Introducing Classes (Chapter 6 of Schilit)

Object Oriented Programming BS (CS/SE) II

By

Abdul Haseeb

Class Fundamentals

- Core of Java
- Basic foundation of OOP
- Until now you just saw a class which encapsulates main, to demonstrate basics of java
- Class is a template:
  - Defines a data type
  - Create multiple objects using that data type
- Object is an instance of class

General Form of class

```
class classname {
    type instance-variable1;
    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
    }
}
```

Class Basics

- Contains code and data
- Code and data to gather are called members of a class
- Code organized inside methods
- Variables of class are called Instance variables
  - Because each object will have a different copy of these values

A Simple Class — Code

- Box with only data

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

A Simple Class — Code

- Two Objects of Box

Use dot operator to access and assign instance variables and methods

How many .class files will be created?

Task

- Create a student datatype having id and name and create two instances of it

Declaring Objects

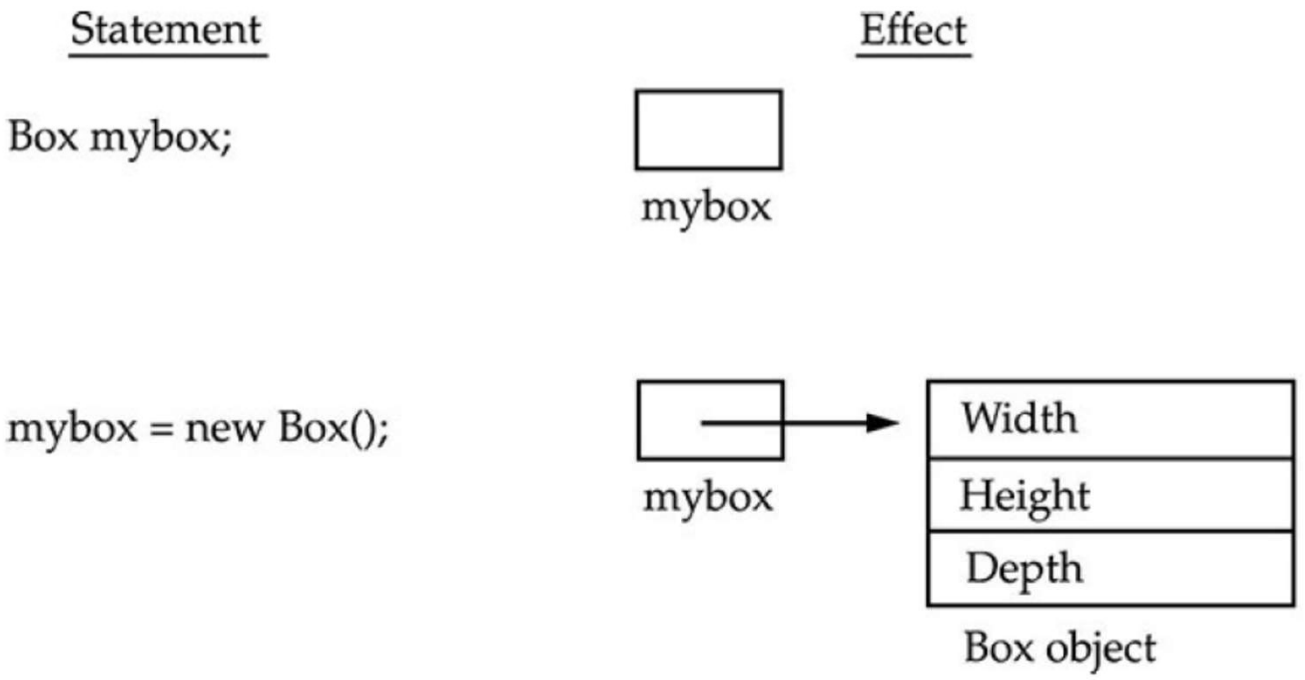


Figure 6-1 Declaring an object of type **Box**

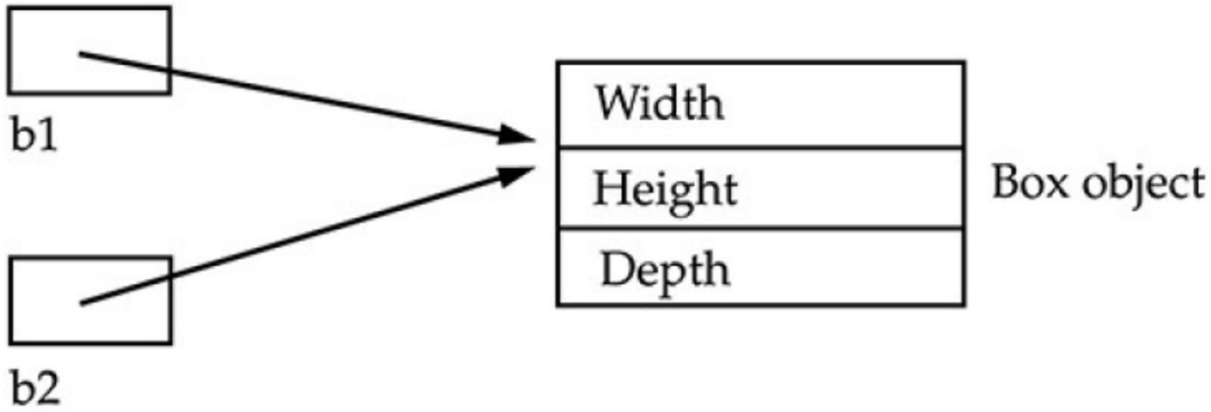
New Operator

- Creates object of a class, and allocates memory to it
- Performs dynamic allocation (at run time)
- Here classname() refers to constructor
  - Specify what should happen when object is created
  - If no constructor is provided, java provides default constructor

- Both point to same object
- Changes in one affect the others

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

Object Reference Variables and use of *null*



Unhooking through the use of null

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

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

Introducing Methods

```
type name(parameter-list) {
    // body of method
}
```

Inserting the volume method to box class

- Call it with box1
- Call it with box2
- You will notice that when we call instance variables inside a method:
  - No Need to use object reference
- When returning a value:
  - Make sure the return data and method return type are compatible:

Use a method which returns some value

Function vs Method

Function	Method
Not associated with objects	Associated with Objects
Invoked through name	Can not invoke by just its name
Independent on class	Dependent on class

Method call vs Method definition

Can We make any class as a private class?

Default value of reference variable?

- Box b1;
- Box b2=new Box()
- Print both reference variables

Where does methods and instance variables get stored?

- Methods get memory from stack:
  - Also the local variables of the method
- Instance variables are stored along with objects in heap

Returning value from method

- Automatic Type Promotion

Method Parameters

- Parameters are useful way of Generalization:
  - Method can work on variety of data

Why this code is not the right choice?

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

Reasons

- Chances to make errors
- Can forget to set a value
- In well designed java programs:
  - Instance variables should be directly accessed by only the methods of a class
- Lets create a setter method

Concept of Constructor

- Why not to initialize the instance variables, when an object is created:
  - Saved from writing setter codes for every object
- Done with the help of constructors(default, user defined):
  - A special method:
    - Used to Initialize the instance variables of an object
    - No return type
    - Same name as the name of class

Types of Constructor

- Default
- Parameterized
- Copy(parameter of constructor is actually an object of Same class)

Some important points

- Java provides a default constructor:
  - Sets the instance variables to default values
- If you provide your own constructor:
  - Default will not work

() indicate call to constructor

```
Box mybox1 = new Box();
```

Lets create a parameterized constructor

- You will see that default constructor will not work then.

Keyword *this* and variable hiding

- this keyword refers to the calling object:
  - Object which has invoked the method

- If we don’t use this keyword:
  - Local variable will hide instance variable and value will be zero

Keyword *this* and variable hiding

- Explore some other of uses of this as well…

Lets do one example:

- Create a setter method having same parameter name and instance variable name
- No use of this
- Print it and see the result

Now try it with this keyword

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

Garbage Collection

- In **Java, garbage collection** is the process of managing memory, automatically. It finds the unused objects (that are no longer used by the program) and delete or remove them to free up the memory.

- Performed automatically in java by Garbage collector, gc() method is called

Garbage Collection

If you want to perform explicitly:

By making a reference null

```
Student student = new Student();
student = null;
```

By assigning a reference to another

```
Student studentOne = new Student();
Student studentTwo = new Student();
studentOne = studentTwo; // now the first object referred by studentOne is available
for garbage collection
```

Stack Animation

- https://yongdanielliang.github.io/animation/web/Stack.html

Code: Stack Class

```
// This class defines an integer stack that can hold 10 values
class Stack {
  int stck[] = new int[10];
  int tos;

  // Initialize top-of-stack
  Stack() {
    tos = -1;
  }

  // Push an item onto the stack
  void push(int item) {
    if(tos==9)
      System.out.println("Stack is full.");
    else
      stck[++tos] = item;
  }
```

Code: Stack Class

```
  // Pop an item from the stack
  int pop() {
    if(tos < 0) {
       System.out.println("Stack underflow.");

       return 0;
    }
    else
      return stck[tos--];
  }
}
```

Code: Stack Class

```
class TestStack {
  public static void main(String args[]) {
    Stack mystack1 = new Stack();
    Stack mystack2 = new Stack();

    // push some numbers onto the stack
    for(int i=0; i<10; i++) mystack1.push(i);
    for(int i=10; i<20; i++) mystack2.push(i);

    // pop those numbers off the stack
    System.out.println("Stack in mystack1:");
    for(int i=0; i<10; i++)
       System.out.println(mystack1.pop());

    System.out.println("Stack in mystack2:");
    for(int i=0; i<10; i++)
       System.out.println(mystack2.pop());
  }
}
```