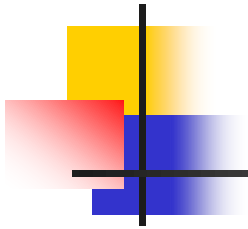
03. Class concepts – (Java)

Class fundamentals
Methods
Constructors
Inner Classes



Class Fundamentals

- What is a Class?
  - A Class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

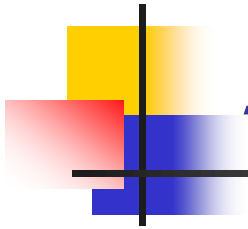
The General Form of a Class

UML Class Diagram

ClassName
List of attributes;
List of behaviors

Java Representation

```
class ClassName {
    type instance-variable;
    type methodName(paramenter-list) {
        // method body.
    }
}
```

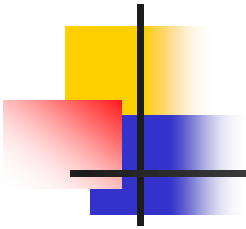


Instance Variable

- Any item of data that is associated with a particular object. Each object has its own copy of the instance variables defined in the class. Also called a field.

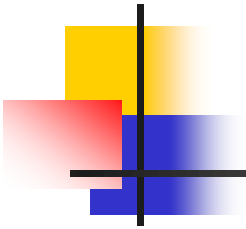
Instance Method

- Any method that is invoked with respect to an instance of a class. Also called simply a method.



class variable

- A data item associated with a particular class as a whole--not with particular instances of the class. Class variables are defined in class definitions. Also called a *static field*.



class method

- A method that is invoked without reference to a particular object. Class methods affect the class as a whole, not a particular instance of the class. Also called a *static method*.

A Simple Class

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

Object Creation - instantiation

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

```
public 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.depth
            * myBox.height;
        System.out.println("Volume is:" +vol);
    }
}
```

This way we can create any number of objects of Box say myBox1, myBox2, myBox3, …

Class Instantiation Statement:

Box myBox = new Box();

Class

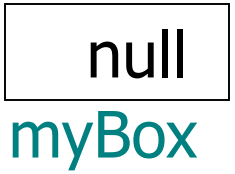
Object

Operator

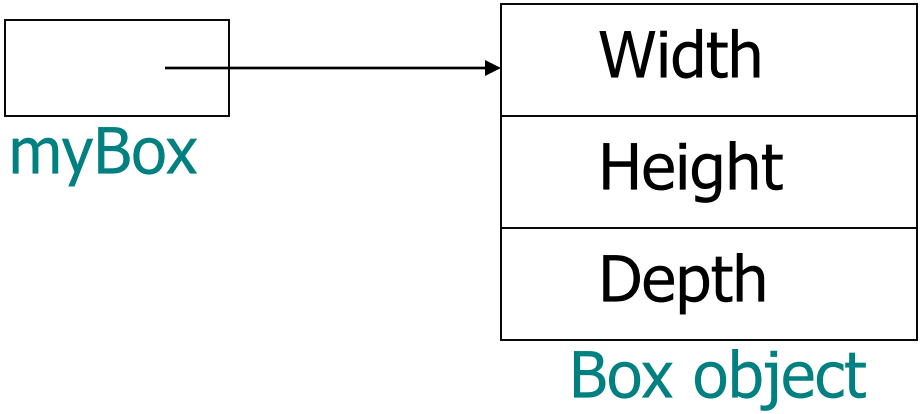
Constructor
(Default)

Instantiation is a 2 step process:

Box myBox;



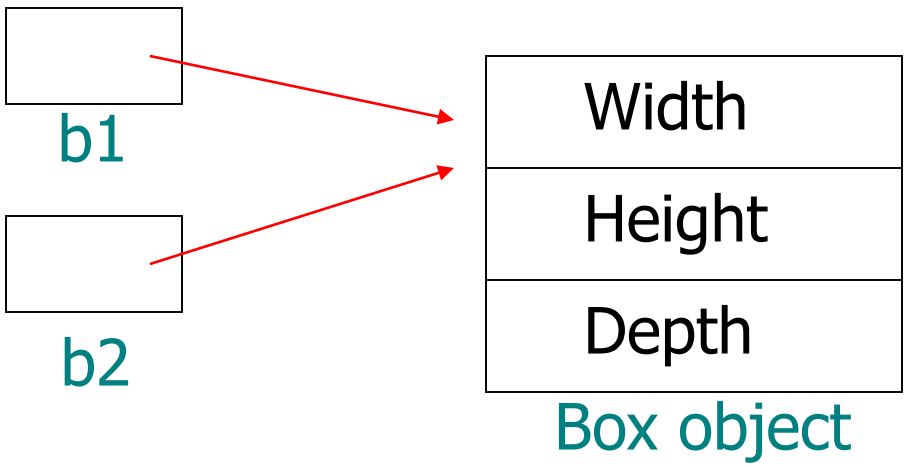
myBox = new Box();



Object Reference Assignment

Box b1 = new Box();

Box b2 = b1;



- Both b1 and b2 refer to the same object and not two distinct objects, but they are not linked in any other way.

Methods

- Method signature:

```
return_type methodName(parameter_list) {
    // method body
}
```

- ***return_type*** specifies the type of data returned by the method. If the method does not return a value, its return type must be ***void***.
- ***methodName*** – any legal identifier other than the keywords.

- ***parameter_list*** – sequence of type and identifier pairs separated by commas.
- Methods that have a return type other than void return a value to the calling routine using a return statement as given below:

  return value;

  Here value is the value returned.

Box Class - Adding a method

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

        //  Instance method.
        void getVolume()  {
                System.out.print("Volume is: ");
                System.out.println(width*height*depth);
        }
}
```

Instance variables

```
public class BoxDemo  {

   public static void main(String args[])  {

        Box myBox1 = new Box();
        Box myBox2 = new Box();

        myBox1.width = 10;
        myBox1.height = 20;
        myBox1.depth = 15;

        myBox2.width = 3;
        myBox2.height = 6;
        myBox2.depth = 9;

        myBox1.getVolume();

        myBox2.getVolume();
   }
}
```

Method returning a value

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

        //  Instance method.
        double getVolume()  {
                double volume = width*height*depth;
                return volume;
        }
}
```

```
public class BoxDemo  {

   public static void main(String args[])  {

        Box myBox1 = new Box();
        double volume;

        myBox1.width = 10;
        myBox1.height = 20;
        myBox1.depth = 15;

        volume = myBox1.getVolume();
        System.out.println("Volume is: " + volume);

   }
}
```

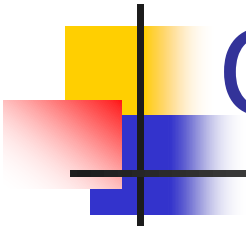
Method that takes a parameter

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

        //  Instance method.
        double getVolume()  {
                return width*height*depth;
        }

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

Formal parameters

```
public class BoxDemo  {

    public static void main(String args[])  {

        Box myBox1 = new Box();
        double volume;

        myBox1.setDim(10, 20, 15);

        volume = myBox1.getVolume();
        System.out.println("Volume is: " + volume);

    }
}
```

Actual parameters



Constructors

- Instead of using a separate method for initializing an object during its creation, it is more convenient and concise to initialize them automatically when they are created.
- This automatic initialization is done by a special method called a ***constructor***.
- A ***constructor*** is special because it does not have a return type, not even void.

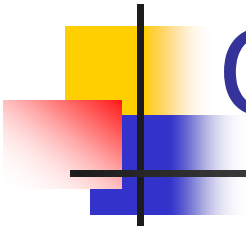
```
class Box  {
        double width;
        double height;
        double depth;

        //  Constructor
        Box()  {
                width = 10;
                height = 10;
                depth = 10;
        }
}
```

Instantiation:

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

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

        //  Constructor
        Box(double w, double h, double d)  {
                width = w;
                height = h;
                depth = d;
        }
        //  Instance method.
        double getVolume()  {
                return width*height*depth;
        }
}
```

```
public class BoxDemo  {

    public static void main(String args[])  {

        Box myBox1 = new Box(10, 20, 15);
        Box myBox2 = new Box(3, 6, 9);
        double volume;

        volume = myBox1.getVolume();
        System.out.println("Volume is: " + volume);

        volume = myBox2.getVolume();
        System.out.println("Volume is: " + volume);

    }
}
```



Constructors

- Instead of using a separate method for initializing an object during its creation, it is more convenient and concise to initialize them automatically when they are created.
- This automatic initialization is done by a special method called a ***constructor***.
- A ***constructor*** is special because it does not have a return type, not even void.

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

        //  Constructor
        Box()  {
                width = 10;
                height = 10;
                depth = 10;
        }
}
```

Instantiation:

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

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

        //  Constructor
        Box(double w, double h, double d)  {
                width = w;
                height = h;
                depth = d;
        }
        //  Instance method.
        double getVolume()  {
                return width*height*depth;
        }
}
```

```
public class BoxDemo  {

    public static void main(String args[])  {

        Box myBox1 = new Box(10, 20, 15);
        Box myBox2 = new Box(3, 6, 9);
        double volume;

        volume = myBox1.getVolume();
        System.out.println("Volume is: " + volume);

        volume = myBox2.getVolume();
        System.out.println("Volume is: " + volume);

    }
}
```

Exercise

Point
-myX : double -myY : double
+getX() : double +getY() : double +Point(x : double, y : double) +Point() +setPoint(x : double, y : double) : void

Exercise

Circle
-blue : int -green : int -origin : Point -radius : double -red : int
+Circle(org : Point, rad : double) +getB() : int +getG() : int +getOrigin() : Point +getR() : int +getRadius() : double +setOrigin(org : Point) : void +setRadius(r : double) : void +setRGB(r : int, g : int, b : int) : void