

```
/* Here, Box defines three constructors to initialize
   the dimensions of a box various ways.
*/
class Box {
    double width;
    double height;
    double depth;

    // constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // constructor used when no dimensions specified
    Box() {
        width = -1; // use -1 to indicate
        height = -1; // an uninitialized
        depth = -1; // box
    }

    // constructor used when cube is created
    Box(double len) {
        width = height = depth = len;
    }

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

```
class OverloadCons {
    public static void main(String args[]) {
        // create boxes using the various constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

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

        // get volume of second box
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);

        // get volume of cube
        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
    }
}
```

Passing Objects as Parameters to Methods

```

// Objects may be passed to methods.
class Test {
    int a, b;

    Test(int i, int j) {
        a = i;
        b = j;
    }

    // return true if o is equal to the invoking object
    boolean equalTo(Test o) {
        if(o.a == a && o.b == b) return true;
        else return false;
    }
}

class PassOb {
    public static void main(String args[]) {
        Test ob1 = new Test(100, 22);
        Test ob2 = new Test(100, 22);
        Test ob3 = new Test(-1, -1);

        System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
        System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
    }
}

```

Also used to copy values of one object to other with the help of constructors

```
// Here, Box allows one object to initialize another.  
  
class Box {  
    double width;  
    double height;  
    double depth;  
  
    // Notice this constructor. It takes an object of type Box.  
    Box(Box ob) { // pass object to constructor  
        width = ob.width;  
        height = ob.height;  
        depth = ob.depth;  
    }  
}
```

A closer look at Argument passing

Pass by value

Pass by reference

Parameter vs Argument

Pass By Value

- If the parameters are primitive types:
 - They are passed by value always in java
 - A copy of those values is given to the method
 - What every operation is performed inside the method over those values, doesn't affect the original value (Copy is passed)



Demo




Pass By Reference

- If the parameters are Object types:
 - They are passed by reference in java
 - Because you pass a reference to memory location
 - Changes made through reference are affected at actual location

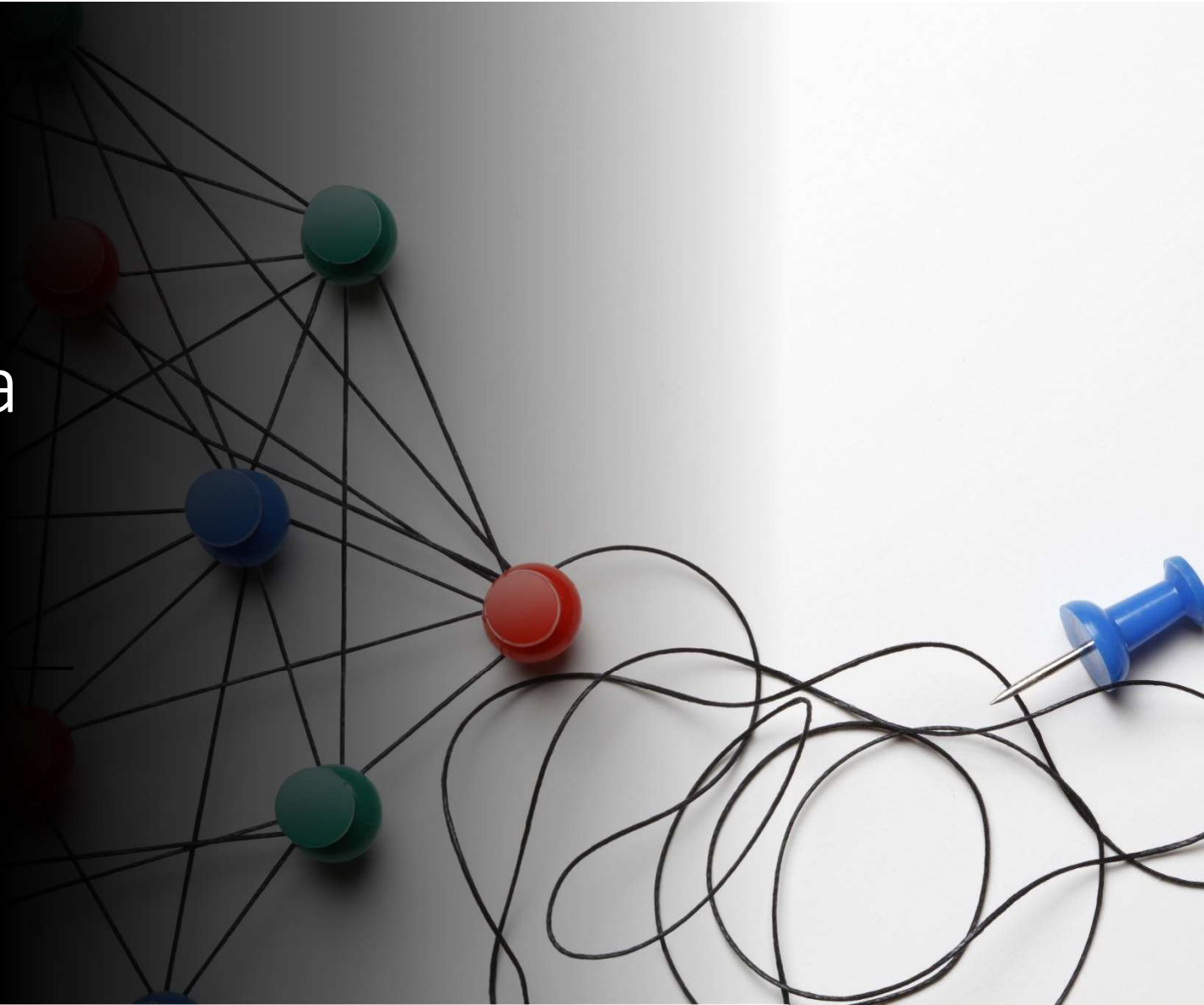


Demo





Returning
objects from a
method as
return type



Recursion

- A process:
 - Method calls itself
 - Recursive Method

Demo

Encapsulation



Process of wrapping up code and data in a single unit

Exp: Capsule mixed from several herbs



We can create a fully encapsulated class in Java by making all the data members of the class private.

How to access and modify the values then?

Getter and Setter Methods

Setter:

- Assign/Modify the values of instance variables
- Also known as Mutator
- Return type is void

Getter:

- Used to retrieve or get the values of instance variables
- Also known as accessor
- Return type is similar to the datatype of instance variable

Why getter/setter, when we have constructor

Constructor just for Initialization

Setter for initializing, as well as modifying/changing the values

Getter for retrieving

Advantages of Encapsulation



Class can be read-only/write-only



It provides you the control over the data:

Value of Marks should be greater than zero

Salary shouldn't be negative



Data Hiding is also achieved, because private member can only be accessed inside the methods of same class



Code becomes more easy to understand and readable

Can we make constructor private?

- Yes you can but on one cost:
 - You can not create the object outside of the class



Demo