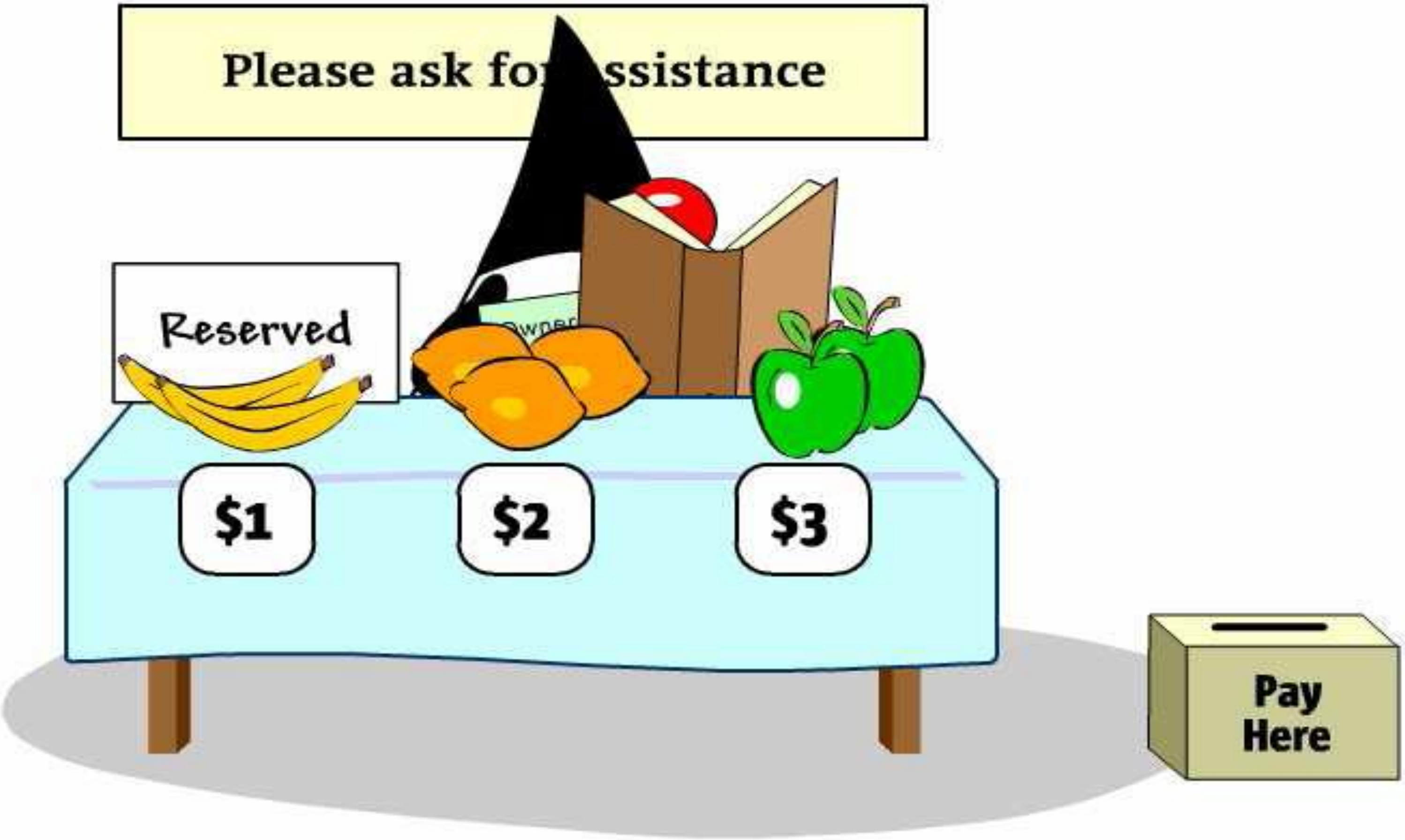
A customer can select some fruit and make payment, all without the shopkeeper's involvement.

Encapsulation



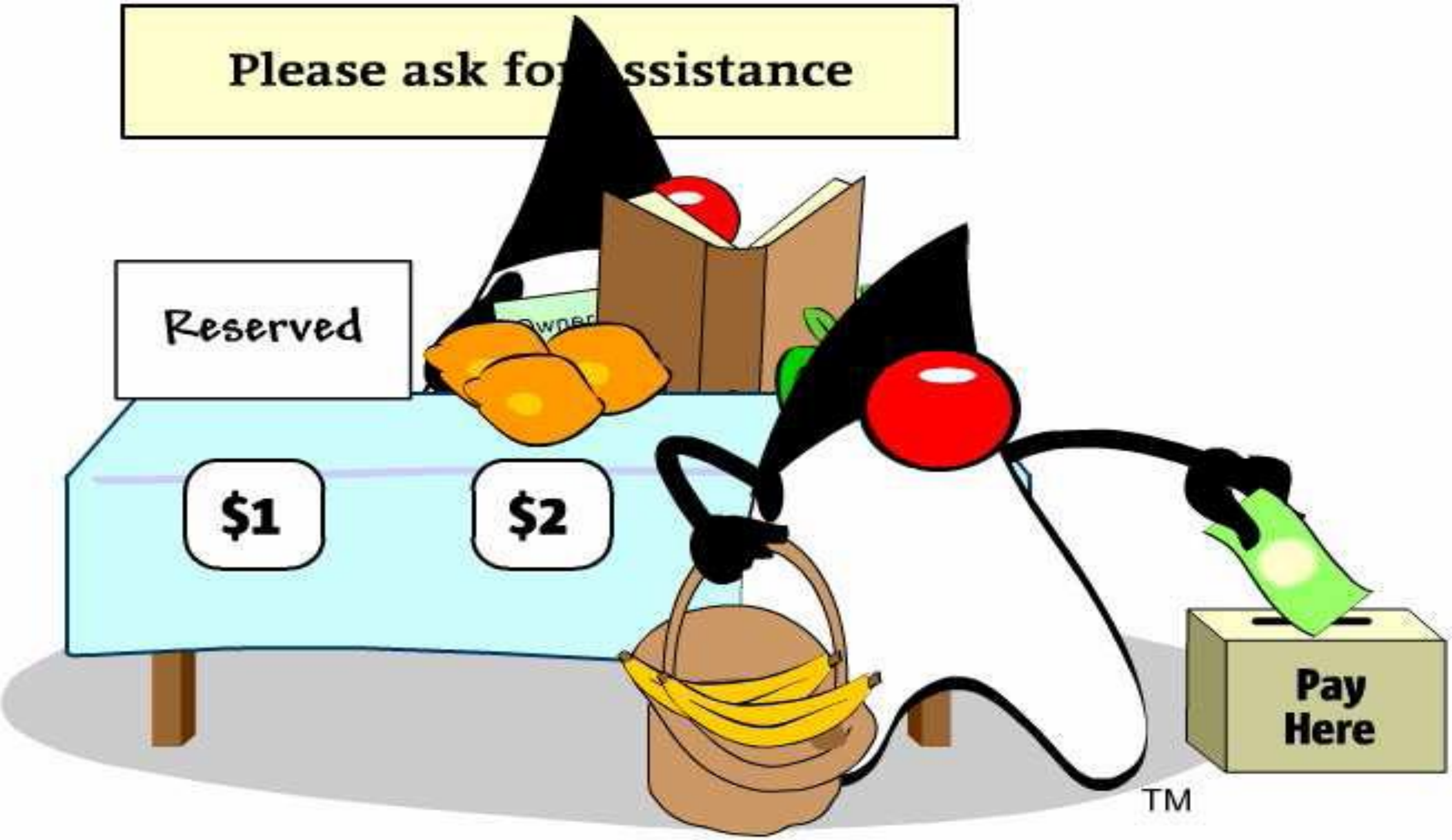
The shopkeeper receives a request to reserve some fruit.
He puts aside the fruit, as requested.

Encapsulation



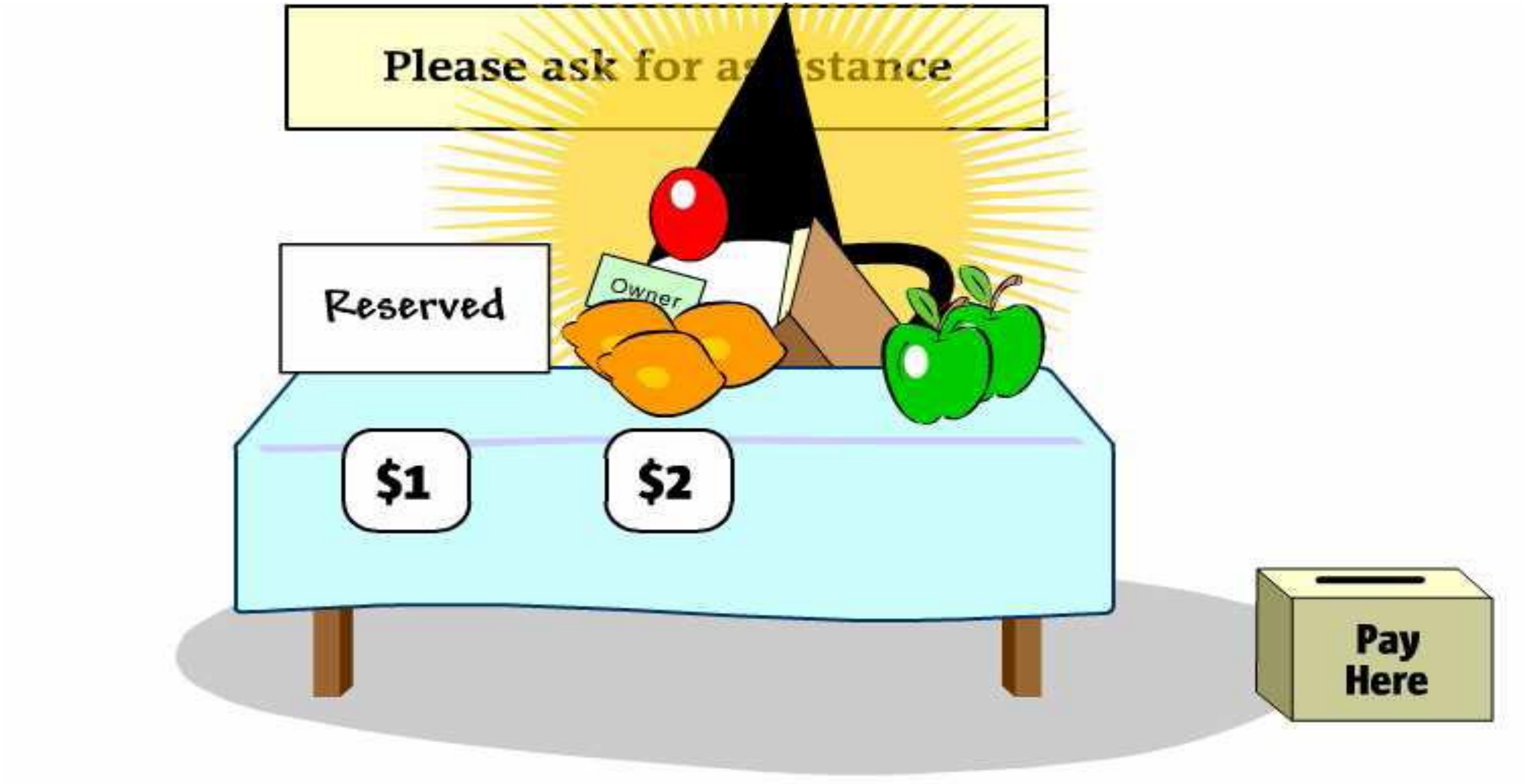
However, because of the fruit stand's design, a customer can choose the fruit that has been reserved.

Encapsulation



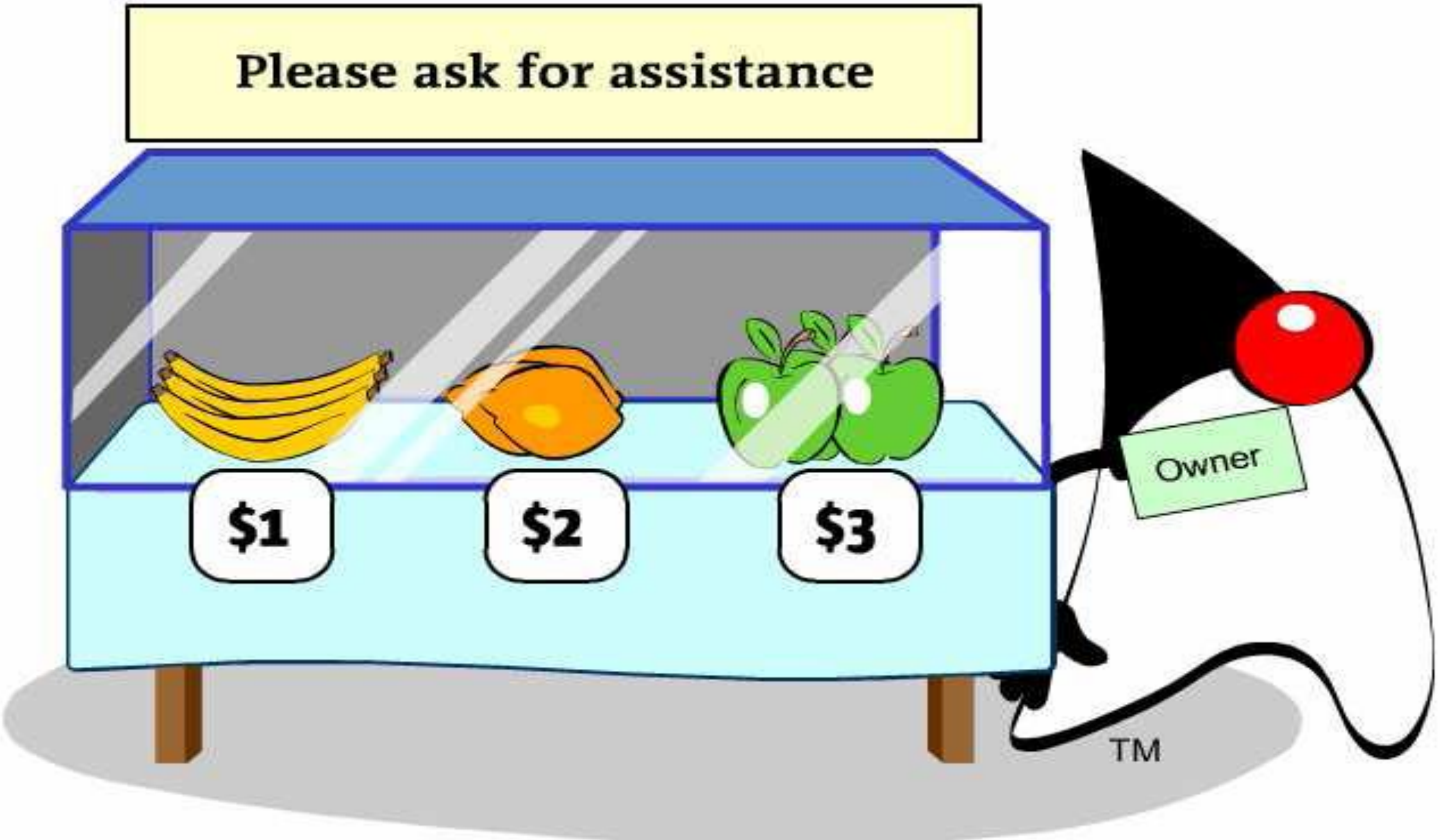
However, because of the fruit stand's design, a customer can choose the fruit that has been reserved.

Encapsulation



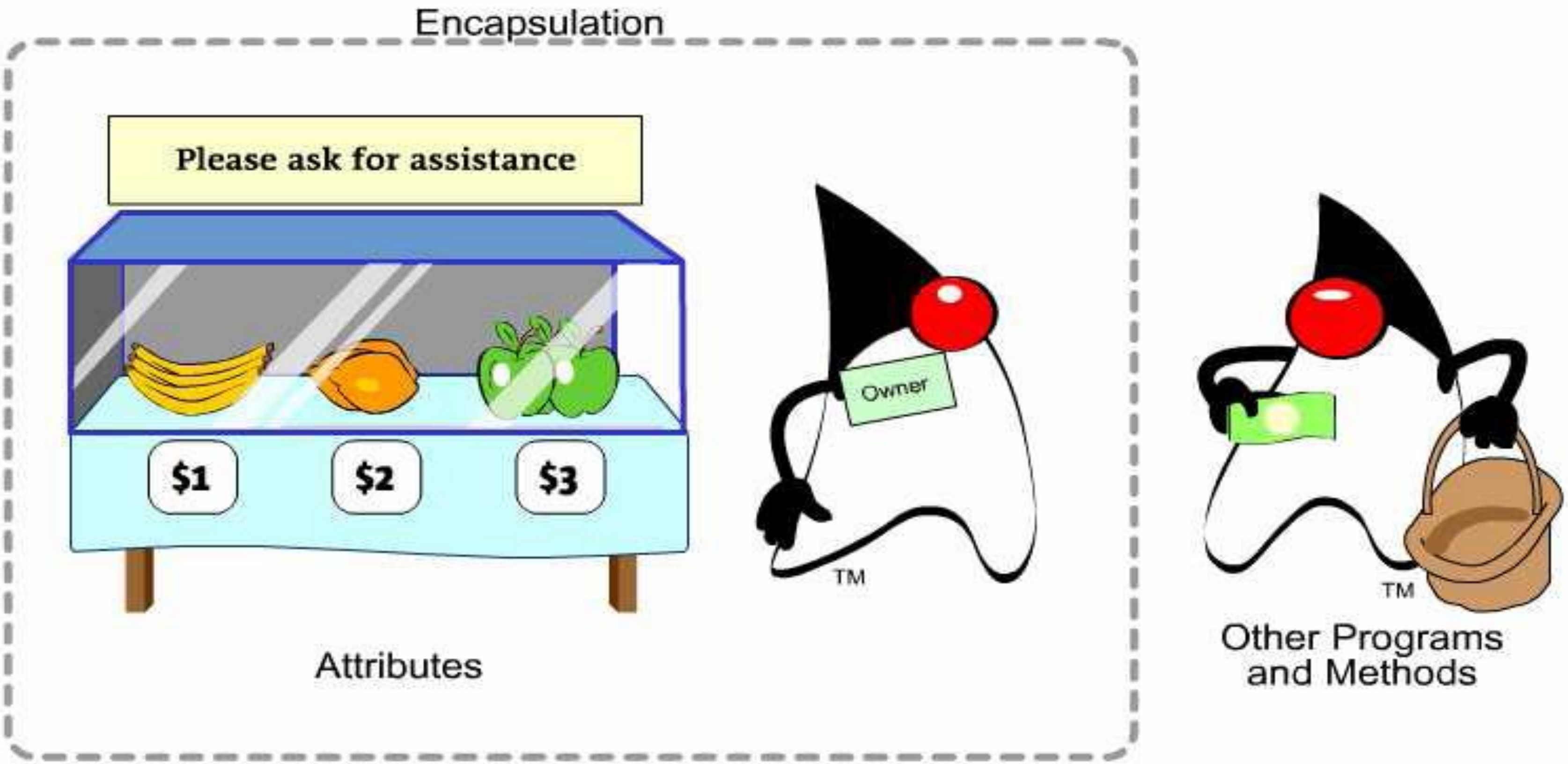
However, because of the fruit stand's design, a customer can choose the fruit that has been reserved.

Encapsulation



By enclosing the fruit behind glass, and controlling all access by customers, the shopkeeper has encapsulated the process of buying fruit.

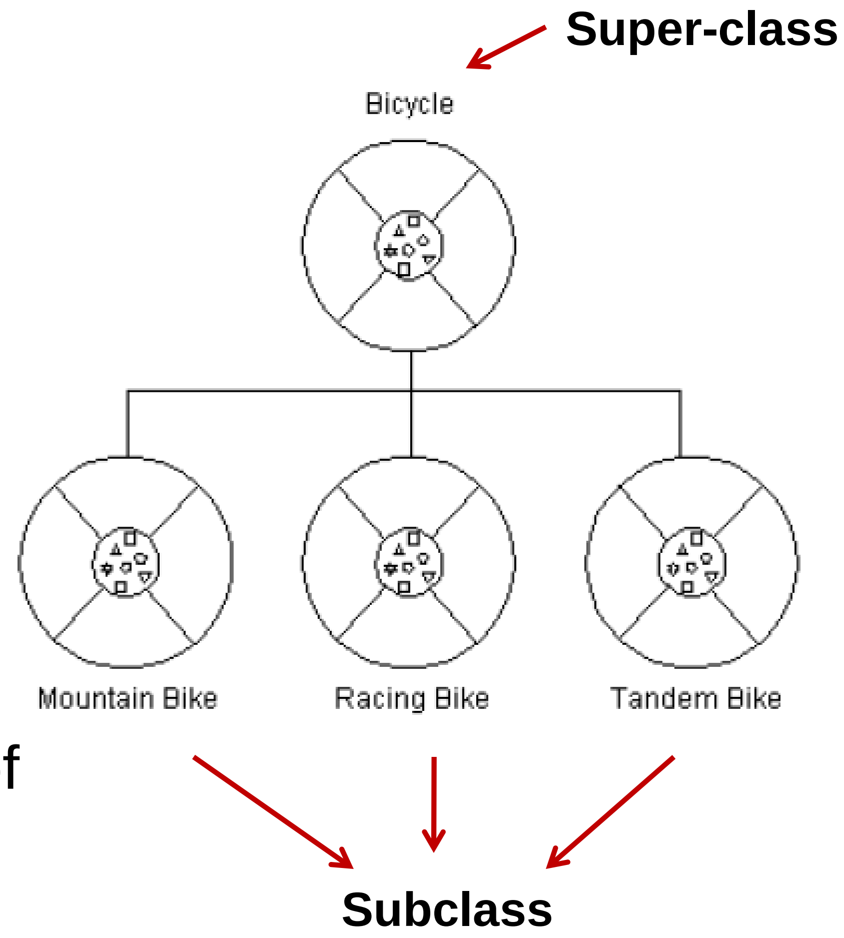
Encapsulation



Encapsulation in the Java programming language is protection of the attributes from abritrary access by other programs and methods. You control how access to attributes is accomplished.

Inheritance

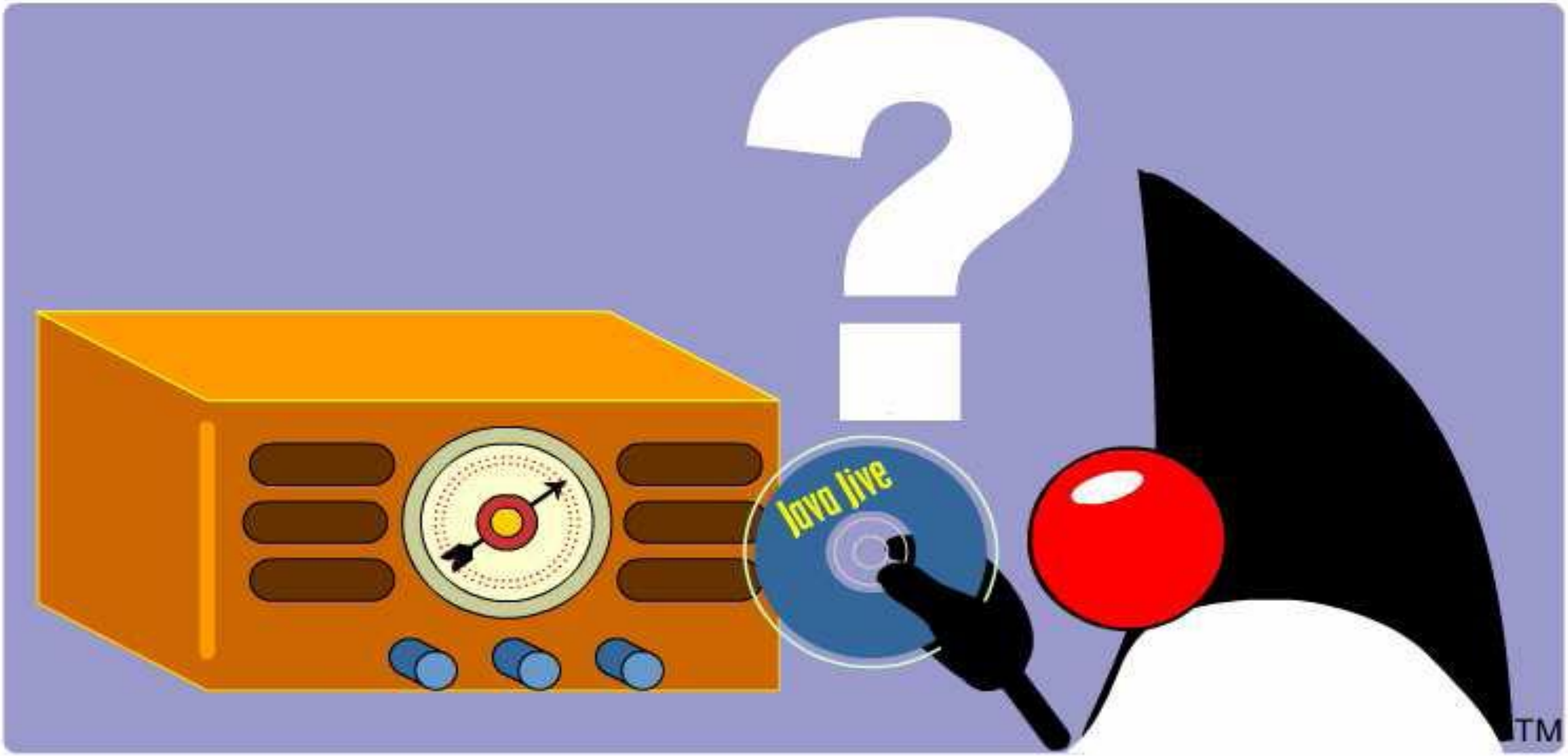
- Features:
  - a class obtains variables and methods from another class
  - the former is called subclass, the latter super-class
  - a sub-class provides a specialized behavior with respect to its super-class
  - inheritance faciliates code reuse and avoids duplication of data



Super-class

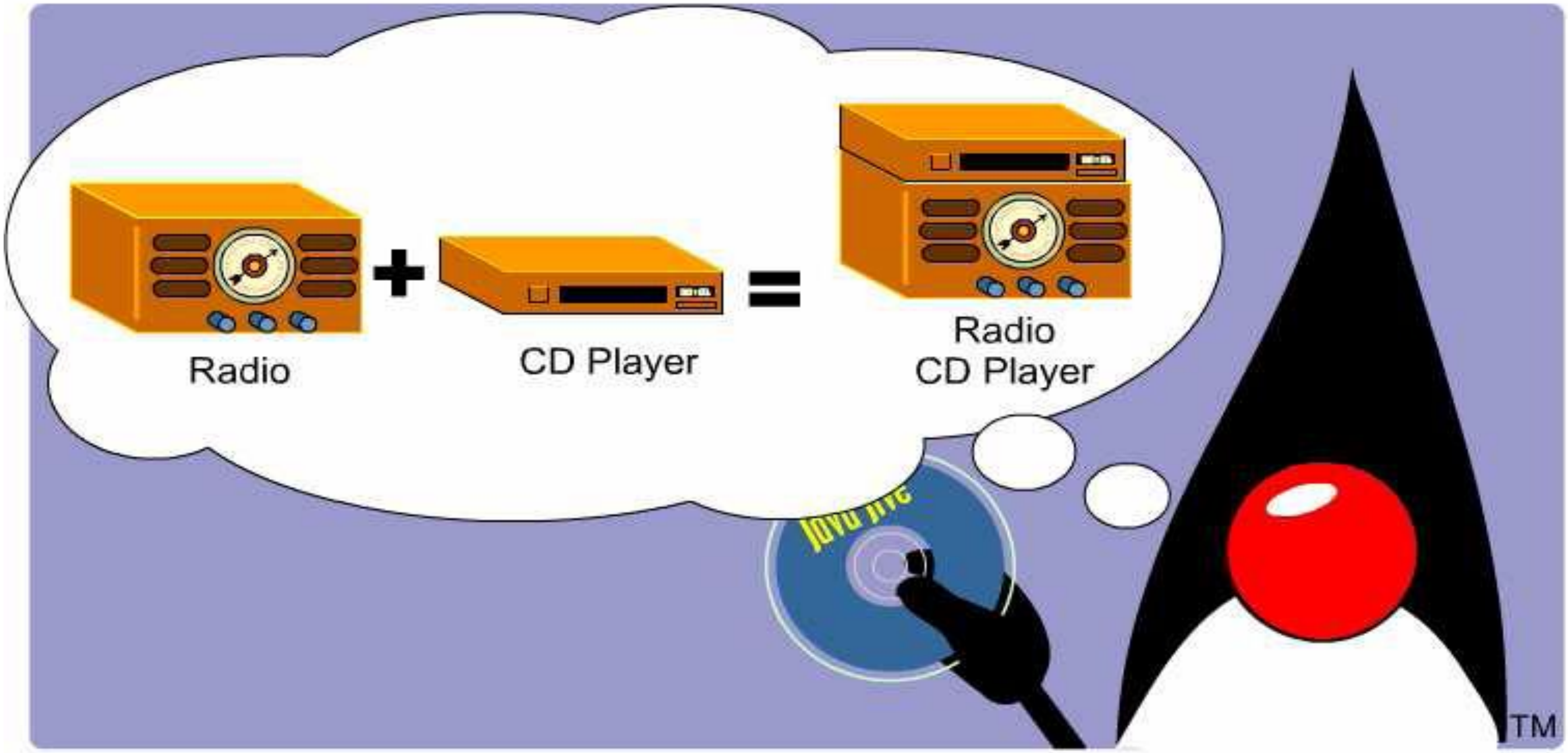
Subclass

Inheritance



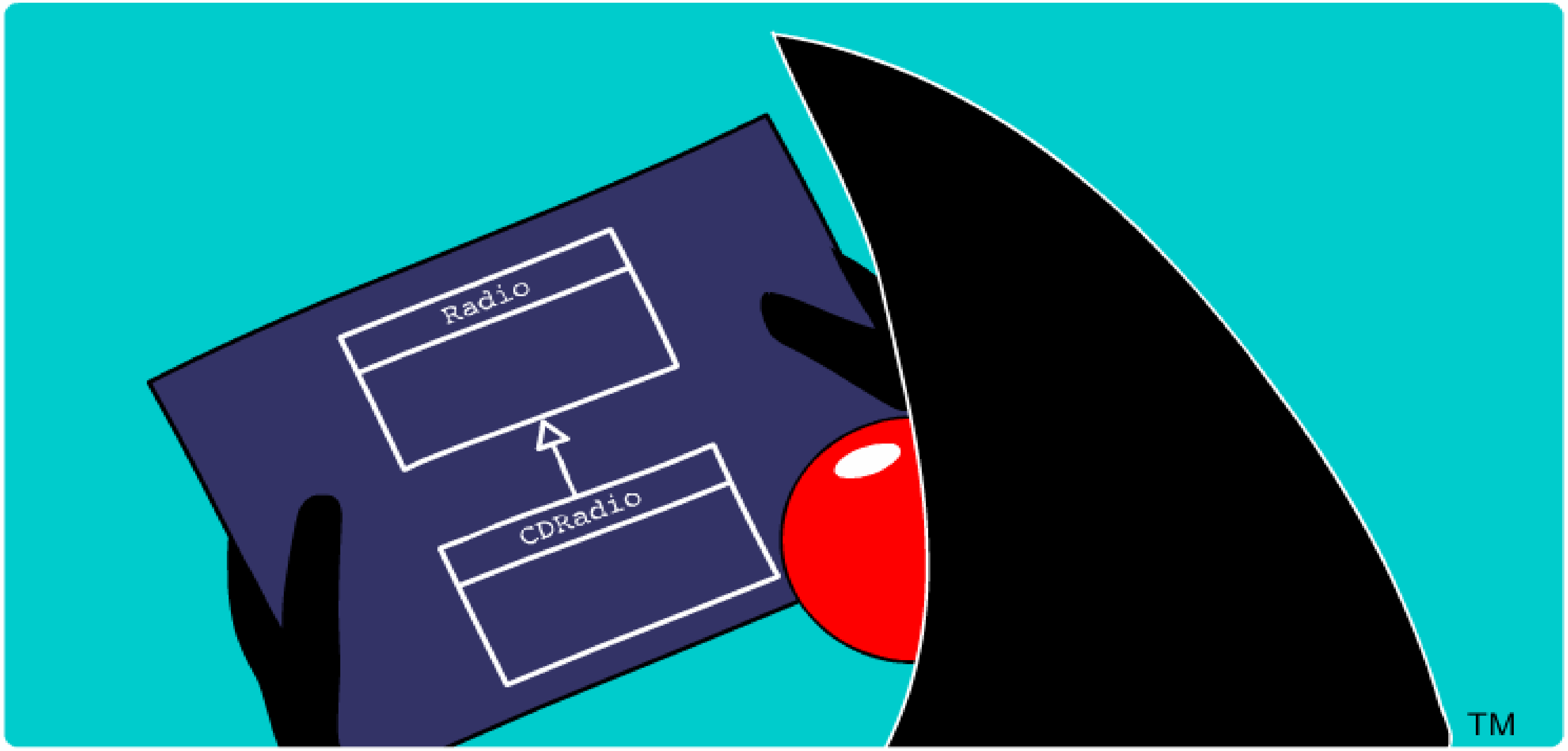
Duke would like to listen to his music CD, but his radio is not equipped with a CD player.

Inheritance



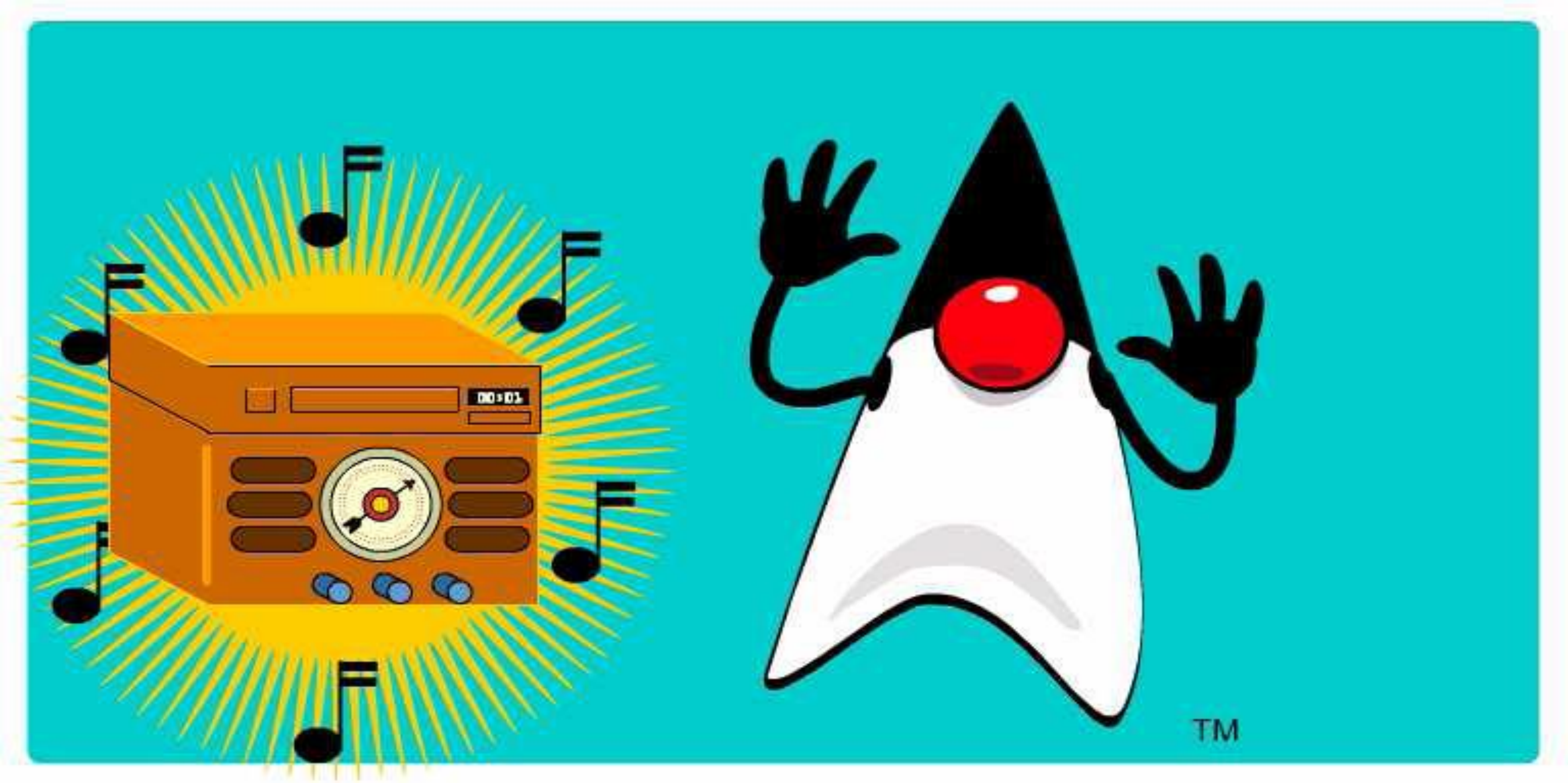
Duke decides he will create a specialized radio, one that plays CDs.

Inheritance



Duke creates a blueprint for a radio CD player.

Inheritance

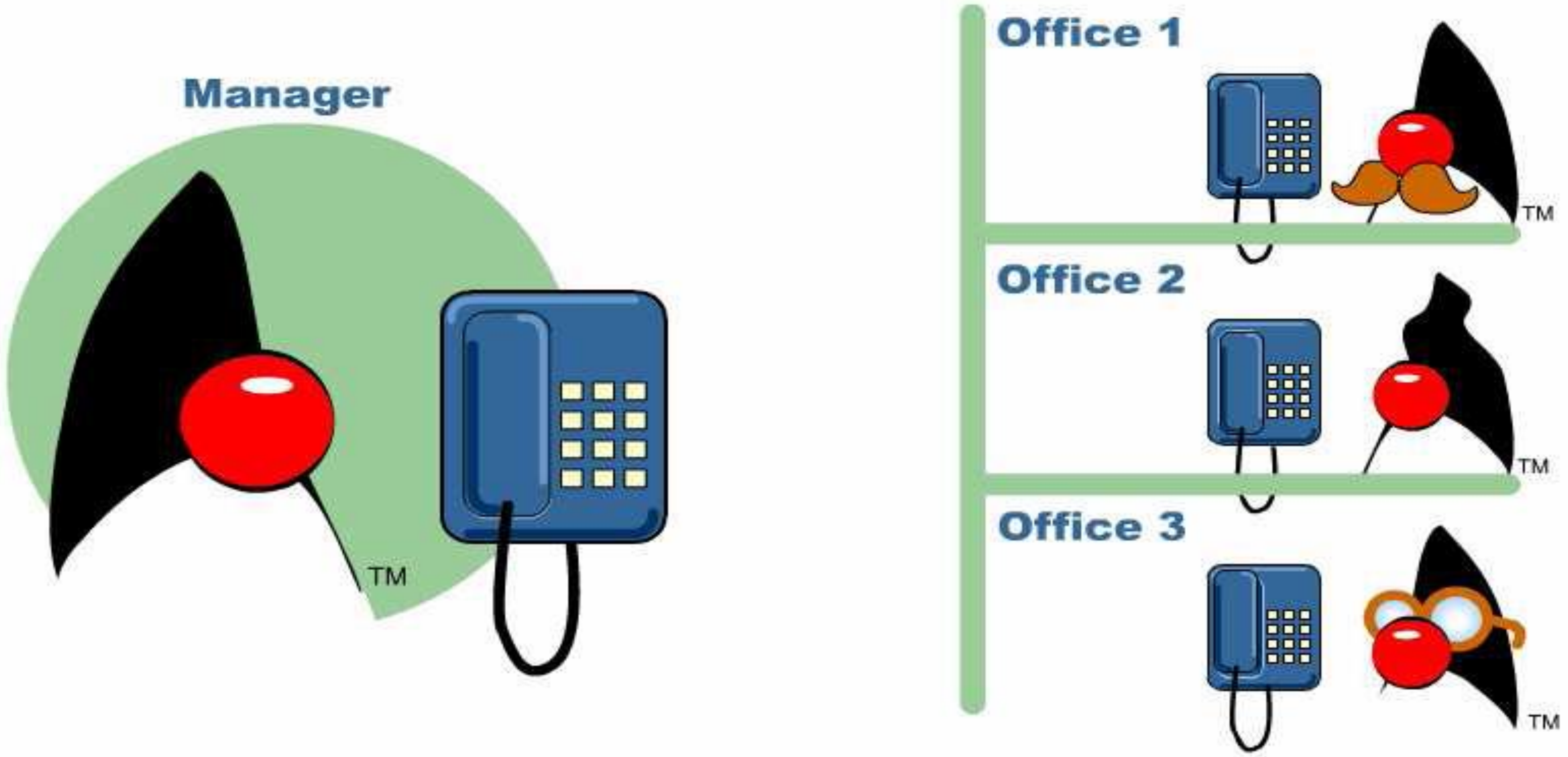


Now Duke can enjoy listening to CDs on his radio CD player.

Polymorphism

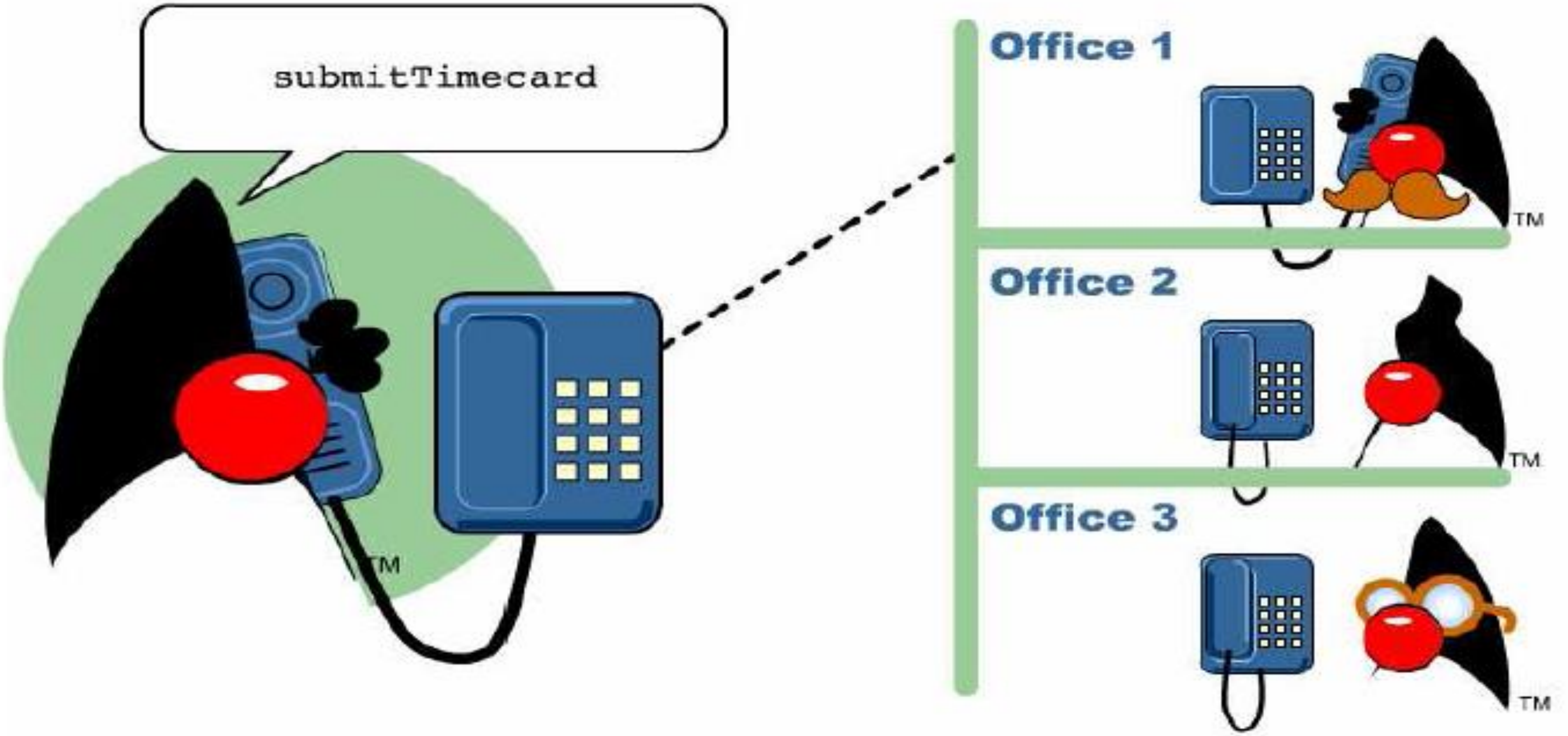
- Polymorphism = many different (poly) forms of objects that share a common interface respond differently when a method of that interface is invoked.
- Polymorphism is enabled by inheritance:
  - a super-class defines an interface that all sub-classes must follow
  - it is up to the sub-classes how this interface is implemented; a subclass may override methods of its super-class
- Therefore, objects from the classes related by inheritance may receive the same requests but respond to such requests in their own ways.

Polymorphism



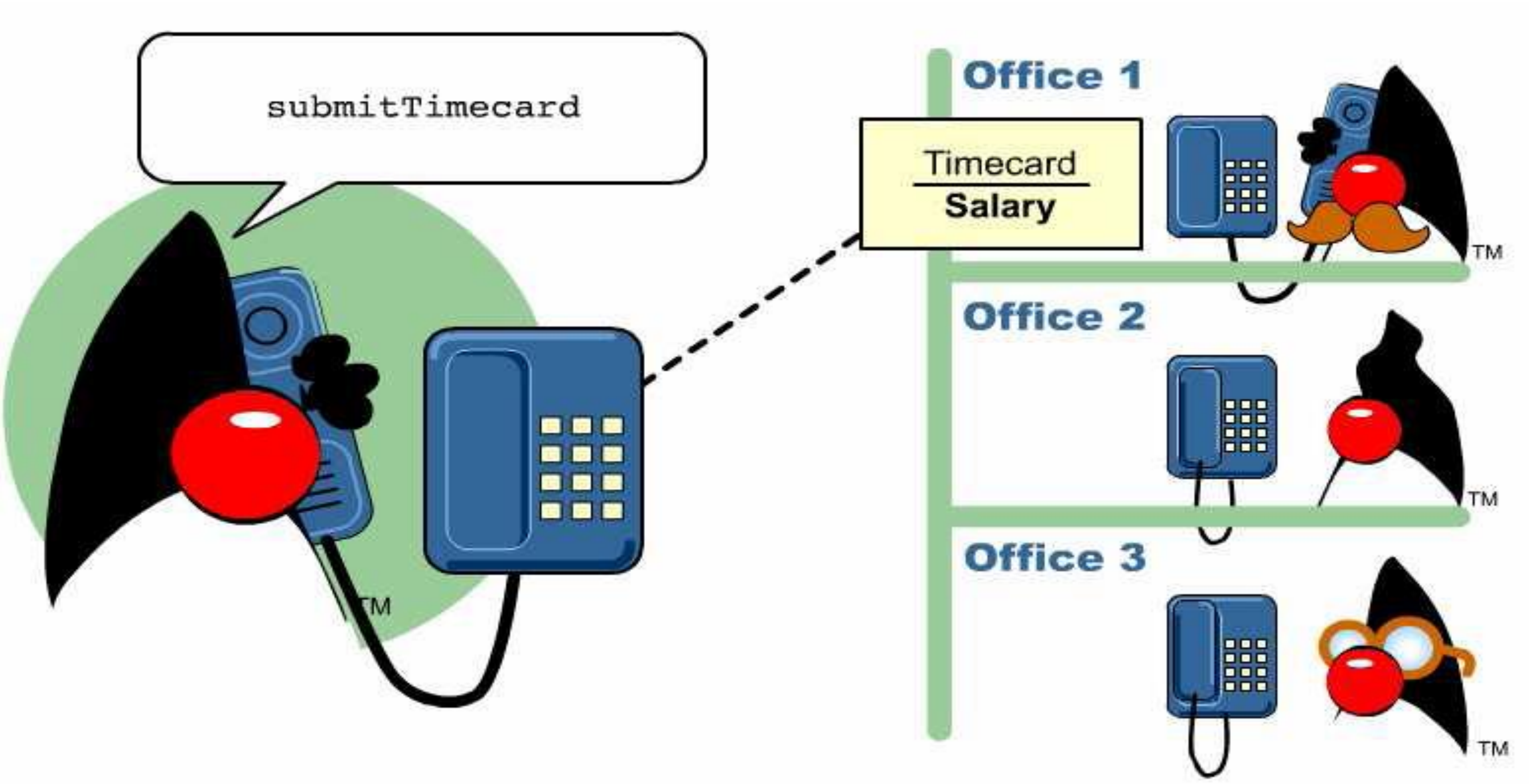
To illustrate polymorphism, imagine a manager with a phone connected to several employee offices.

Polymorphism



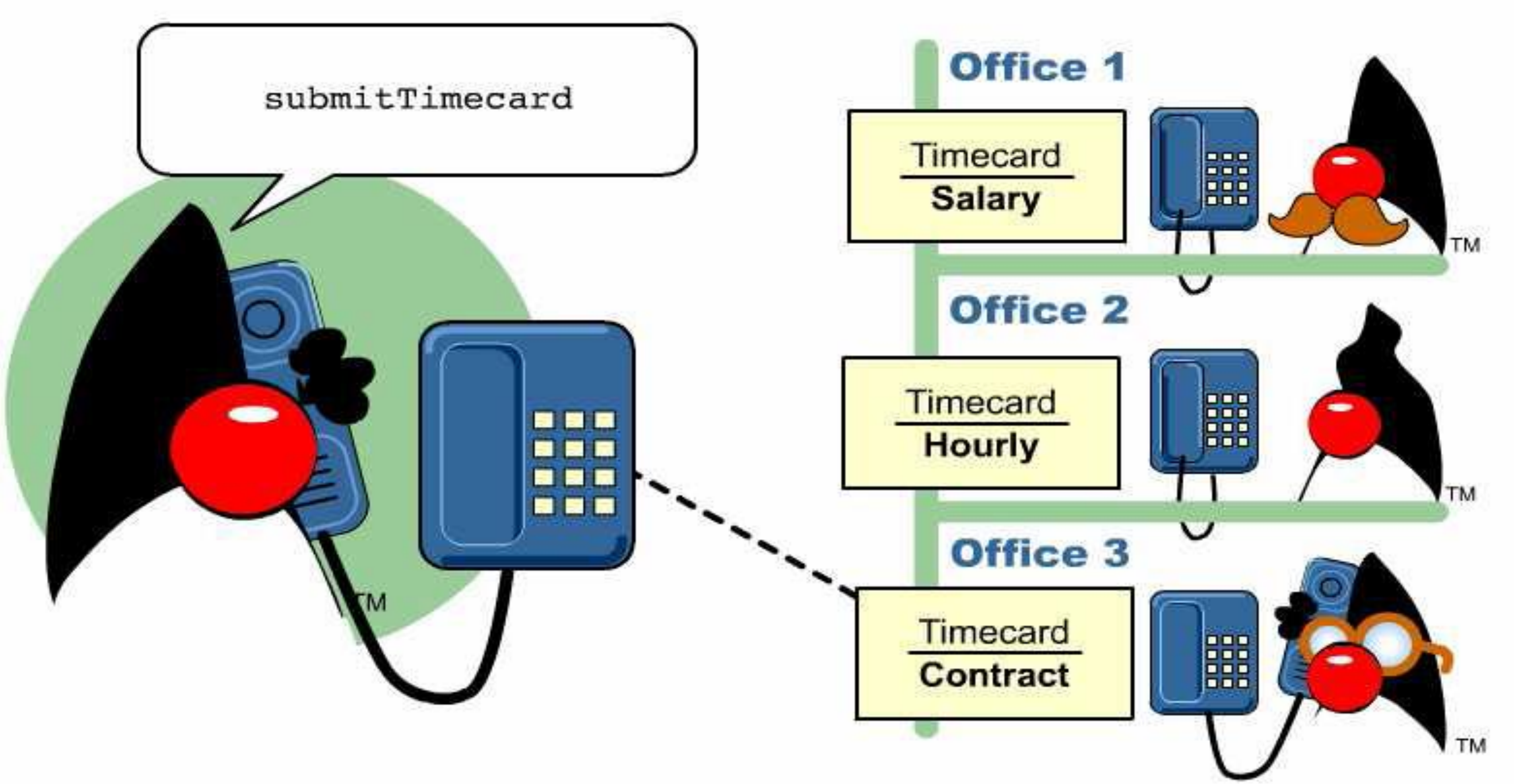
The manager places a call to the first office, and sends the `submitTimecard` method.

Polymorphism



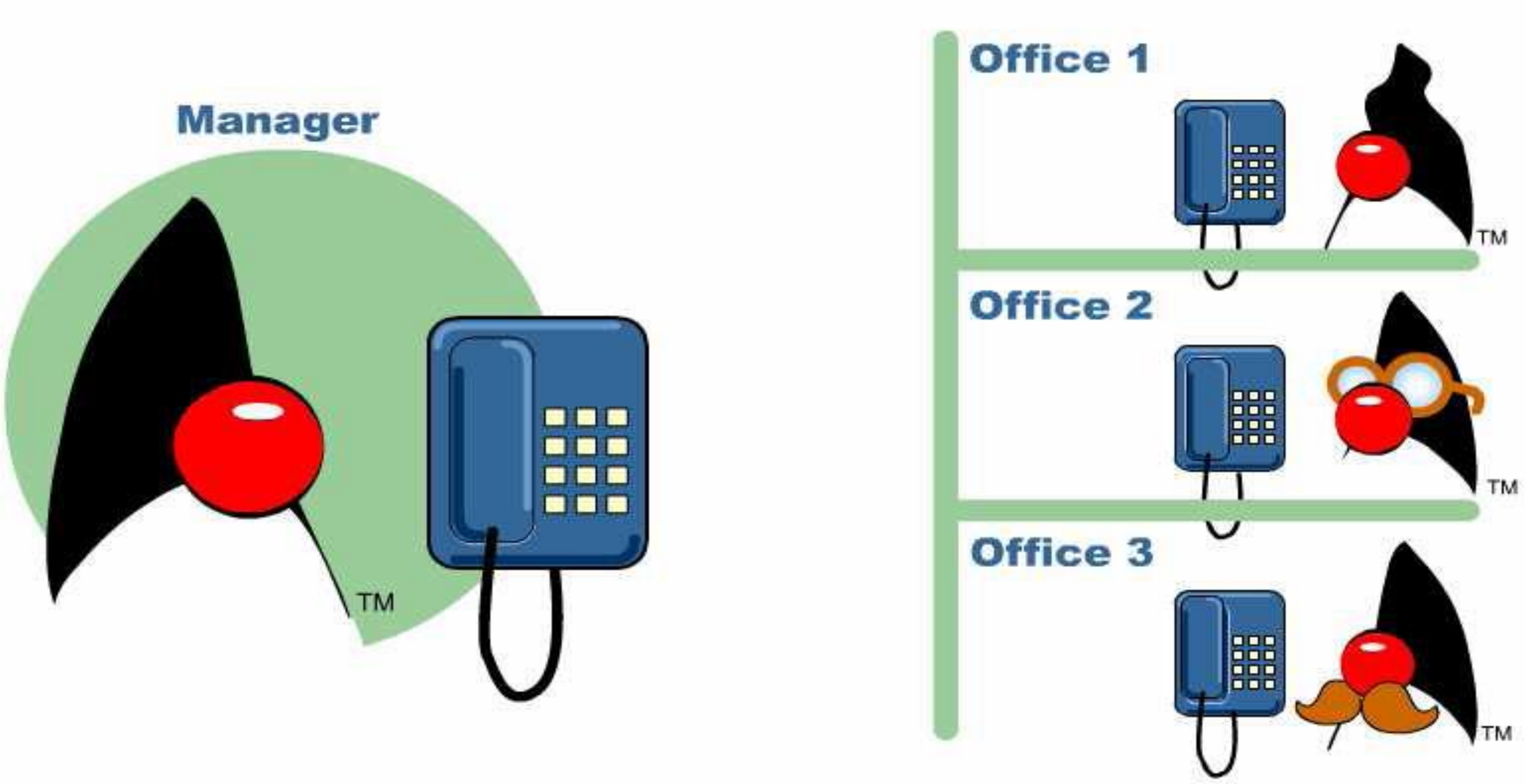
The employee in the first office is on salary. Salaried employees have their own method for submitting a timecard, which this employee follows.

Polymorphism



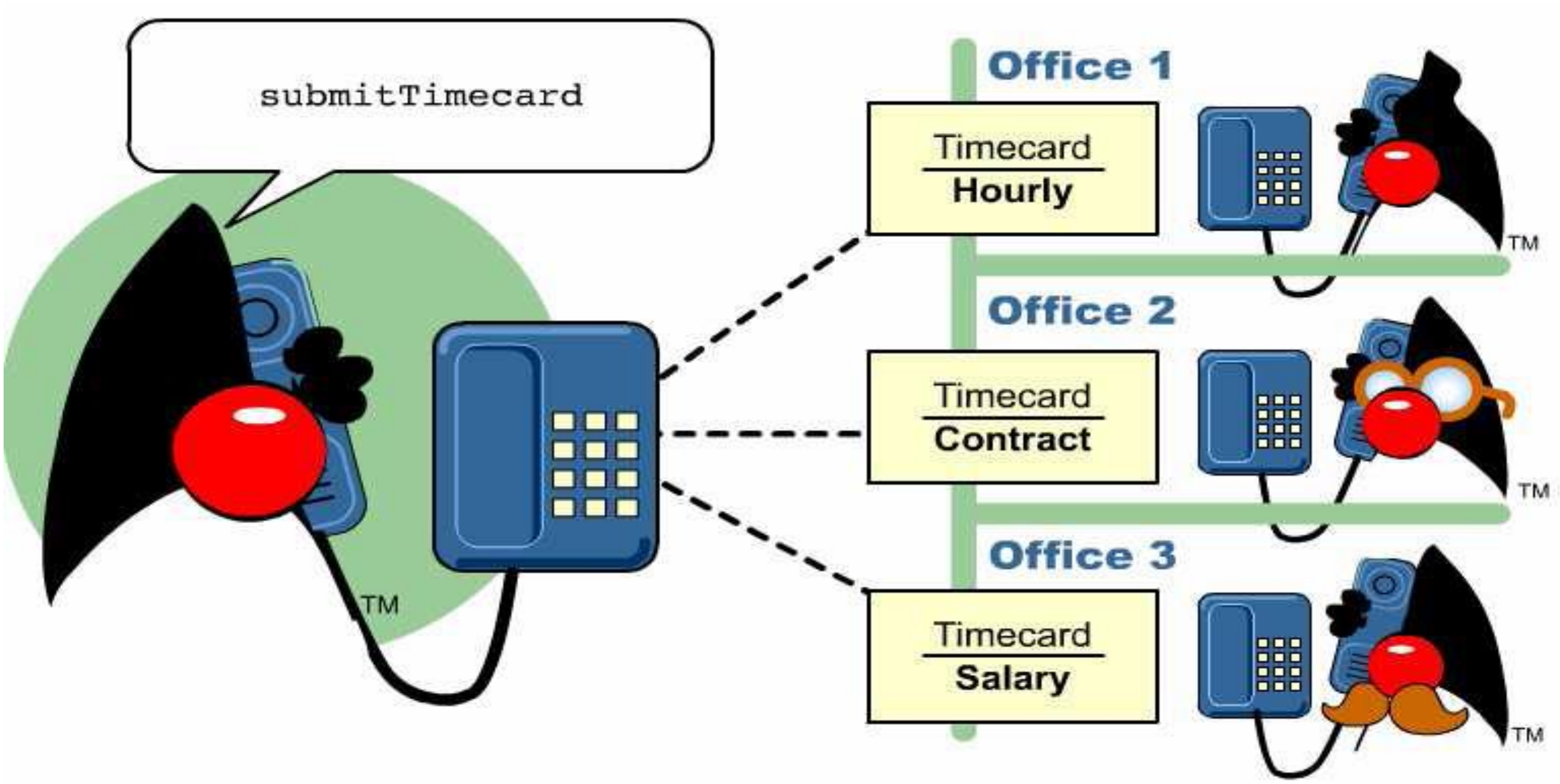
The manager places calls to each of the other offices. The employees in each office follow their own `submitTimecard` method.

Polymorphism



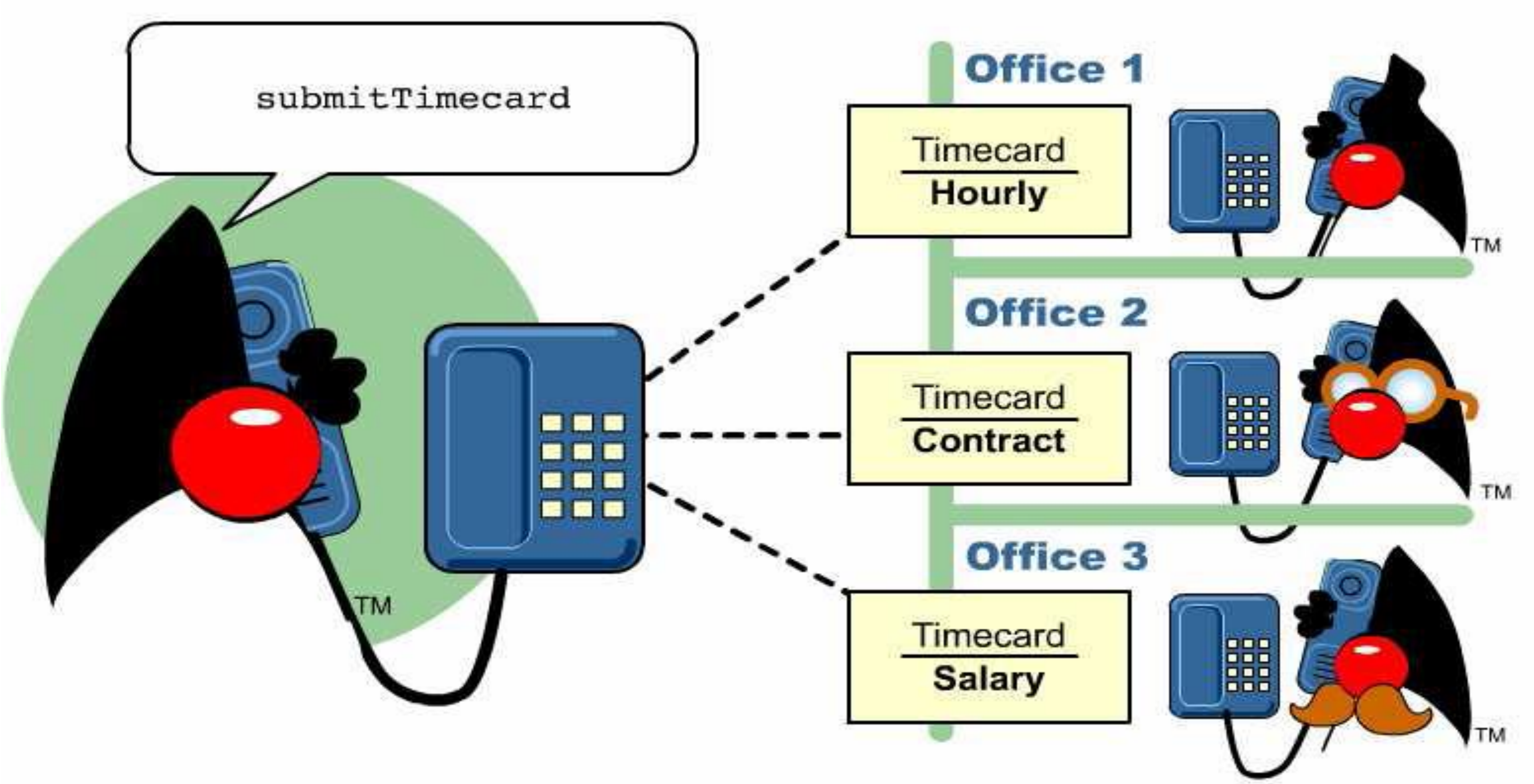
During the next pay period, the employees have switched offices.

Polymorphism



The manager can still place the same call to all the offices, knowing that the employees will follow their own `submitTimcard` method.

Polymorphism



Polymorphism is the ability for objects from separate, yet related classes to receive the same message, but act on it in their own way.

Object

- Everything in Java is an object.
- Well ... almost.
- Object lifecycle:
  - Creation
  - Usage
  - Destruction

- As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class:

- Declaration: A variable declaration with a variable name with an object type.
- Instantiation: The 'new' keyword is used to create the object.
- Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Creating Objects

- <variableName>: the name of the object reference that will refer to the object that you want to create
  □ <className>: the name of an existing class
  □ <classConstructor>: a constructor of the class
  □ The right side of the equation creates the object of the class specified by <className> with the new operator, and assigns it to <variableName> (i.e. <variableName> points to it)

Object Creation

- A variable s is declared to refer to the objects of type/class String:

```
String s;
```

- The value of s is null; it does not yet refer to any object.
- A new String object is created in memory with initial “abc” value:

```
String s = new String(“abc”);
```

- Now s contains the address of this new object.

Object Usage

- Objects are used mostly through variables.
- Four usage scenarios:
  - one variable, one object
  - two variables, one object
  - two variables, two objects
  - one variable, two objects

One Variable, One Object

- One variable, one object:

  String s = new String(“abc”);

- What can you do with the object addressed by s?
  - Check the length: s.length() == 3
  - Return the substring: s.substring(2)
  - etc.
- Depending what is allowed by the definition of String.

Two Variables, One Object

- Two variables, one object:

```
String s1 = new String(“abc”);
String s2;
```

- Assignment copies the address, not value:

```
s2 = s1;
```

- Now s1 and s2 both refer to one object. After

```
s1 = null;
```

- s2 still points to this object.

Two Variables, Two Objects

- Two variables, two objects:

```
String s1 = new String(“abc”);
String s2 = new String(“abc”);
```

- s1 and s2 objects have initially the same values:

```
s1.equals(s2) == true
```

- But they are not the same objects:

```
(s1 == s2) == false
```

- They can be changed independently of each other.

One Variable, Two Objects

- One variable, two objects:

```
String s = new String(“abc”);
s = new String(“cba”);
```

- The “abc” object is no longer accessible through any variable.

Object Destruction

- A program accumulates memory through its execution.
- Two mechanism to free memory that is no longer need by the program:
  - manual – done in C/C++
  - automatic – done in Java
- In Java, when an object is no longer accessible through any variable, it is eventually removed from the memory by the garbage collector.
- Garbage collector is parts

of the Java Run-Time Environment.

Class

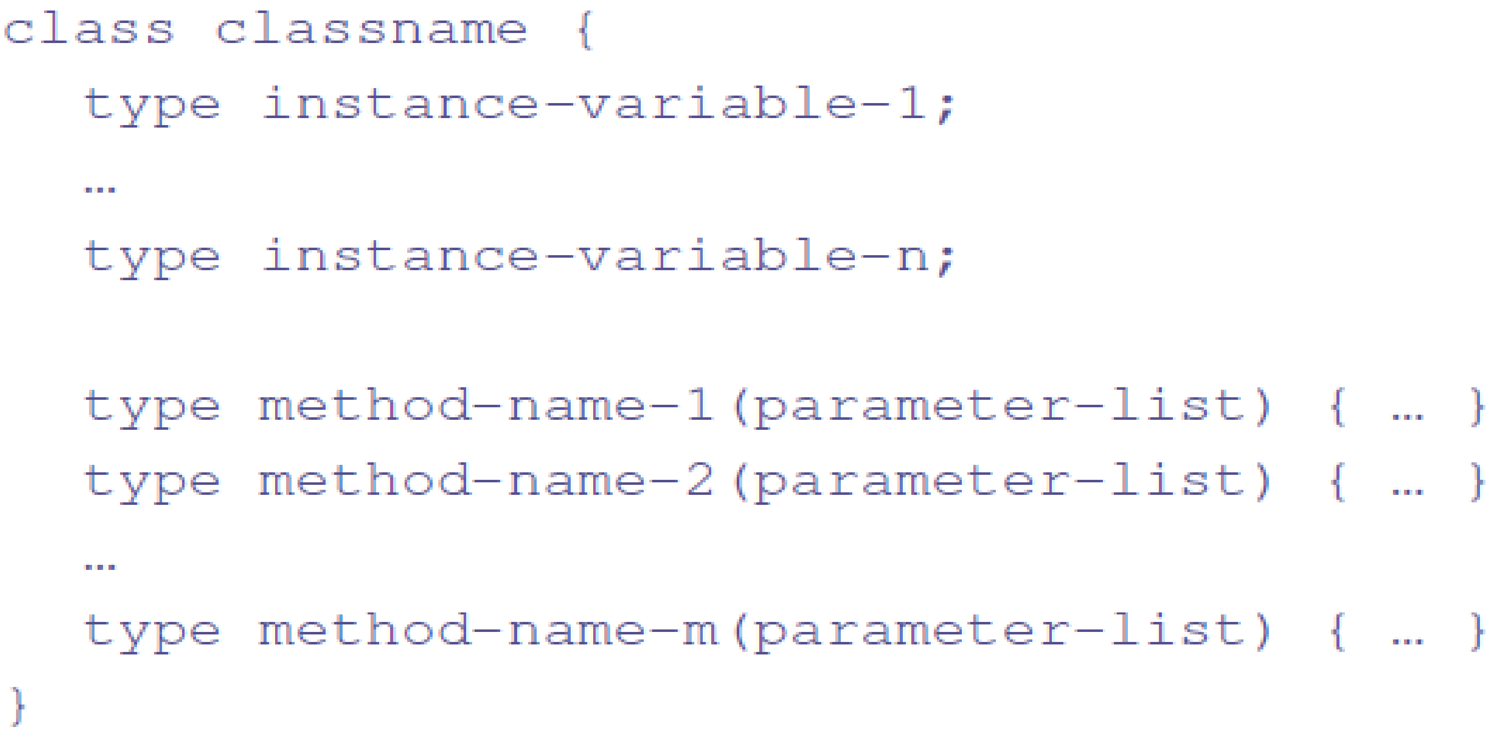
- A basis for the Java language.
- Each concept we wish to describe in Java must be included inside a class.
- A class defines a new data type, whose values are objects:
  - a class is a template for objects
  - an object is an instance of a class

- The general syntax:
  <modifier> class <className> { }
  □<className> specifies the name of the class
  □class is the keyword
  □<modifier> specifies some characteristics of the class:
  • *Access modifiers*: private, protected, *default* and public
  • *Other modifiers*: abstract, final, and strictfp

```
// Class Declaration
public class Dog {
    // Instance Variables
    String breed;
    String size;
    int age;
    String color;
// method 1
    public String getInfo() {
        return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+color);
    }
    public static void main(String[] args) {
        Dog maltese = new Dog();
        maltese.breed="Maltese";
        maltese.size="Small";
        maltese.age=2;
        maltese.color="white";
        System.out.println(maltese.getInfo());
    }
}
```

Class Definition

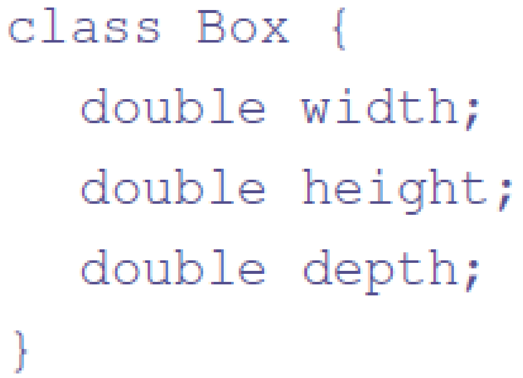
- A class contains a name, several variable declarations (instance variables) and several method declarations. All are called members of the class.
- General form of a class:



```
class classname {
   type instance-variable-1;
   …
   type instance-variable-n;

   type method-name-1(parameter-list) { … }
   type method-name-2(parameter-list) { … }
   …
   type method-name-m(parameter-list) { … }
}
```

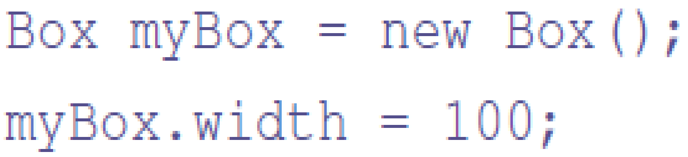
Example: Class

- A class with three variable members:



```
class Box {
   double width;
   double height;
   double depth;
}
```

- A new Box object is created and a new value assigned to its width variable:



```
Box myBox = new Box();
myBox.width = 100;
```

Example: Class Usage

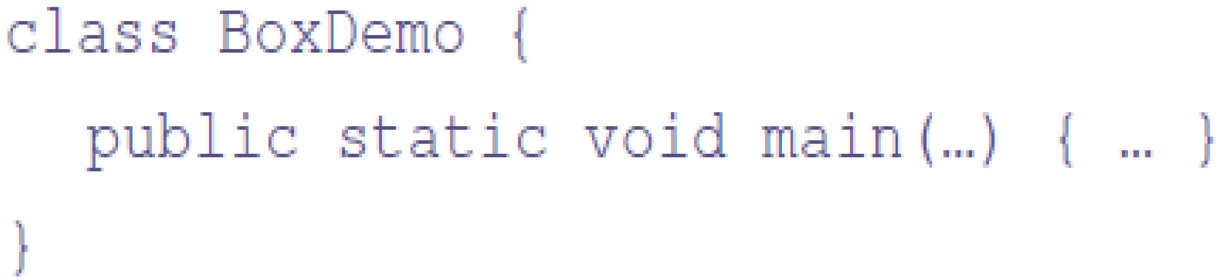
```
class BoxDemo {
  public static void main(String args[]) {
     Box mybox = new Box();
     double vol;

     mybox.width = 10;
     mybox.height = 20;
     mybox.depth = 15;

     vol = mybox.width * mybox.height * mybox.depth;
     System.out.println("Volume is " + vol);
  }
}
```

Compilation and Execution

- Place the Box class definitions in file Box.java:

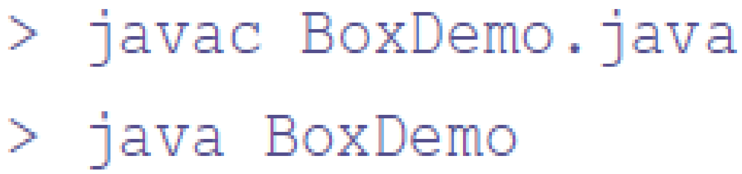
```
class Box { … }
```

- Place the BoxDemo class definitions in file BoxDemo.java:



```
class BoxDemo {
   public static void main(…) { … }
}
```

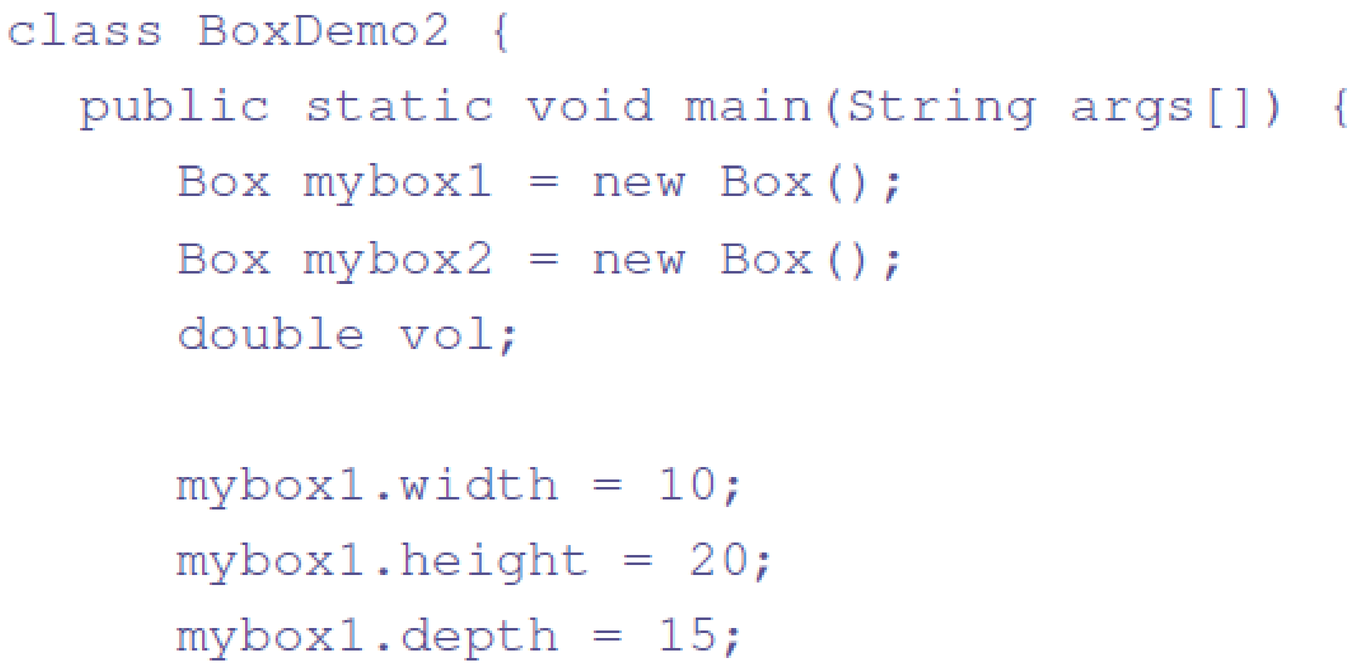
- Compilation and execution:



```
> javac BoxDemo.java
> java BoxDemo
```

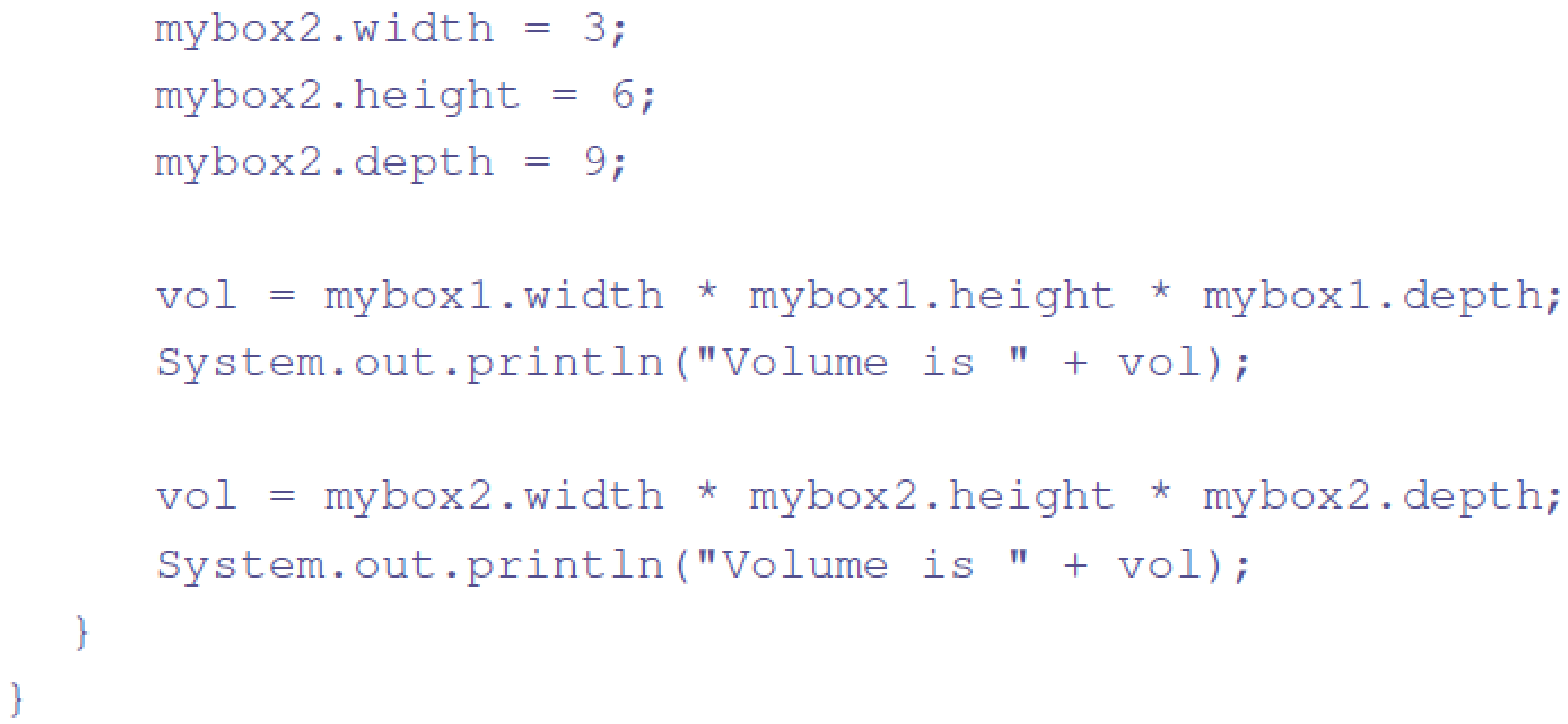
Variable Independence

- Each object has its own copy of the instance variables: changing the variables of one object has no effect on the variables of another object.
- Consider this example:



```
class BoxDemo2 {
  public static void main(String args[]) {
     Box mybox1 = new Box();
     Box mybox2 = new Box();
     double vol;

     mybox1.width = 10;
     mybox1.height = 20;
     mybox1.depth = 15;
```

Variable Independence



```
    mybox2.width = 3;
    mybox2.height = 6;
    mybox2.depth = 9;

    vol = mybox1.width * mybox1.height * mybox1.depth;
    System.out.println("Volume is " + vol);

    vol = mybox2.width * mybox2.height * mybox2.depth;
    System.out.println("Volume is " + vol);
  }
}
```

What are the printed volumes of both boxes?

Declaring Objects

- Obtaining objects of a class is a two-stage process:
  - Declare a variable of the class type:

    Box myBox;

    - The value of myBox is a reference to an object, if one exists, or null.
    - At this moment, the value of myBox is null.
  - Acquire an actual, physical copy of an object and assign its address to the variable. How to do this?

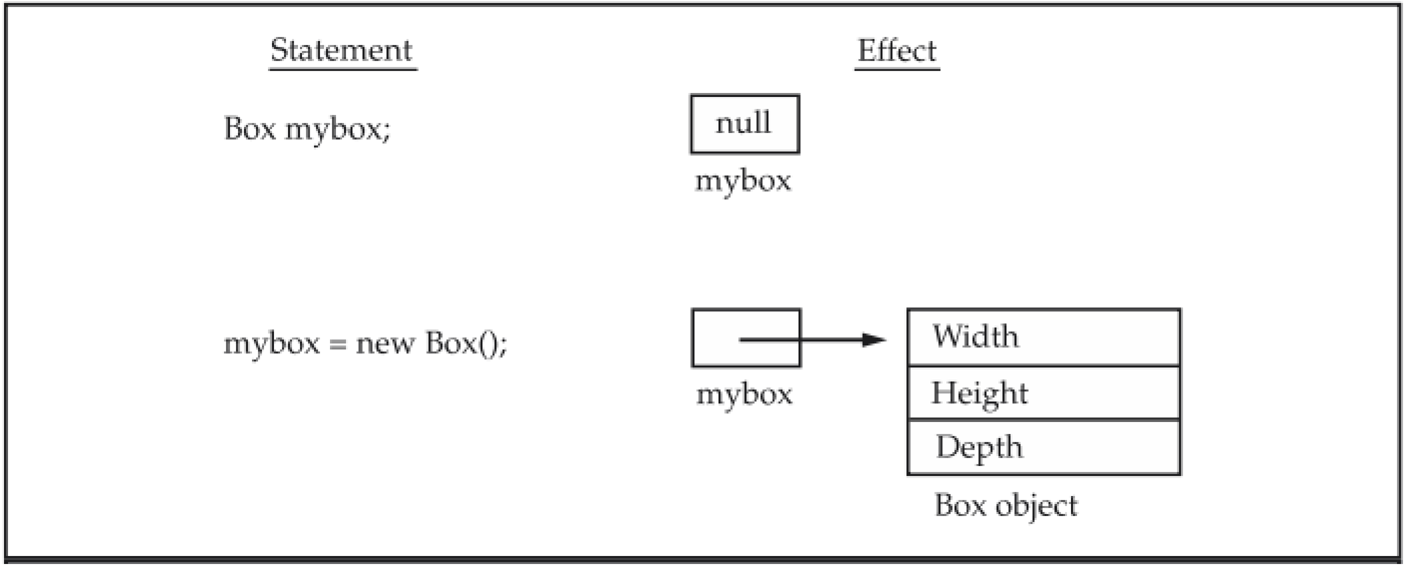
Operator new

- Allocates memory for a Box object and returns its address:

```
Box myBox = new Box();
```

- The address is then stored in the myBox reference variable.

Declaring VS Operator new



```
Box mybox;

mybox = new Box();
```

Memory Allocation

- Memory is allocated for objects dynamically.
- This has both advantages and disadvantages:
  - as many objects are created as needed
  - allocation is uncertain – memory may be insufficient
- Variables of simple types do not require new:

```
int n = 1;
```

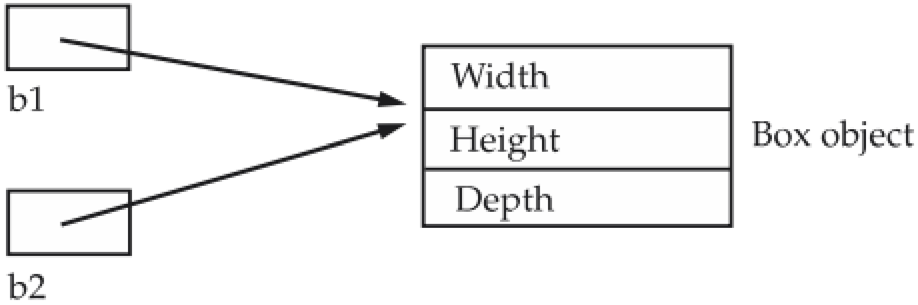
- In the interest of efficiency, Java does not implement simple types as objects. Variables of simple types hold values, not references.

Assigning Reference Variable

- Assignment copies address, not the actual value:
  - Box b1 = new Box();
  - Box b2 = b1;
- Both variables point to the same object.
- Variables are not in any way connected. After

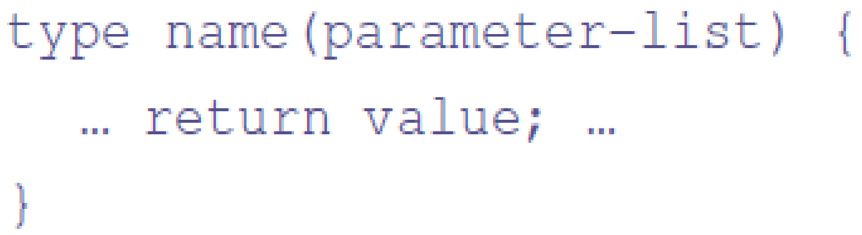
    b1 = null;

  b2 still refers to the original object.



Methods

- General form of a method definition:

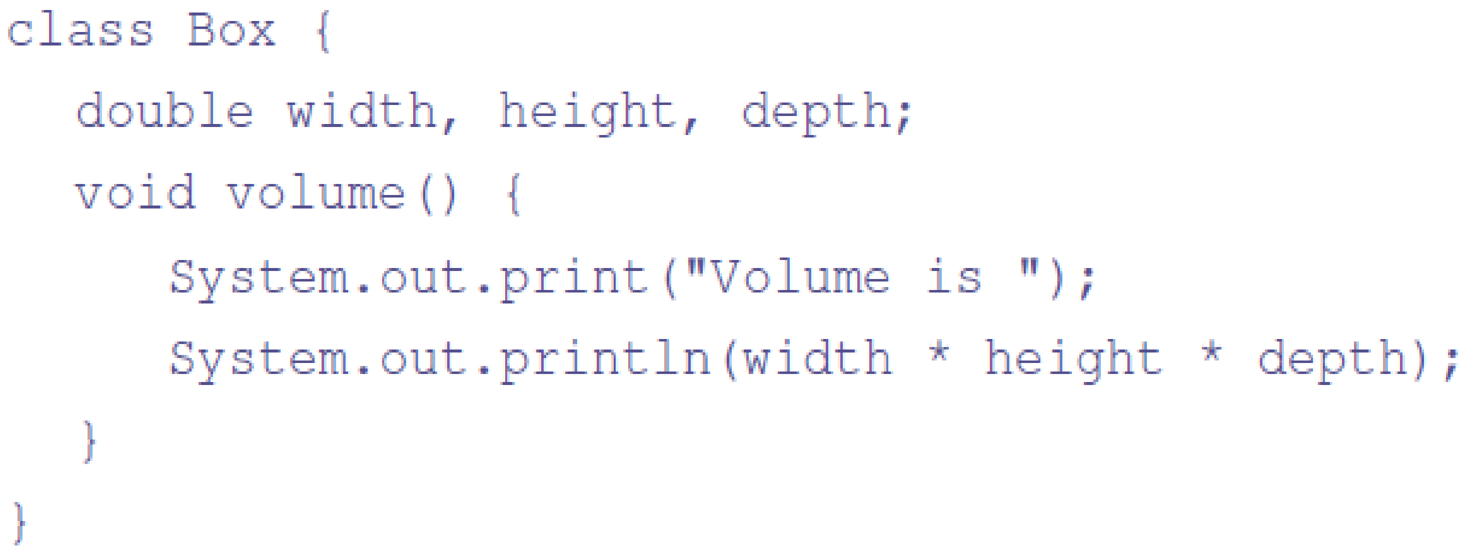


```
type name(parameter-list) {
   … return value; …
}
```

- Components:
  - type - type of values returned by the method. If a method does not return any value, its return type must be void.
  - name is the name of the method
  - parameter-list is a sequence of type-identifier lists separated by commas
  - return value indicates what value is returned by the method.

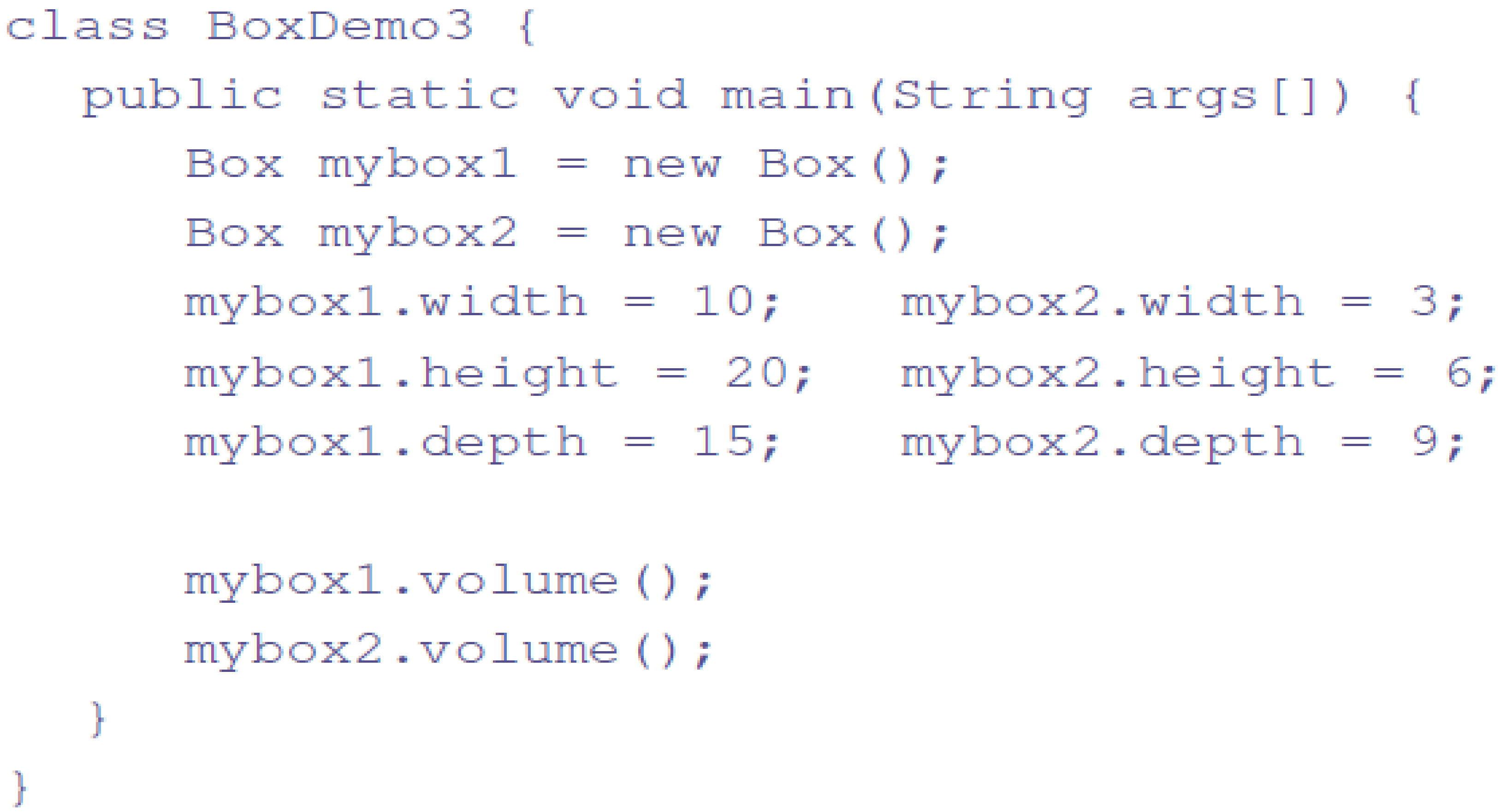
Example: Method

- Classes declare methods to hide their internal data structures, as well as for their own internal use:
- Within a class, we can refer directly to its member variables:



```
class Box {
  double width, height, depth;
  void volume() {
     System.out.print("Volume is ");
     System.out.println(width * height * depth);
  }
}
```

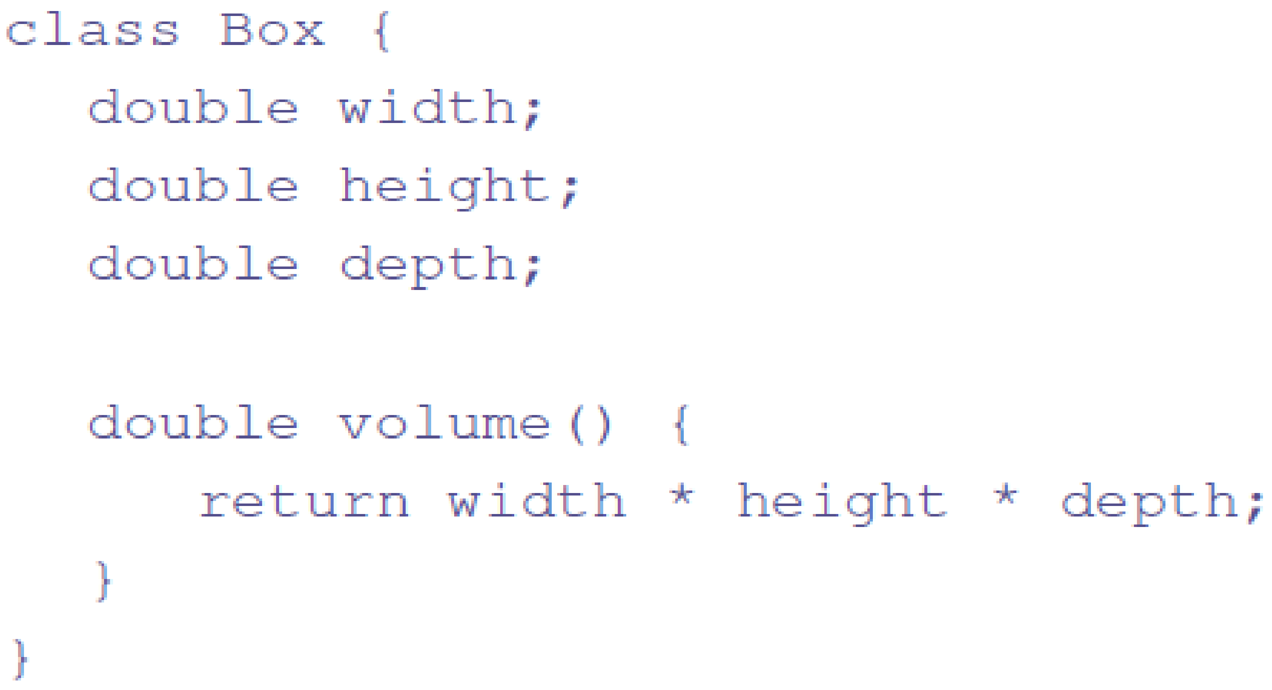
Example: Method



```
class BoxDemo3 {
  public static void main(String args[]) {
     Box mybox1 = new Box();
     Box mybox2 = new Box();
     mybox1.width = 10;    mybox2.width = 3;
     mybox1.height = 20;   mybox2.height = 6;
     mybox1.depth = 15;    mybox2.depth = 9;

     mybox1.volume();
     mybox2.volume();
  }
}
```

Value-Returning Method

- The type of an expression returning value from a method must agree with the return type of this method:



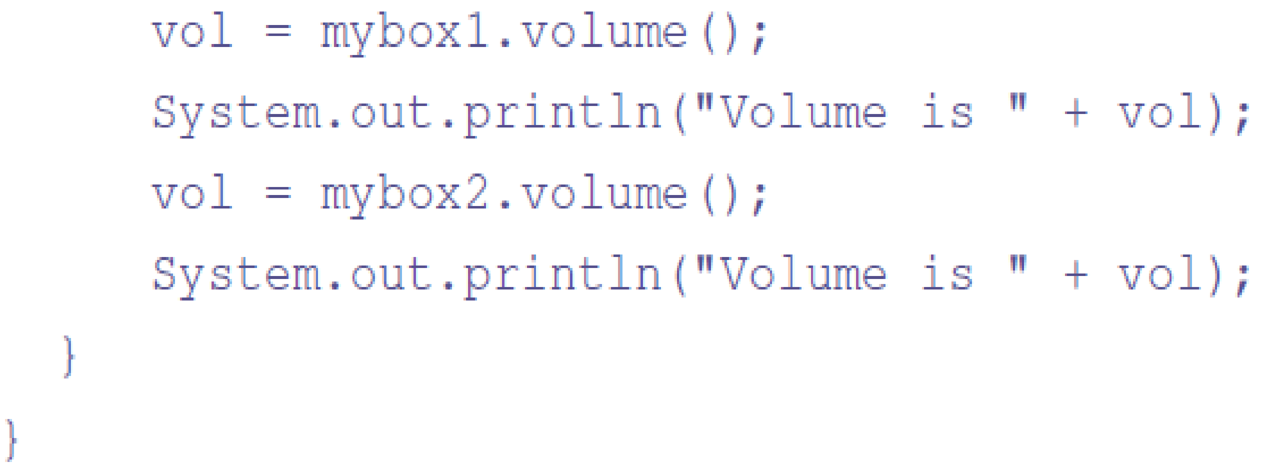
```
class Box {
   double width;
   double height;
   double depth;

   double volume() {
      return width * height * depth;
   }
}
```

Value-Returning Method

```
class BoxDemo4 {
  public static void main(String args[]) {
     Box mybox1 = new Box();
     Box mybox2 = new Box();
     double vol;
     mybox1.width = 10;
     mybox2.width = 3;
     mybox1.height = 20;
     mybox2.height = 6;
     mybox1.depth = 15;
     mybox2.depth = 9;
```

Value-Returning Method

- The type of a variable assigned the value returned by a method must agree with the return type of this method:



```
     vol = mybox1.volume();
     System.out.println("Volume is " + vol);
     vol = mybox2.volume();
     System.out.println("Volume is " + vol);
  }
}
```

Parameterized Method

- Parameters increase generality and applicability of a method:
  - method without parameters

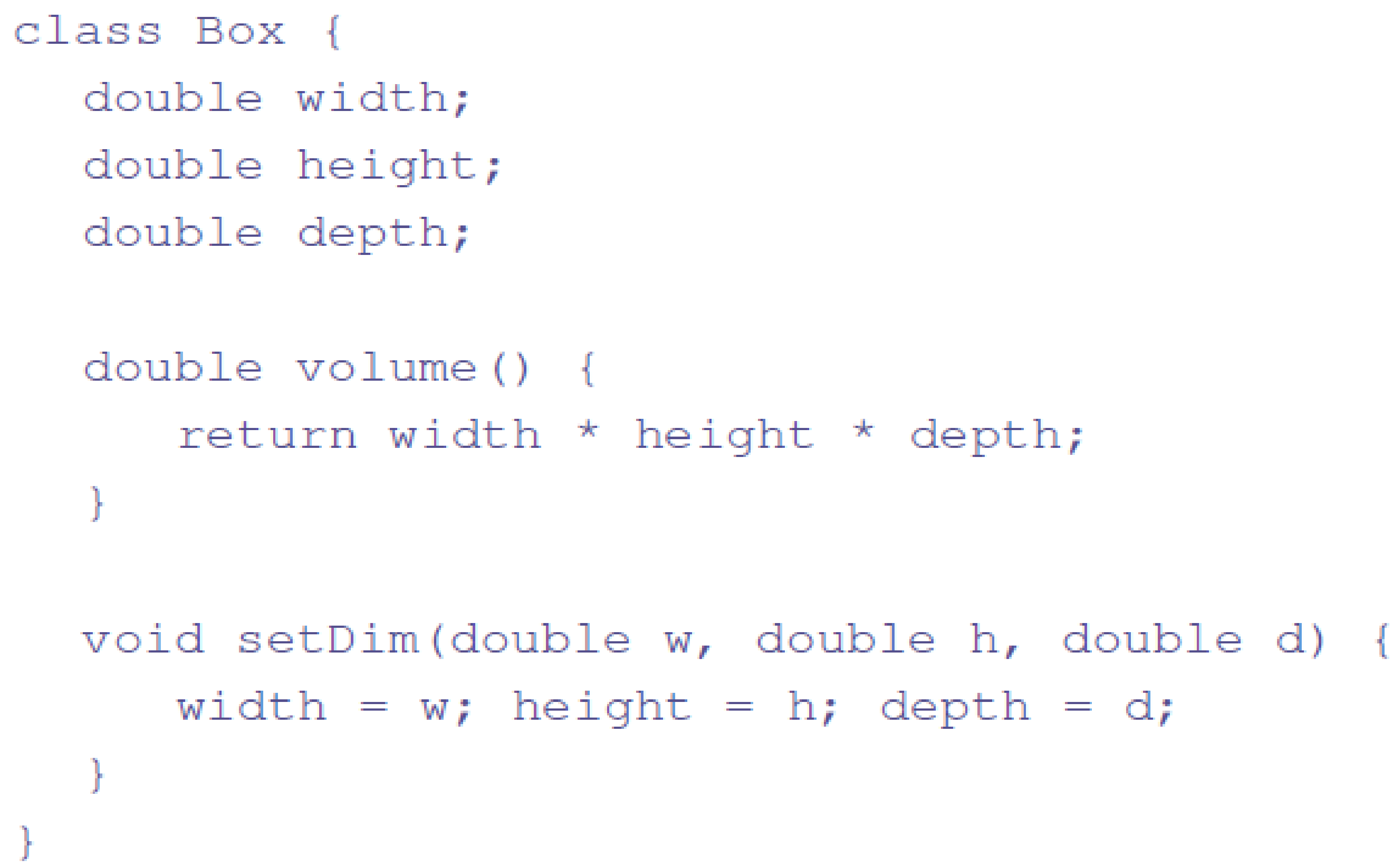
    ```
    int square() { return 10*10; }
    ```

  - method with parameters

    ```
    int square(int i) { return i*i; }
    ```

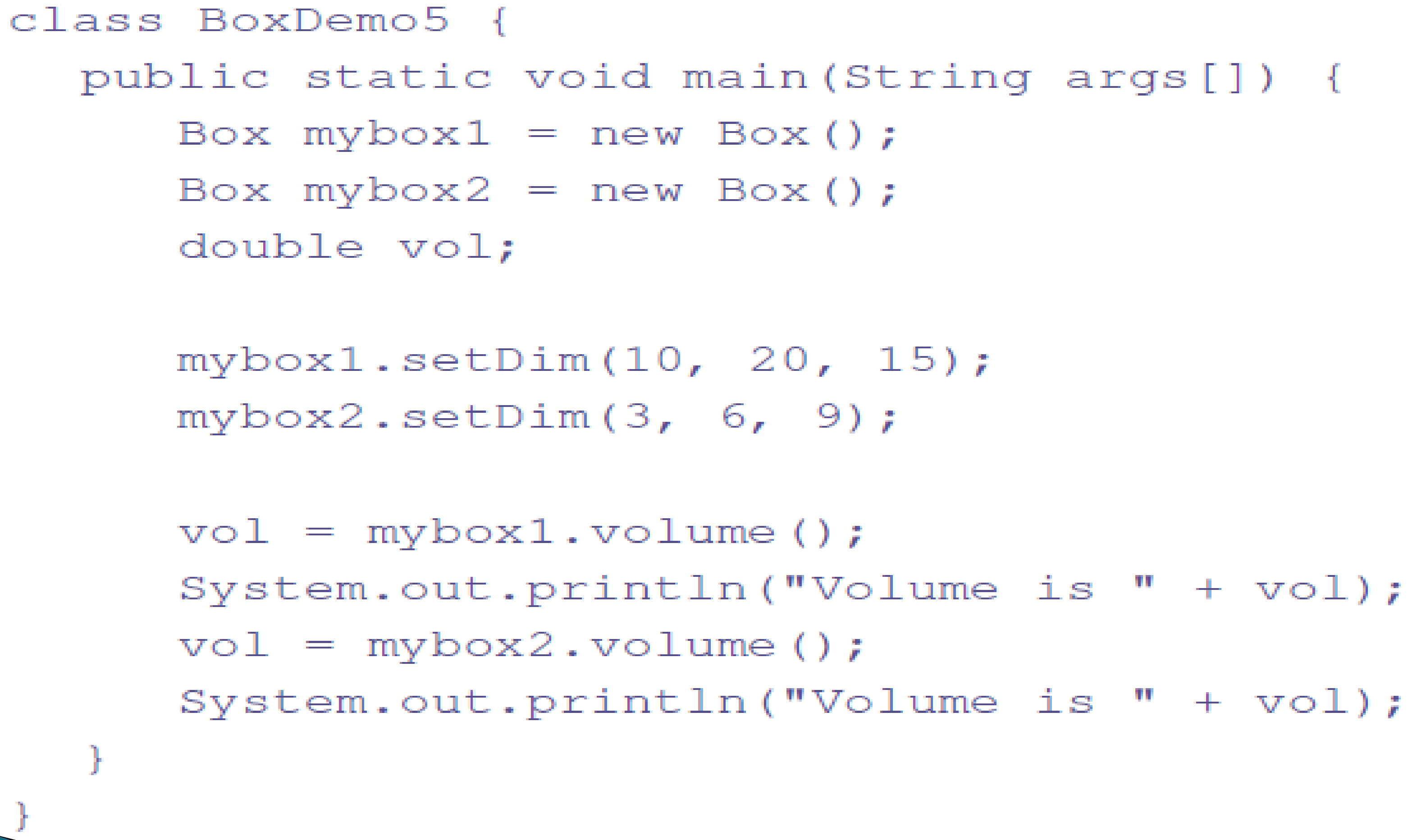
- **Parameter:** a variable receiving value at the time the method is invoked.
- **Argument:** a value passed to the method when it is invoked.

Example: Parameterized Method



```
class Box {
   double width;
   double height;
   double depth;

   double volume() {
       return width * height * depth;
   }

   void setDim(double w, double h, double d) {
       width = w; height = h; depth = d;
   }
}
```

Example: Parameterized Method



```
class BoxDemo5 {
  public static void main(String args[]) {
     Box mybox1 = new Box();
     Box mybox2 = new Box();
     double vol;

     mybox1.setDim(10, 20, 15);
     mybox2.setDim(3, 6, 9);

     vol = mybox1.volume();
     System.out.println("Volume is " + vol);
     vol = mybox2.volume();
     System.out.println("Volume is " + vol);
  }
}
```

```
class Student{
 int rollno;
 String name;
 void insertRecord (int r, String n){
  rollno=r;
  name=n; }
 void displayInformation()
{System.out.println(rollno+" "+name);}}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,“ali");
  s2.insertRecord(222,“aleena");
  s1.displayInformation();
```

```
class Rectangle{
 int length;
 int width;
 void insert(int l, int w){
  length=l;
  width=w; }
 void calculateArea(){System.out.println(length*width);} }
class TestRectangle1{
 public static void main(String args[]){
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
```

Questions?