OOP with Java

Classes & Objects

Classes and Objects

- Any concept we wish to represent is encapsulated in a class.
- Java class includes instance variables and methods.
- Class defines the shape and behavior of an object
- Class is a template for an object.
- Object is an instance of a class
- A Java program consists of one or more classes

Example:

- **A class is an abstract description of objects**

```
class Car {
        ...description of a car goes here...
}
```

- **some objects of class:**

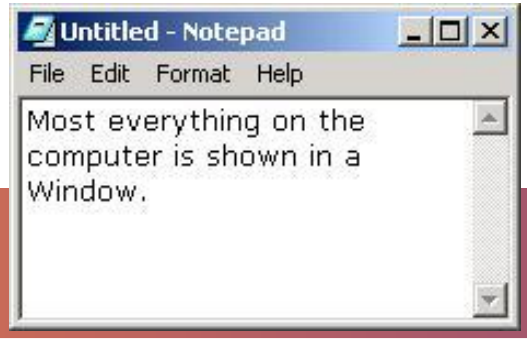


Example:

- Here is another example of a class:

```
class Window { ... }
```

- Some examples of Windows:



Basic Terminology

- **A *class* defines a kind of objects:**
  - ✓**specifies the kinds of *attributes* (*data*) an object can have.**
  - ✓**provides *methods* specifying the *actions* an object can take.**
- **An object is an *instance* of the class.**
- **Person is a class**
  - ✓**Alice and Bob are objects of the Person class.**

What does a class have?

- *Members* of a class*:*
  - ✓***Attributes (instance variables)***

    For each instance of the class (object), values of attributes can vary.
  - ✓***Methods***

- **Person class**
  - ✓**Attributes: name, address, phone number**
  - ✓**Methods: change address, change phone number**
- **Alice object**
  - **Name is Alice, address is …**
- **Bob object**
  - **Name is Bob, address is …**

Example:

```
class ClassName
{       type instanceVariable1;
        // ...
        type instanceVariableN;

        type methodName1(parameter-list)
        {       // body of method
        }
        // ...
        type methodNameN(parameter-list)
        {       // body of method
        }
}
```

Example:

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



```
Box     mybox = new Box();        // create a Box object

mybox.width = 100;
```

Goto BoxDemo.java

Example:

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

Goto BoxDemo.java

```
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;
                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);
  }
}
```

Goto BoxDemo2.java

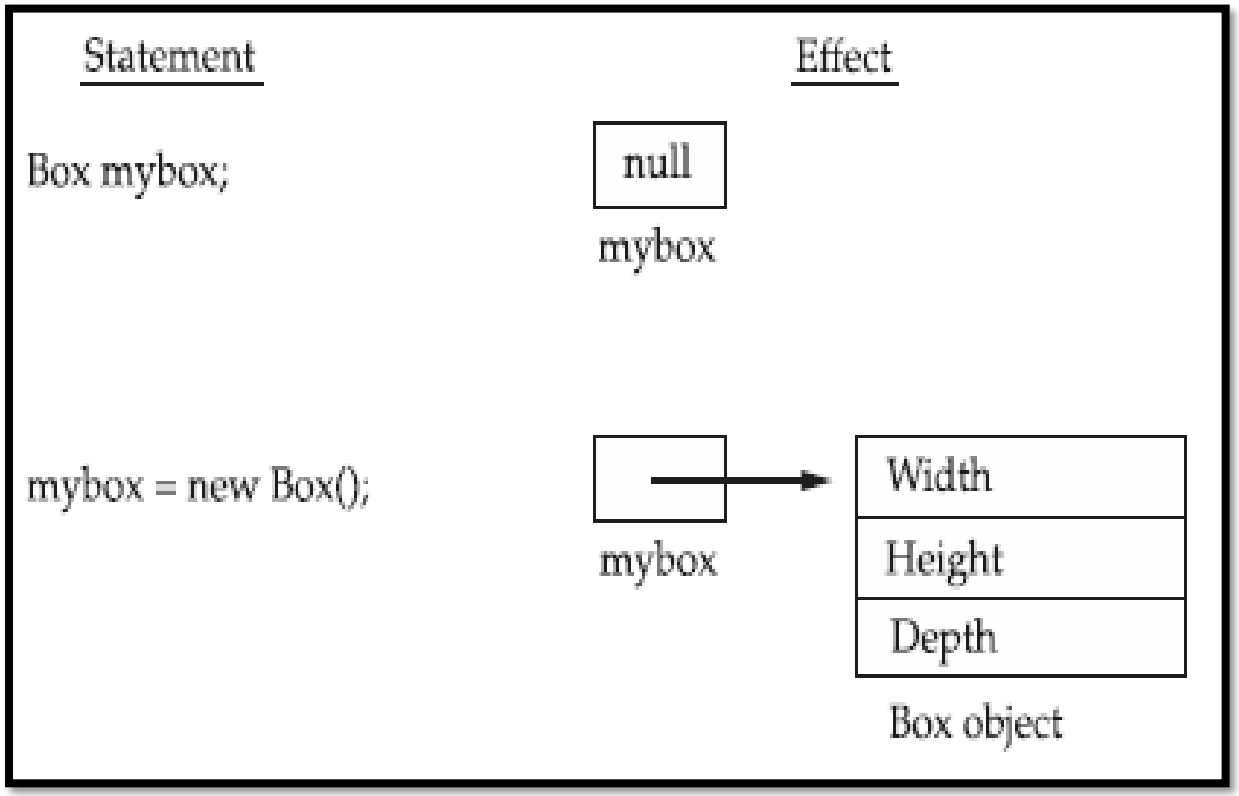
Declaring Objects

Box mybox = new Box();

This statement combines the two steps.

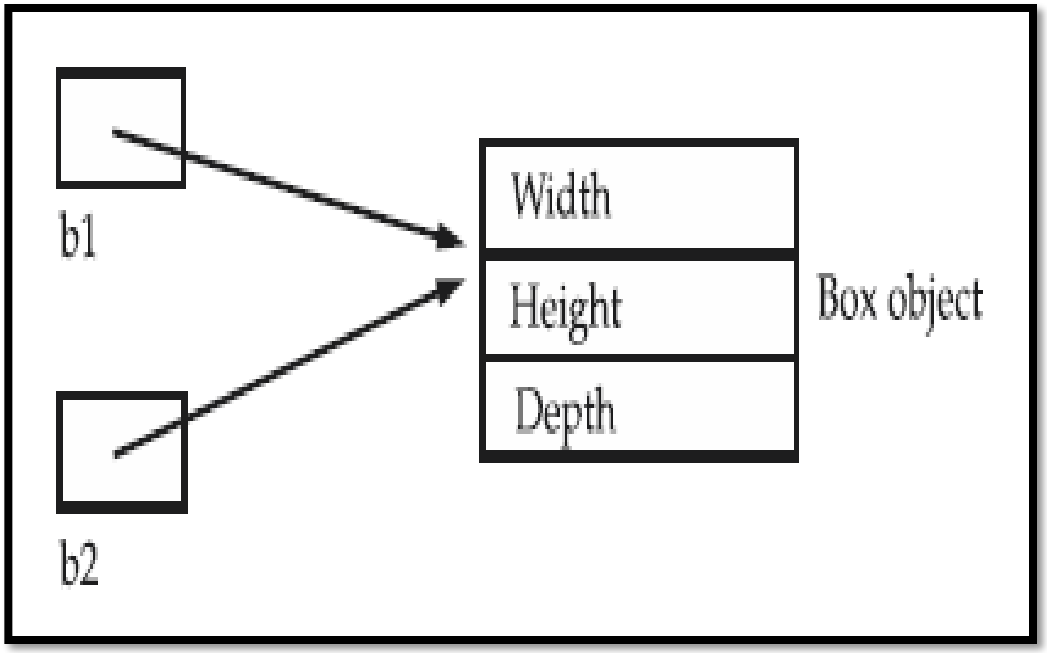
```
Box    mybox;          // declare reference to object
mybox = new Box();     // allocate a Box object
```

"new" operator dynamically allocates memory for an object.



Assigning Object Reference Variables

```
Box b1 = new Box();
Box b2 = b1;
```




```
Box b1 = new Box();
Box b2 = b1;
// ...
b1 = null;
```

* b1 has been set to null, but b2 still points to the original object.

Introducing Methods

```
Class ClassName{
    type  var1;
    type var2;
    .
    .
    type methodName(parameter-list)
     {
         // body of method
     }
}
```

Adding a Method to the Box Class

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

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

Goto BoxDemo3.java

```
class BoxDemo3
{      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.volume();
          mybox2.volume();
   }
 }
```

Goto BoxDemo3.java

Returning value

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

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

Goto BoxDemo4.java

```
class BoxDemo4
{    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;
            mybox2.width = 3;     mybox2.height = 6;      mybox2.depth = 9;

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

Goto BoxDemo4.java

Adding parameterized method

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

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

```
int x, y;
x = square(5);          //x equals to 25
x = square(9);          //x equals to 81
y = 2;
x = square(y);          //x equals to 4
```

```
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;
          }
}
```

Goto BoxDemo5.java

```
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);
        }
}
```

Goto BoxDemo5.java

Constructor

Constructor

- **Constructor is a special type of method.**
- **Constructor has the same name as the class name.**
- **Constructor cannot return values.**
- **Constructor is normally used for initializing objects.**
- **gets invoked “automatically” at the time of object creation.**
- **A class can have more than one constructor.**

```
class Box
{
        double width;   double height;   double depth;
         Box()
         {
              System.out.println("Constructing Box");
               width = 10;    height = 20;   depth = 30;
         }
        double volume()
        {     return width * height * depth;     }
 }
```

Goto BoxDemo6.java

```
class BoxDemo6
 {       public static void main(String args[])
        {
             Box mybox1 = new Box();
             Box mybox2 = new Box();

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

Goto BoxDemo6.java

Parameterized Constructors

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

        Box(double w, double h, double d)
        {         width = w;
                  height = h;
                  depth = d;
        }
        double volume()
        {      return width * height * depth;    }
}
```

Goto BoxDemo7.java

```
class BoxDemo7
{     public static void main(String args[])
      {
            Box mybox1 = new Box(10, 20, 15);
            Box mybox2 = new Box(3, 6, 9);
            double vol;
            vol = mybox1.volume();
            System.out.println("Volume is " + vol);
            vol = mybox2.volume();
           System.out.println("Volume is " + vol);
      }
}
```

Goto BoxDemo7.java

“this” keyword

```
Box(double w, double h, double d)
 {
        this.width = w;
        this.height = h;
        this.depth = d;
}
```

- **this** can be used inside any method to refer to the *current* object.
- **this** is always a reference to the object on which the method was invoked

Instance variable hiding

```
// to resolve name-space collisions.
Box(double width, double height, double depth)
 {
        this.width = width;
        this.height = height;
        this.depth = depth;
}
```

OVERLOADING METHODS

```
class OverloadDemo
 {
       void test()
      {   System.out.println("No parameters");
      }
      void test(int a)
     {    System.out.println("a: " + a);
     }
     void test(int a, int b)
     {    System.out.println("a and b: " + a + " " + b);
     }
     double test(double a)
     {    System.out.println("double a: " + a);
          return a*a;
     }
}
```

```
class Overload
{     public static void main(String args[])
      {
           OverloadDemo ob = new OverloadDemo();
          double result;
          ob.test();
          ob.test(10);
          ob.test(10, 20);
          result = ob.test(123.25);
          System.out.println("Result of ob.test(123.25): " + result);
     }
}
```

```
// Automatic type conversions apply to overloading.
class OverloadDemo
{       void test()
        {       System.out.println("No parameters");
        }
        void test(int a, int b)
        {       System.out.println("a and b: " + a + " " + b);
        }
        void test(double a)
        {       System.out.println("Inside test(double) a: " + a);
        }
}
```

```
class Overload
 {              public static void main(String args[])
        {
                OverloadDemo ob = new OverloadDemo();
                int i = 88;
                ob.test();
                ob.test(10, 20);
                ob.test(i); // this will invoke test(double)
                ob.test(123.2); // this will invoke test(double)
        }
}
```

OVERLOADING CONSTRUCTORS

```
class Box
{       double width;  double height; double depth;
        Box(double w, double h, double d)
        {       width = w;      height = h;      depth = d;      }

         Box()
        {       width = -1; // use -1 to indicate
                height = -1; // an uninitialized
                depth = -1; // box      }
        Box(double len)
        {       width = height = depth = len;   }

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

```
class OverloadCons
{       public static void main(String args[])
        {
                Box mybox1 = new Box(10, 20, 15);
               Box mybox2 = new Box();
               Box mycube = new Box(7);
               double vol;
               vol = mybox1.volume();
               System.out.println("Volume of mybox1 is " + vol);
               vol = mybox2.volume();
               System.out.println("Volume of mybox2 is " + vol);
               vol = mycube.volume();
               System.out.println("Volume of mycube is " + vol);
}
}
```

ACCESS MODIFIERS IN JAVA

Access Modifiers

- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

- There are 4 types of java access modifiers:
  - private
  - default
  - protected
  - public

Private Access Modifier

- The private access modifier is accessible **only within class**.
- Example:

```
class A
{
    private int data=40;
    private void msg()
    System.out.println("Hello java");
}
public class B
{
     public static void main(String args[])
    {
          A obj=new A();
          System.out.println(obj.data);   //Compile Time Error
          obj.msg();    //Compile Time Error
       }
}
```

Private Constructor

- If you make any class constructor private, you ***cannot create the instance of that class from outside the class***.
- Example:

```
class A
{
private A(){}   //private constructor
void msg(){System.out.println("Hello java");}
}
public class B
{
 public static void main(String args[])
{
   A obj=new A();    //Compile Time Error
 }
}
```

- Note: A class cannot be private or protected except nested class.

Default Access Modifier

- If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible ***only within package***.

```
//save by A.java
package pack;
class A
{
void msg(){System.out.println("Hello");}
}
```

++

```
//save by B.java
package mypack;
import pack.*;
class B
{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

- The scope of class A and its method msg() is default so it cannot be accessed from outside the package.

Protected Access Modifier

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor.
- It can't be applied on the class.

```
//save by A.java
package pack;
public class A
{
protected void msg() {System.out.println("Hello");}
}
++++++++++++++++++++++++++++++++++++++++++++++++++
//save by B.java
package mypack;
import pack.*;
 class B extends A
{
  public static void main(String args[])
{
   B obj = new B();
   obj.msg();
  }
}
```

Output:Hello

Public Access Modifier

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.
- **Example of public access modifier**

```
//save by A.java
  package pack;
public class A
{
public void msg()
{System.out.println("Hello");}
}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//save by B.java
package mypack;
import pack.*;
 class B
{
  public static void main(String args[])
{
   A obj = new A();
   obj.msg();
  }
}                                          Output:Hello
```

All Java Access Modifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Access Modifiers with Method Overriding

- If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.
- **Example:**

```
class A
{
protected void msg(){System.out.println("Hello java");}
}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
public class B extends A
{
void msg(){System.out.println("Hello java");}
 public static void main(String args[])
{
   B obj=new B();
   obj.msg();
   }
}
```

- The default modifier is more restrictive than protected. That is why there is compile time error.

Scanner Class

Command-Line Arguments

Passing information into a program during the execution is carried out by passing command line arguments to main().

They are stored as strings in a string array passed to the args parameter of main().

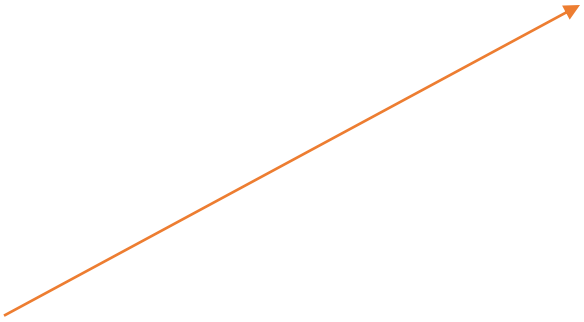
```
//Display all command line arguments
class CommandLine
{
        public static void main(String args[])
        {
                for(int i=0; i< args.length; i++)
                System.out.println("args[" + i + "]: " + args[i]);
        }
}
```

Output:

```
args[0] :this
args[1]:is
args[2]:a
 args[3]:test
args[4]:100
args[5]:-1
```

Program execution at command prompt:

Java Command Line : this is a test 100 -1



Scanner Class

- Reading input from the console enables the program to accept input from the user.

- Predefined classes are organized in the form of packages. This **Scanner** class is found in **java.util** package. So to use the Scanner class, we first need to include java.util package in our program.

Scanner Constructor

Method	Description
Scanner(File *from*) throws FileNotFoundException	Creates a **Scanner** that uses the file specified by *from* as a source for input.
Scanner(File *from*, String *charset*) throws FileNotFoundException	Creates a **Scanner** that uses the file specified by *from* with the encoding specified by *charset* as a source for input.
Scanner(InputStream *from*)	Creates a **Scanner** that uses the stream specified by *from* as a source for input.
Scanner(InputStream *from*, String *charset*)	Creates a **Scanner** that uses the stream specified by *from* with the encoding specified by *charset* as a source for input.
Scanner(Readable *from*)	Creates a **Scanner** that uses the **Readable** object specified by *from* as a source for input.
Scanner (ReadableByteChannel *from*)	Creates a **Scanner** that uses the **ReadableByteChannel** specified by *from* as a source for input.
Scanner(ReadableByteChannel *from*, String *charset*)	Creates a **Scanner** that uses the **ReadableByteChannel** specified by *from* with the encoding specified by *charset* as a source for input.
Scanner(String *from*)	Creates a **Scanner** that uses the string specified by *from* as a source for input.

TABLE 18-14 The **Scanner** Constructors

Scanner Class

- We include a package in a program with the help of **import** keyword. We can either import the **java.util.Scanner** class or the entire **java.util** package.
- To import a class or a package, add one of the following lines to the very beginning of your code.

```
import java.util.Scanner;      // This will import just the Scanner class

import java.util.*;            // This will import the entire java.util package
```

Scanner Class

- After importing, we need to write the following statement in our program.

  Scanner s = new Scanner (System.in);

Here by writing **Scanner s**, we are declaring **s** as an object of **Scanner** class. **System.in** within the round brackets tells Java that this will be System Input i.e. input will be given to the system.

Scanner Class

- Console input is not directly supported in Java, but you can use the Scanner class to create an object to read input from System.in
- The whole line

  Scanner input = new Scanner(System.in)
- (System.in) creates a Scanner object and assigns its reference to the variable input.

TABLE 2.2 Methods for `Scanner` Objects

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

```
import java.util.Scanner;
class ScannerTest
{   public static void main(String args[])
    {    Scanner sc=new Scanner(System.in);
         System.out.println("Enter your rollno");
         int rollno=sc.nextInt();
         System.out.println("Enter your name");
         String name=sc.next();
         System.out.println("Enter your fee");
        double fee=sc.nextDouble();
        System.out.println("Rollno:"+rollno+" name:"+name+"
    fee:"+fee);
     }
}
```