Class and Objects in Java

By
M. BABY ANUSHA,
ASST.PROF IN CSE DEPT.,
RGUKT,NUZVID

OBJECTIVES

- Classes
- Objects
- Methods
- Constructors

CLASS IN JAVA

```
class
One
 {
   }
```

```
class
College
 {
    }
```

```
class
Book
{
  }
```

```
class AnyThing
{
      }
```

CLASS DEFINITION

A class contains a name, several variable declarations (instance variables) and several method declarations. All are called members of the class.

General form of a class:

```
class    classname   {
    type    instance-variable-1;
    …
    type    instance-variable-n;

    type    method-name-1(parameter-list)          {   … }
    type    method-name-2(parameter-list)          {   … }
    …
    type    method-name-m(parameter-list)          {   … }
}
```

EXAMPLE: CLASS

A class with three variable members:

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

A new Box object is created and a new value assigned to its width variable:

```
Box  myBox  =  new  Box();
myBox.width    = 100;
```

EXAMPLE: CLASS USAGE

```
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                          depth; "   +  vol);
    }
}
```

COMPILATION AND EXECUTION

Place the Box class definitions in file Box.java:

```
class   Box  {   … }
```

Place the BoxDemo class definitions in file BoxDemo.java:

```
class   BoxDemo{
    public  static  void main(…)  {   … }
}
```

Compilation and execution:

```
>  javac   BoxDemo.java
>  java  BoxDemo
```

VARIABLE INDEPENDENCE 1

Each object has its own copy of the instance variables: changing the variables of one object has no effect on the variables of another object.

Consider this example:

```
class   BoxDemo2  {
    public  static  void       main(Strin       args[])  {
        Box  mybox1      new Box();
        =  Box  mybox2   new Box();
        double vol;

        mybox1.width     =  10;
        mybox1.           =  20;
        height           = 15;
        mybox1.depth
```

VARIABLE INDEPENDENCE 2

```
        mybox2.width =  3;
        mybox2.height=  6;
        mybox2.depth = 9;

        vol   =  mybox1.width *  mybox1.depth;
        System.out.println("mybox1.height is " + vol);
        Volume

        vol   =  mybox2.width *  mybox2.depth;
    }   System.out.println("mybox2.height is " + vol);
}       Volume
```

What are the printed volumes of both boxes?

OBJECTS

```
class Book
{

}


class JavaBook
{
  public ststic void main(String args[])
        {
          Book b=new Book();
        }
}
```

```
class College
      {

            }


class Kits
{
Public static void
   main(String args[])
{
    College c=new College();
}}
```

b and c are called as objects

DECLARING OBJECTS

Obtaining objects of a class is a two-stage process:

1) Declare a variable of the class type: Box myBox;

   The value of myBox is a reference to an object, if one exists, or null. At this moment, the value of myBox is null.

2) Acquire an actual, physical copy of an object and assign its address to the variable. How to do this?

OPERATOR NEW

Allocates memory for a Box object and returns its address:

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

The address is then stored in the myBox reference variable.

Box() is a class constructor - a class may declare its own constructor or rely on the default constructor provided by the Java environment.

MEMORY ALLOCATION

Memory is allocated for objects dynamically.

This has both advantages and disadvantages:

1) as many objects are created as needed
2) allocation is uncertain – memory may be insufficient

Variables of simple types do not require new: `int n = 1;`

In the interest of efficiency, Java does not implement simple types as objects. Variables of simple types hold values, not references.

ASSIGNING REFERENCE VARIABLES

Assignment copies address, not the actual value:

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

Both variables point to the same object.

Variables are not in any way connected. After

```
b1  = null;
```

b2 still refers to the original object.

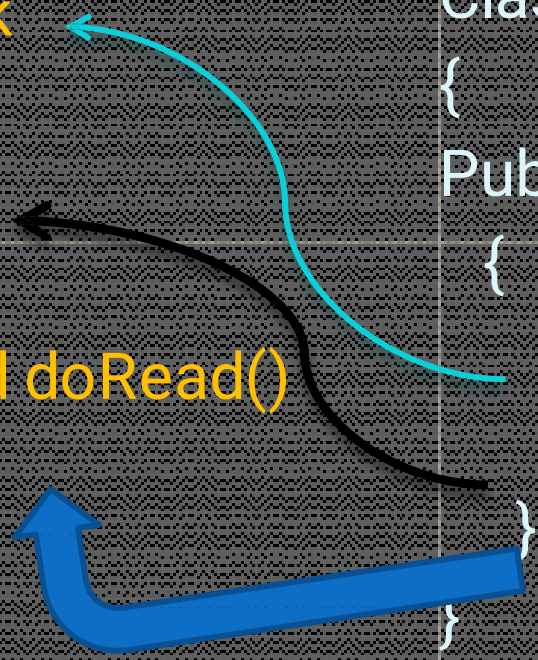
METHODS

```
class Book
{
Int pages;

Public void doRead()
{
}

  }
```

```
Class JavaBook
{
Public static void main(String args[])
  {
    Book b=new Book();  b.
    pages=200;  b.doRead();
  }
}
```



METHODS

General form of a method definition:

```
type name(parameter-list)  {
     … return   value;   …
}
```

Components:

1) type - type of values returned by the method. If a method does not return any value, its return type must be void.
2) name is the name of the method
3) parameter-list is a sequence of type-identifier lists separated by commas
4) return value indicates what value is returned by the method.

EXAMPLE: METHOD 1

Classes declare methods to hide their internal data structures, as well as for their own internal use:

Within a class, we can refer directly to its member variables:

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

EXAMPLE: METHOD 2

When an instance variable is accessed by code that is not part of the class in which that variable is defined, access must be done through an object:

```
Class  BoxDemo3 {
public static  void main(String   args[]) {
            Box  mybox1       new Box();
            = Box  mybox2     new Box();
            = mybox1.         =  10;        mybox2.width =  3;
            width  mybox1.     =  20;       mybox2.              =  6;
            height  mybox1.  = 15;          height               = 9;
            depth                           mybox2.
            mybox1.volume();                depth
            mybox2.volume();
        }
    }
```

VALUE-RETURNING METHOD 1

The type of an expression returning value from a method must agree with the return type of this method:

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

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

VALUE-RETURNING METHOD 2

```
class   BoxDemo4  {
    public  static  void main(String args[]) {
        Box  mybox1       new Box();
        =                 new Box();
        Box mybox2
        =
        double vol; = mybox1.
        width   =  10;
        mybox2.width  =  3;
        mybox1.height=  20;
        mybox2.height=  6;
        mybox1.depth = 15;
        mybox2.depth = 9;
```

VALUE-RETURNING METHOD 3

The type of a variable assigned the value returned by a method must agree with the return type of this method:

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

PARAMETERIZED METHOD

Parameters increase generality and applicability of a method:

1) method without parameters

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

2) method with parameters

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

Parameter: a variable receiving value at the time the method is invoked.

Argument: a value passed to the method when it is invoked.

EXAMPLE: PARAMETERIZED METHOD 1

```
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;
    }
}
```

EXAMPLE: PARAMETERIZED METHOD 2

```
class   BoxDemo5  {
    public  static  void main(String args[]) {
        Box  mybox1       new Box();
        =                 new Box();
        Box mybox2
        double vol;
        =

        mybox1.setDim(10,      20,   15);
        mybox2.setDim(3,      6, 9);

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

CONSTRUCTOR

```
class One
  {
  One()
{
//Initialization
}
    }
```

```
class
College
    {

          }
```

```
class Book
{
  Book()
{
// Some Initialization
}
  }
```

```
class AnyThing
{

        }
```

CONSTRUCTOR

A constructor initializes the instance variables of an object.

It is called immediately after the object is created but before the new operator completes.

1) it is syntactically similar to a method:
2) it has the same name as the name of its class
3) it is written without return type; the default return type of a class constructor is the same class

When the class has no constructor, the default constructor automatically initializes all its instance variables with zero.

EXAMPLE: CONSTRUCTOR 1

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

    Box()  {
        System.out.println("                    Box");
        Constructing                    depth  =  10;
    }   width  = 10; height  =  10;

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

EXAMPLE: CONSTRUCTOR 2

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

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

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

PARAMETERIZED CONSTRUCTOR 1

So far, all boxes have the same dimensions.

We need a constructor able to create boxes with different dimensions:

```
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; }
}
```

PARAMETERIZED CONSTRUCTOR 2

```
class   BoxDemo7  {
    public  static  void main(String              args[]) {
        Box  mybox1       new  Box(10,    20,     15);
        =                 new  Box(3,  6, 9);
        Box mybox2
        double vol;
        =

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

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

Thank you