CHAPTER-4

CLASSES, OBJECTS AND METHODS

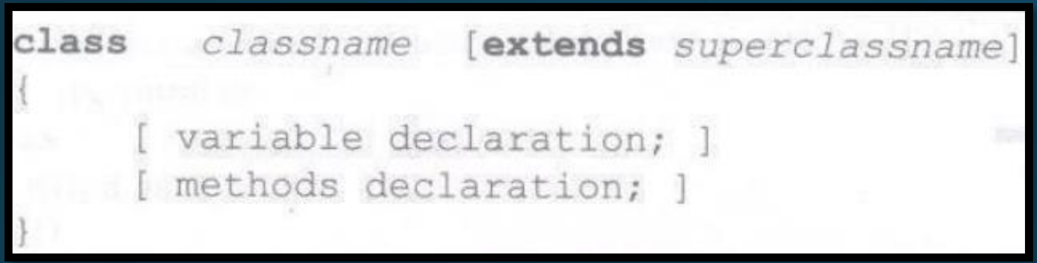
INTRODUCTION

- Java is a true object-oriented language and therefore the underlying structure of all Java programs is classes.
- A class is a blueprint or template for creating objects. It defines the structure and behavior of objects of that type.
- Classes create objects and objects use methods to communicate between them.
- In Java, the data items are called fields and the functions are called methods.
- A class is essentially a description of how to make an object that contains fields and methods.

DEFINING A CLASS :

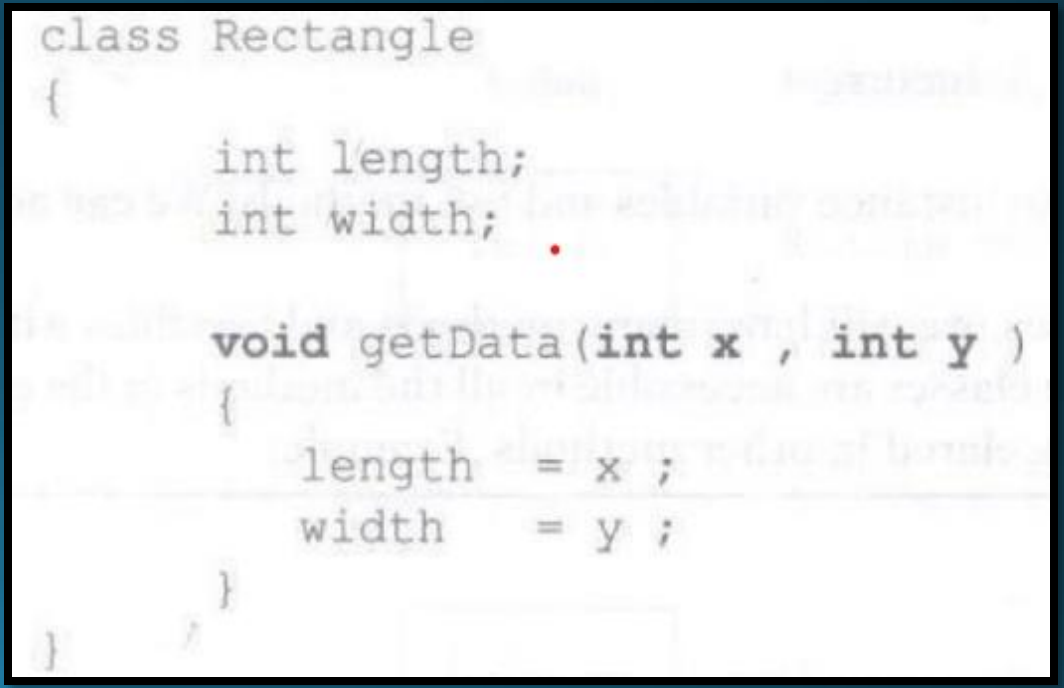
- A class is a user-defined data type with a template that serves to define its properties.
- Once the class type has been defined, we can create "variables" of that type.
- In Java, these variables are termed as instances of classes, which are the actual objects.
- Classes provide a convenient method for packing together a group of logically related data items and functions that work on them.
- In Java, the data items are called **fields** and the functions are called **methods.**

The basic form of a class definition is:



```
class    classname   [extends superclassname]
{
     [ variable declaration; ]
     [ methods declaration; ]
}
```

Example:



```
class Rectangle
{
        int length;
        int width;

        void getData(int x , int y )
        {
            length  = x ;
            width   = y ;
        }
}
```

- **Everything inside the square brackets is optional.**

class Empty

{

}

ADDING VARIABLES

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated.

Example:

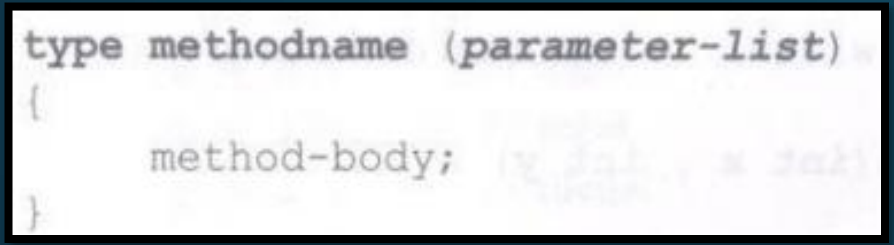
```
class Rectangle    //class Rectangle contains integer type instance variable(length,width)
 {    int width;
      int length;
 }
```

- Instance variables are also known as **member variables.**

ADDING METHODS

A class with only data fields (without methods)has no life. We must add methods that are necessary for manipulating the data contained in the class.

The general form of a method declaration :

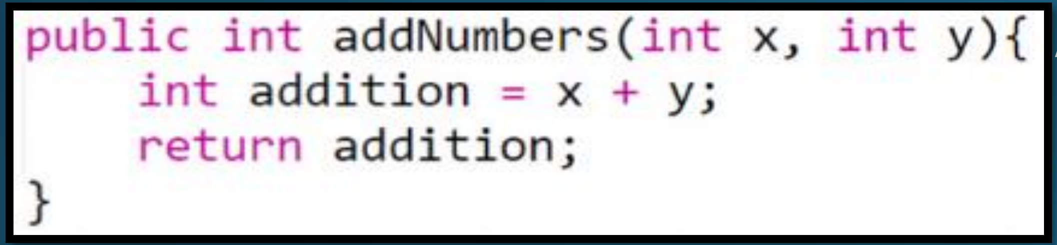


```
type methodname (parameter-list)
{
      method-body;
}
```

Method declarations have four basic parts:

- The name of the method (method name)
- The type of the value the method returns(type)
- A list of parameters(parameter-list)
- The body of the method

Example:



```
public int addNumbers(int x, int y){
    int addition = x + y;
    return addition;
}
```

//The method name is addNumbers , int x and int y are the parameters

//Method Body

CREATING OBJECTS:

- Anything we wish to represent in a Java program must be encapsulated in a class that defines the state and behavior of the basic program components known as objects
- An object in Java is essentially a block of memory that contains space to store all the instance variables.
- Creating an object is also referred to as instantiating an object.
- Objects in Java are created using the **new operator**. The new operator creates an object of the specified class and returns a reference to that object.

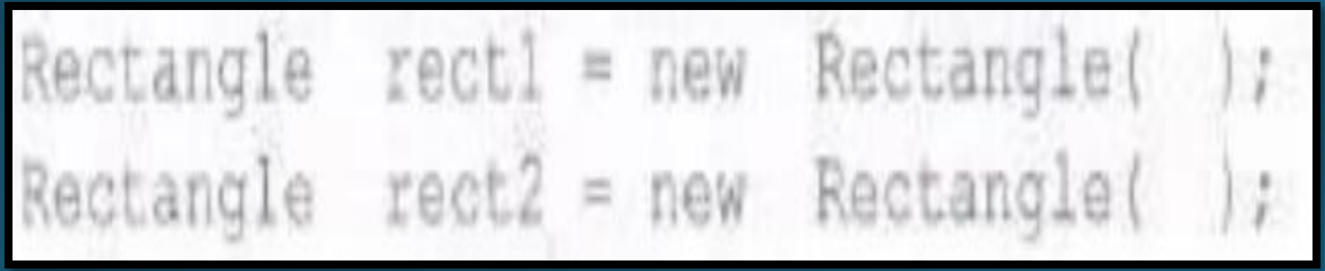
Example of creating an object of type Rectangle.

Rectangle rect1 / / declare

rect1 = new Rectangle ( ) / / instantiate

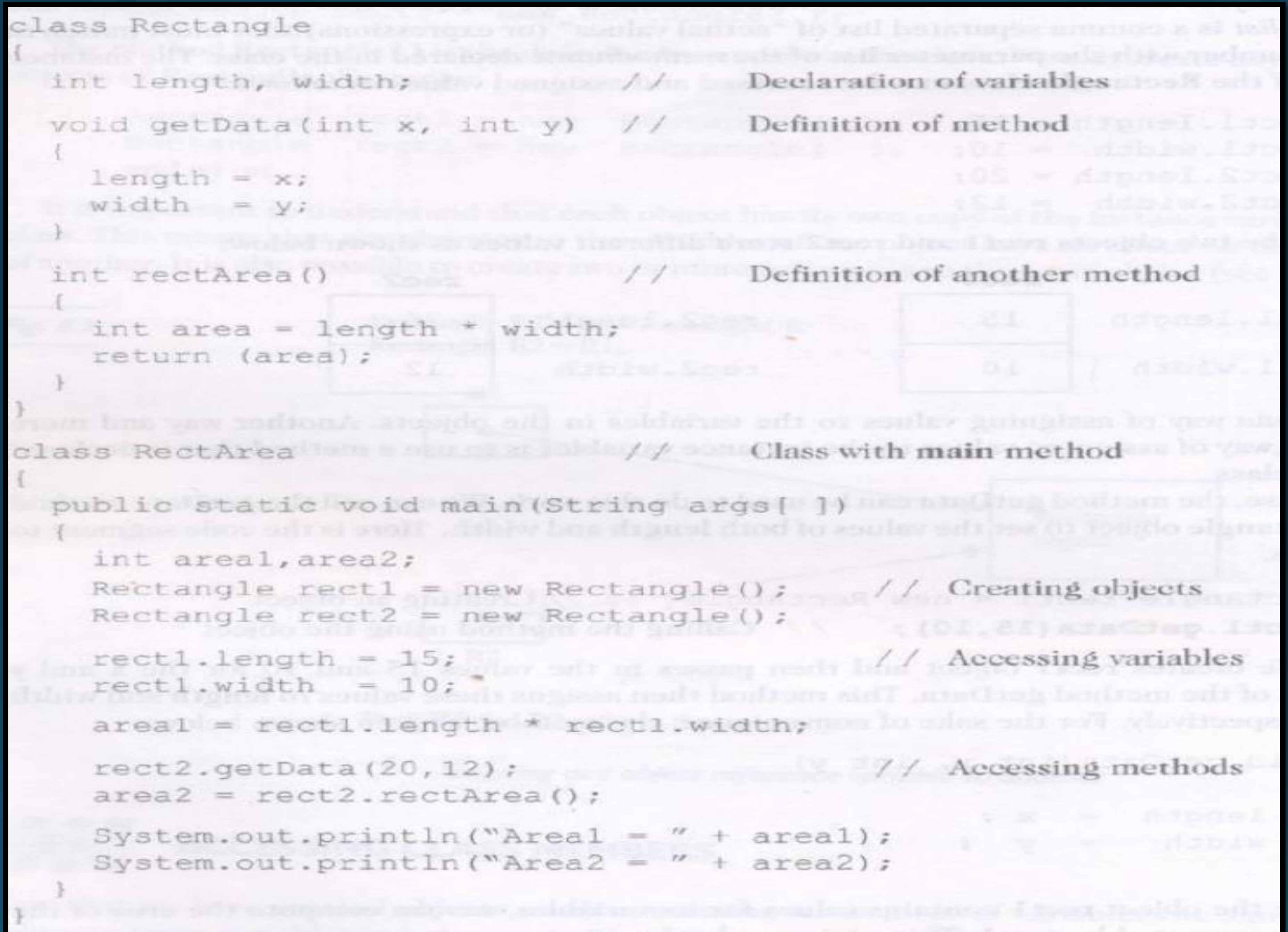
Rectangle rect1 = new Rectangle ( );

The method Rectangle ( ) is the default constructor of the class. We can create any number of objects of Rectangle.



```
Rectangle  rect1 = new  Rectangle(  );
Rectangle  rect2 = new  Rectangle(  );
```

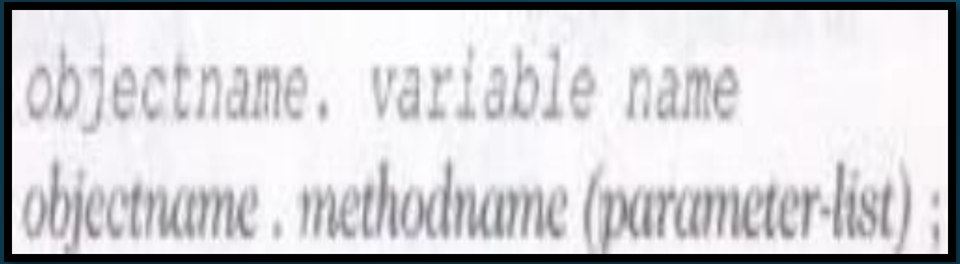
Example:



```
class Rectangle
{
  int length, width;            //  Declaration of variables
  void getData(int x, int y)    //  Definition of method
  {
    length = x;
    width  = y;
  }
  int rectArea()                //  Definition of another method
  {
    int area = length * width;
    return (area);
  }
}
class RectArea                  //  Class with main method
{
  public static void main(String args[ ])
  {
    int area1,area2;
    Rectangle rect1 = new Rectangle();     //  Creating objects
    Rectangle rect2 = new Rectangle();
    rect1.length = 15;                     //  Accessing variables
    rect1.width  = 10;
    area1 = rect1.length * rect1.width;
    rect2.getData(20,12);                  //  Accessing methods
    area2 = rect2.rectArea();
    System.out.println(“Area1 = ” + area1);
    System.out.println(“Area2 = ” + area2);
  }
}
```

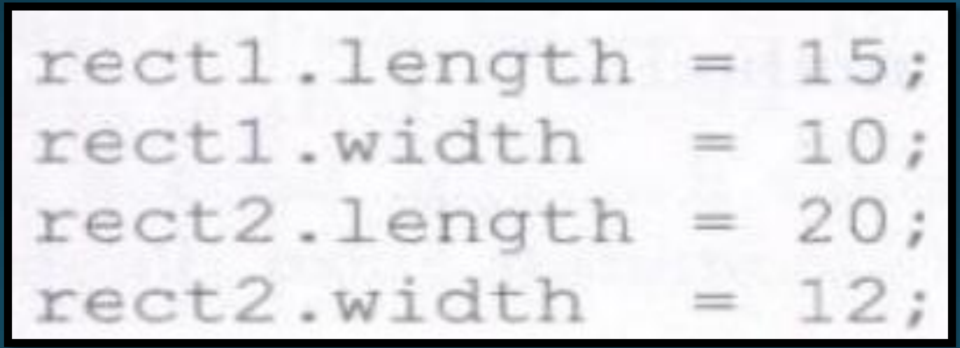
ACCESSING CLASS MEMBERS

- All variables must be assigned values before they are used.
- When accessing instance variables or methods from outside the class definition, we cannot do so directly. Instead, we need to use objects of that class.
- To access instance variables or methods of a class using an object, we use the **dot operator(.)**



```
objectname. variable name
objectname . methodname (parameter-list) ;
```

Example:



```
rect1.length  = 15;
rect1.width   = 10;
rect2.length  = 20;
rect2.width   = 12;
```

CONSTRUCTORS:

All objects that are created must be given initial values.

We can do these using two approaches.

- **First Approach: Direct Assignment Using Dot Operator:**

In this approach, after creating objects, individual instance variables of each object are accessed using the dot operator followed by assignment of values.

It can be a tedious approach to initialize all the variables of all the objects.

Example: **rect1.length = 15;**

```
rect1.length = 15;
rect1.width = 10;
```

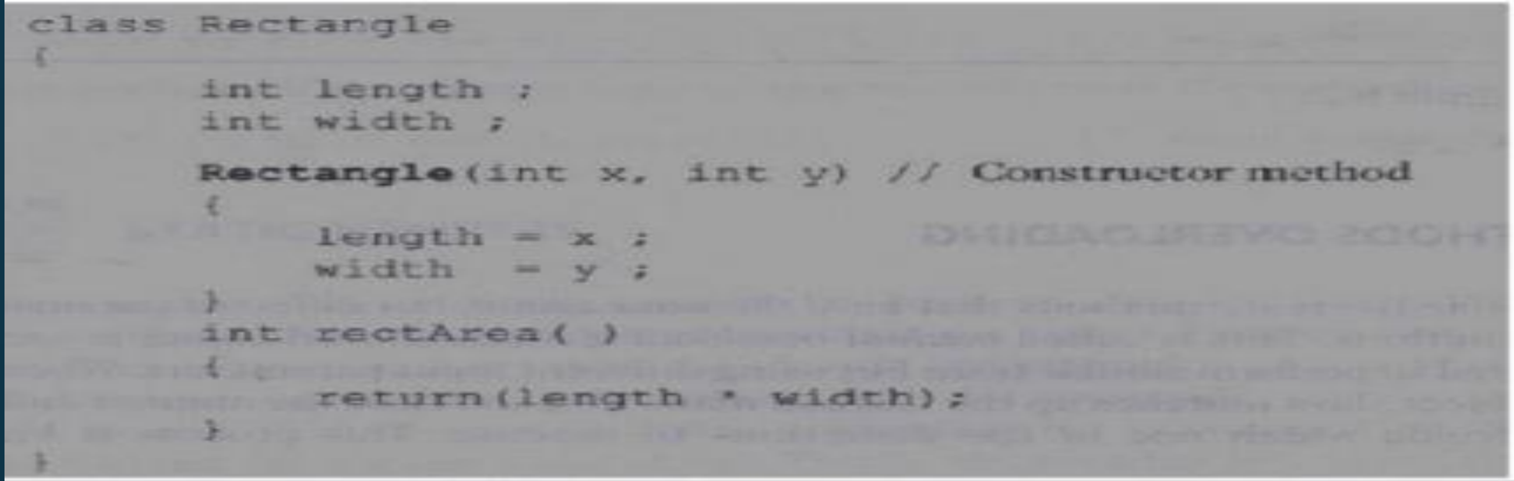
Second Approach: Initialization Method:

Takes the help of a method like **getData** to initialize each object individually using statements like,

rect1.getData (15, l0);

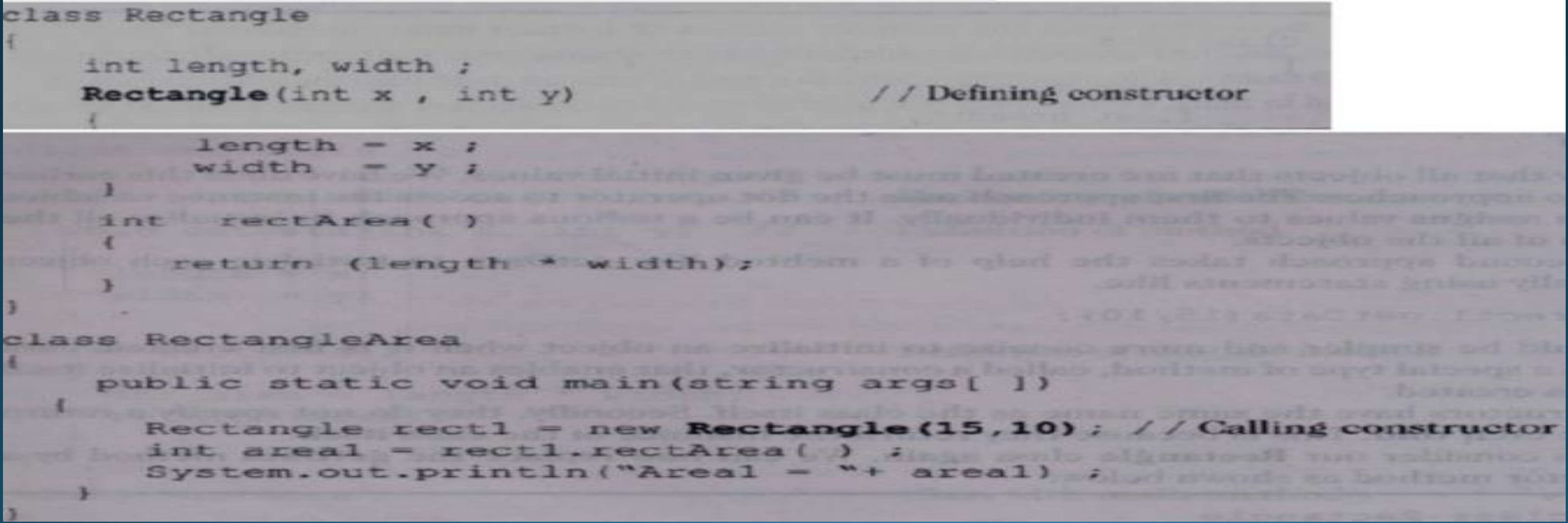
- Constructors have the same name as the class itself.
- They do not specify a return type, not even void.

consider Rectangle as our class .We can replace the getData method by a constructor method as shown below:



```
class Rectangle
{
        int length ;
        int width ;

        Rectangle(int x, int y)  // Constructor method
        {
            length = x ;
            width  = y ;
        }
        int rectArea( )
        {
            return(length * width);
        }
}
```

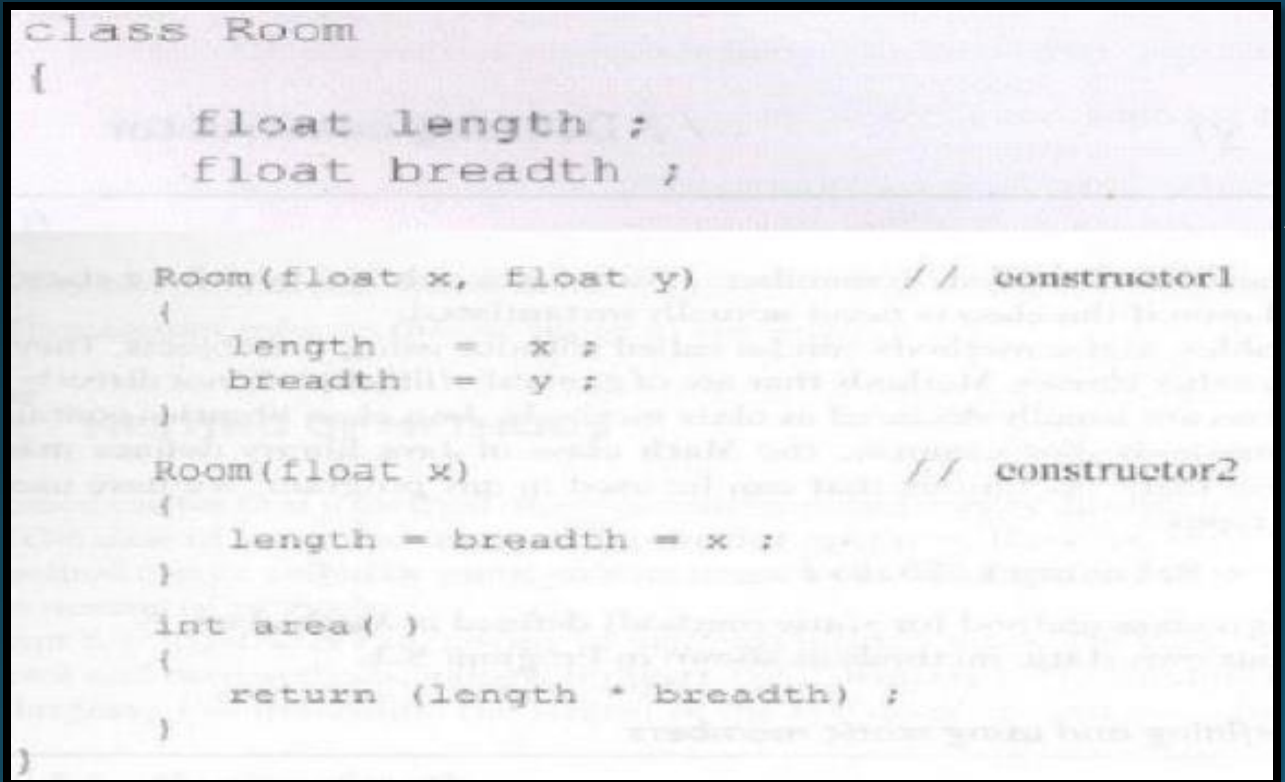


```
class Rectangle
{
     int length, width ;
     Rectangle(int x , int y)                    // Defining constructor
     {
          length = x ;
          width  = y ;
     }
     int  rectArea( )
     {
          return (length * width);
     }
}
class RectangleArea
{
     public static void main(string args[ ])
     {
          Rectangle rect1 = new Rectangle(15,10); // Calling constructor
          int area1 = rect1.rectArea( ) ;
          System.out.println("Area1 = "+ area1) ;
     }
}
```

Method Overloading:

- In Java it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called **method overloading**.
- Method overloading is used when objects are required to perform **similar tasks but using different input parameters.**
- Polymorphism means having many forms. In other words Polymorphism is the ability of a message to be displayed in more than one form.

Example of creating an overloaded method:



```
class Room
{
    float length ;
    float breadth ;

    Room(float x, float y)        // constructor1
    {
        length  = x ;
        breadth = y ;
    }
    Room(float x)                 // constructor2
    {
        length = breadth = x ;
    }
    int area( )
    {
        return (length * breadth) ;
    }
}
```

Here, we are overloading the constructor method Room ().

An object representing a rectangular room will be created as

Room room1 = new Room (25.0, 15.0); //using constructor

On the other hand, if the room is square, then we may create the corresponding object as

Room room2 = new Room (20.0); / / using constructor2

Static Members:

- A class basically contains two sections. One declares variables and the other declares methods. These variables and methods are called **instance variables** and **instance methods**.
- This is because every time the class is instantiated, a new copy of each of them is created. They are accessed using the objects (with dot operator).
- Assume that we want to define a member that is common to all the objects and accessed without using a particular object. That is, the member belongs to the class as a whole rather than the objects created from the class. Such members can be defined as follows:

```
static int count;

static int max(int x, int y);
```

The members that are declared static as shown above are called static members.

- Static variables are used when we want to have a variable common to all instances of a class .
- Like static variables, static methods can be called without using the objects.

For example:

The Math class of Java library defines many static methods to perform math operations that can be used in any program.

For example:

```
float x = Math.sqrt (25.0);
```

- The method sqrt is a class method (or static method) defined in Math class.

Note :

Static methods are called using class names. In fact, no objects have been created for use.

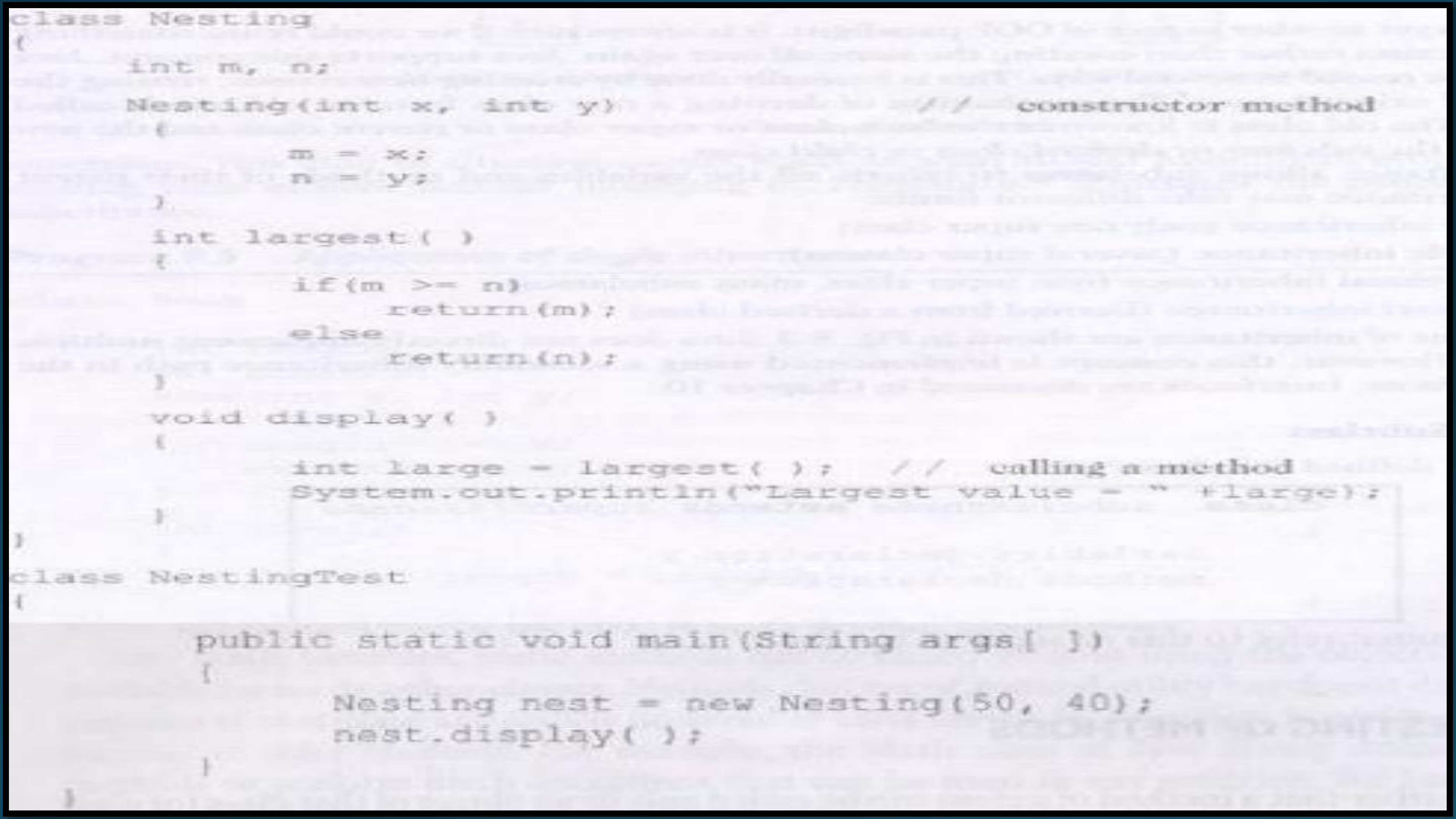
Limitations:

1. They can only call other static methods.
2. They can only access static data.
3. They cannot refer to this or super in anyway

NESTING OF METHODS :

A method of a class can be called only by an object of that class using the dot operator. There is an exception to this. A method can be called by using only its name by another method of the same class. This is known as nesting of methods.

Example:



```
class Nesting
{
        int m, n;
        Nesting(int x, int y)          // constructor method
         {
                m = x;
                n = y;
         }
         int largest( )
         {
                if(m >= n)
                        return(m);
                else
                        return(n);
         }
         void display( )
         {
                int large = largest( );   // calling a method
                System.out.println("Largest value = " +large);
         }
}
class NestingTest
{
        public static void main(String args[ ])
        {
                Nesting nest = new Nesting(50, 40);
                nest.display( );
        }
}
```