# Java Class and Objects

# Class

- The class is at the core of Java.
- Class is that it defines a new data type. Once defined, this new type can be used to create objects of that type.
- Thus, a class is a template for an object, and an object is an instance of a class
- A class is declared by use of the class keyword
- The data, or variables, defined within a class are called instance variables.
- The code is contained within methods.
- Collectively, the methods and variables defined within a class are called members of the class.

```
Class classname {
type instance-variable2; // ... type instance-variableN; type methodname1(parameter-list)
 { // body of method }
 type methodname2(parameter-list)
 { // body of method } // ...
type methodnameN(parameter-list)
 { // body of method }
}
```

- Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables.
- Thus, the data for one object is separate and unique from the data for another.

```
class Student {
    String name;
     int marks;
   void display(){            // method
     System.out.println(name);
     System.out.println(marks);
  }
  float cal_percentage {  // method
    float per;
     per = (marks/300)*100
    return per;
  }
```

# Object

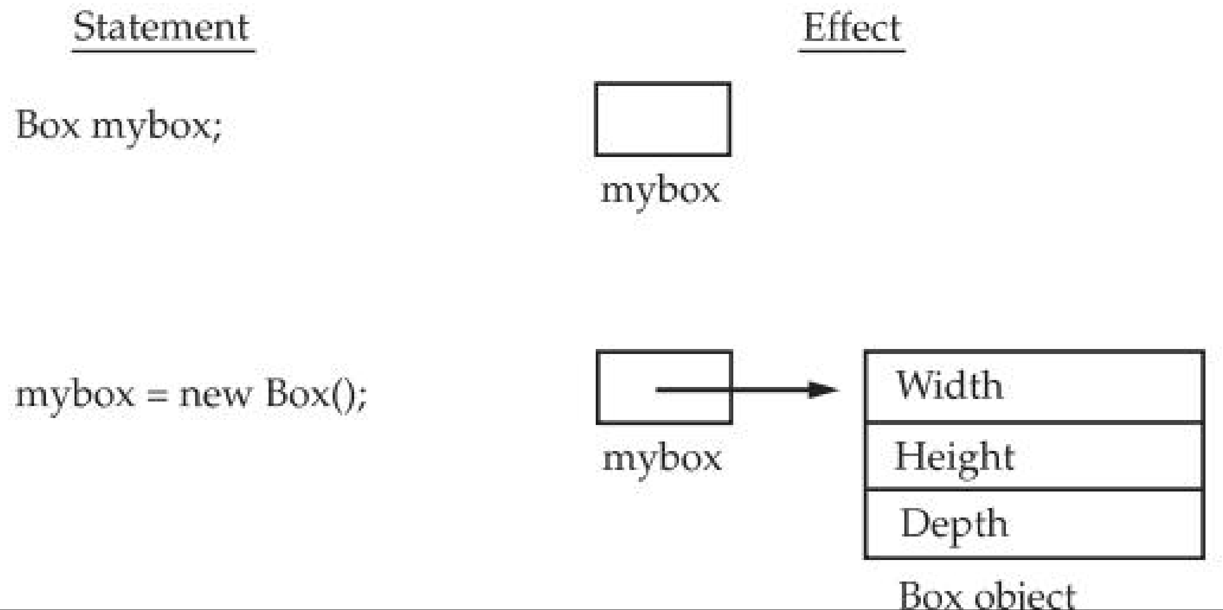
- each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class.
- Every student object will contain its own copies of the instance variables name and marks.
- To access these variables, you will use the dot (.) operator.
- The dot operator links the name of the object with the name of an instance variable.
- Create object with new operator.
- Student s1 = new Student();

s1 is the object of the student

- when you create a class, you are creating a new data type. You can use this type to declare objects of that type.
- obtaining objects of a class is a two-step process. First, you must declare a variable of the class type.
- This variable does not define an object. Instead, it is simply a variable that can refer to an object.
- second, you must acquire an actual, physical copy of the object and assign it to that variable. Using new operator
- The new operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.
- This reference is, more or less, the address in memory of the object allocated by new. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.

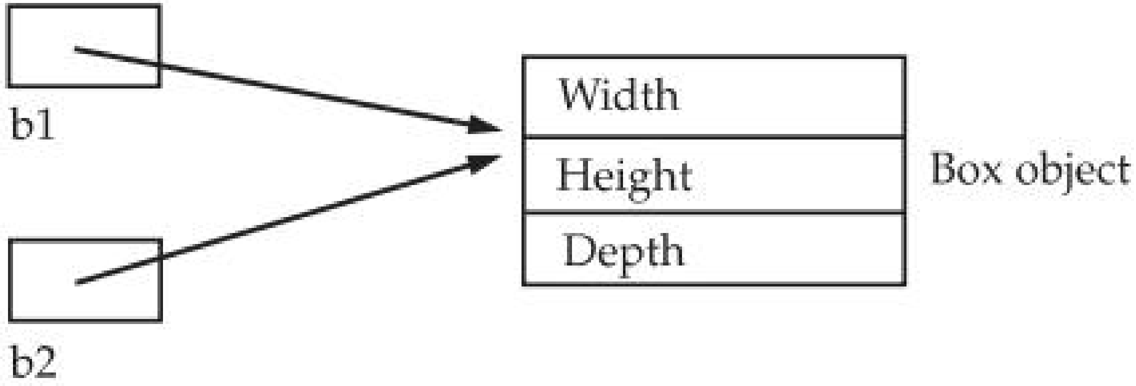
- New Operator
- the new operator dynamically allocates memory for an object.
- class-var = new classname ( );
- Box mybox = new Box(); // create the object dynamically
- Box mybox //declare
- mybox = new Box(); // allocate a Box object

BoxDemo.java



- Java’s primitive types are not implemented as objects. Rather, they are implemented as “normal” variables.
- It is important to understand that new allocates memory for an object during run time.
- The advantage of this approach is that program can create as many or as few objects as it needs during the execution of program.

- Assigning Object Reference Variables
- E.g Box b1 = new Box();
-     Box b2 = b1;
- b1 and b2 will both refer to the same object.
- The assignment of b1 to b2 did not allocate any memory or copy any part of the original object. It simply makes b2 refer to the same object as does b1.
- Thus, any changes made to the object through b2 will affect the object to which b1 is referring,



- Student.java

# Constructor

- Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.
- constructor initializes an object immediately upon creation. It has the same name as the class . Once defined, the constructor is automatically called when the object is created, before the new operator completes.
- Constructors  have no return type, not even void.
- **This is because the implicit return type of a class’ constructor is the class type itself.**

- . When you do not explicitly define a constructor for a class, then Java creates a default constructor for the class
- Box mybox1 = new Box();
- The default constructor automatically initializes all instance variables to their default values
- In program it is allowed to redefine the constructor.

StudentConstructor.java

# Parameterized Constructors

- While the Box( ) constructor in the preceding example does initialize a Box object, it is not very useful—all boxes have the same dimensions. What is needed is a way to construct Box objects of various dimensions. The easy solution is to add parameters to the constructor
- Box(double w, double h, double d)
- { width = w; height = h; depth = d;
- }
- Initialize the object

BoxDemo3.java

Box mybox1 = new Box(10, 20, 15);

StudentConstructor.java

# this Keyword

- To allow this, Java defines the this keyword.
- **this** can be used inside any method to refer to the current object.
- That is, **this** is always a reference to the object on which the method was invoked.
- You can use **this** anywhere a reference to an object of the current class’ type is permitted.
- // A redundant use of this.

```
Box(double width, double height, double depth) {
   this.width = width;
   this.height = height;
   this.depth = depth
```

BoxDemo4.java

```
class Box {
   double width;
   double height;
   double depth;
 // This is the constructor for Box.
 Box(double width, double height,
double depth) {
            this.width = w;
            this.height = h;
            this.depth = d;
 }
 // compute and return volume
            double volume() {
            return width * height * depth;
            }
}
```

```
class BoxDemo4 {
 public static void main(String args[]) {
 // declare, allocate, and initialize Box objects

 Box mybox1 = new Box(10, 20, 15);
 Box mybox2 = new Box(3, 6, 9);

double vol;
 // get volume of first box

   vol = mybox1.volume();
 System.out.println("Volume is " + vol);
 // get volume of second box

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

# Java Copy Constructor

- There is no copy constructor in java. But, we can copy the values of one object to
- another like copy constructor in C++.
- There are many ways to copy the values of one object into another in java. They are:
- By constructor
- By assigning the values of one object into another
- By clone() method of Object class
- 
- StudentCopy.java

# Constructor Overloading

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

```
public class Student
{
    int marks , age;
    String name;

   // Parameterize constructor
   Student(int mrk, String str){
        marks=mrk;
        name = str;
        age=10;
    }
  Student(int mrk, String str , int a){
        marks=mrk;
        name = str;
        age=a;
    }
```

```
void display(){
        System.out.println("student detail");
        System.out.println(“Name :” + name);
        System.out.println(“Marks :” + marks);
        System.out.println(“Age :” +marks);
    }


public static void main(String[] args)
 {
   Student s1 = new Student(170,”XXX”);
   s1.display();
    Student s2 = new Student(170,”XXX”,30);
     s2.display();

 }

}
```

# Difference between constructor and method in java

| Java Constructor | Java Method |
|---|---|
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of the object. |
| Constructor does not have return type | Method have a return type |
| Constructor invoke implicitly | Method invoke explicitly |
| Java compiler provide the default constructor if program does not have constructor | Methods are not provided by the java compiler |
| | |