OOP: Fundamentals

Fundamentals of Java’s OOP Concepts:

Classes & Objects, Constructors & Methods, Access Controls, Inheritance, Polymorphism(Overloading & Overriding), Static , Final, Super, Abstract & Interface.

Md. Mamun Hossain
B.Sc. (Engg.) & M.Sc. (Thesis) in CSE , SUST
Assistant Professor, Dept. of CSE, BAUST

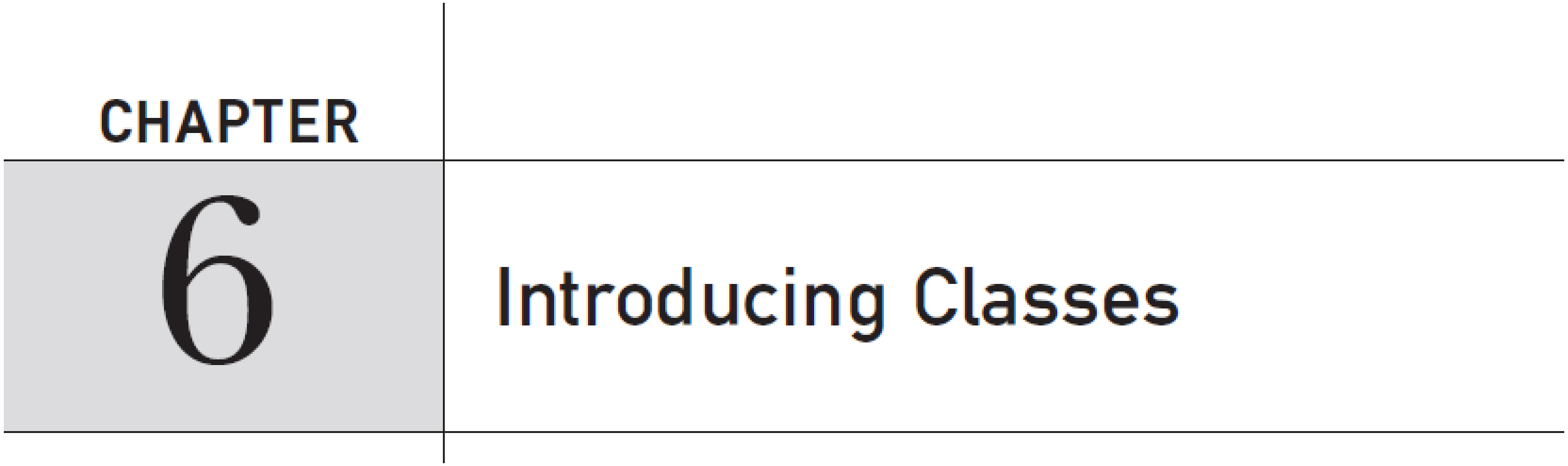
The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

OOP Fundamentals (Chap 6,7&8): Core Concepts

Keywords:

- ✓ **Class, Object: Declaration and Assignment**
- ✓ **Method: Simple& Parameterized,**
- ✓ **Constructor: Default& Parameterized**
- ✓ **The dot(.) operator, initialization of variable: Three ways**
- ✓ ***this* keywords, Garbage Collection**
- ✓ **Method call: Sattic (direct) and Non-static(through object)**
- ✓ **Inheritance: Multiple and Multilevel**
- ✓ **Polymorphism: Overloading & Overriding**
- ✓ **Static, Super, Final, var-arg**
- ✓ **Abstract and Interface**

Referenced Text



CHAPTER

6

Introducing Classes

Java

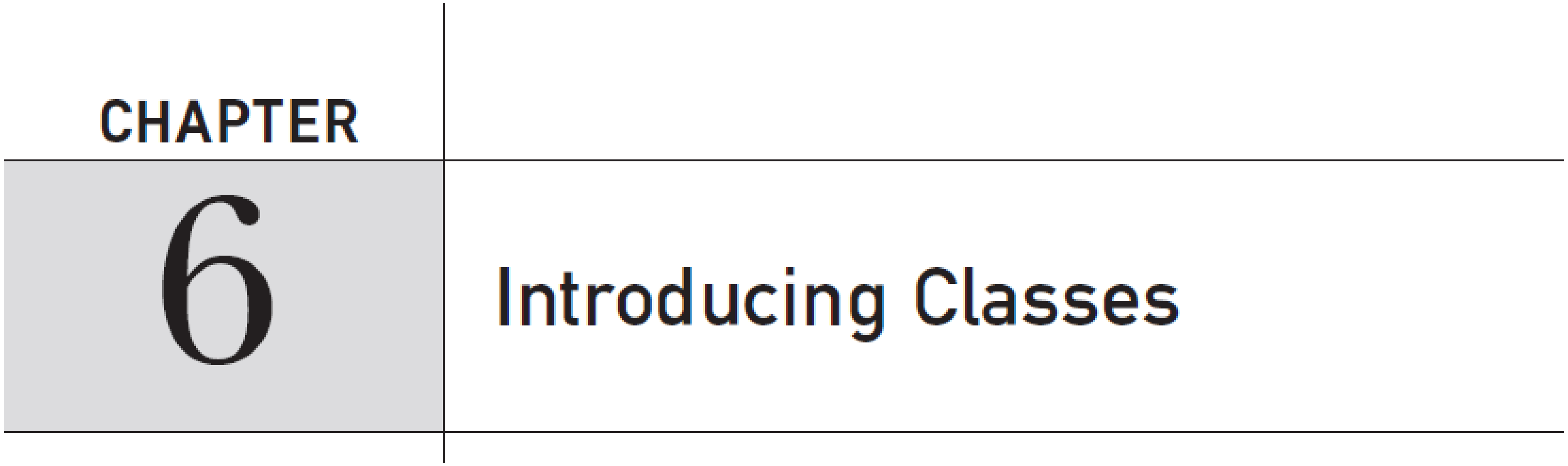
The Complete Reference

Ninth Edition

Comprehensive Coverage of the Java Language

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Object and Class: Contents



CHAPTER

6

Introducing Classes

Keywords:

- ✓ **Class, Object: Class Declaration, Object Creation and assignment**
- ✓ **Method: Simple& Parameterized,**
- ✓ **Constructor: Default& Parameterized**
- ✓ **The dot(.) operator, initialization of variable: Three ways**
- ✓ ***this* keywords, Garbage Collection, The finalize( ) Method**

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java - What is OOP?

OOP stands for **Object-Oriented Programming**.

- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- ✓ OOP is faster and easier to execute
- ✓ OOP provides a clear structure for the programs
- ✓ OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- ✓ OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: *The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.*

OOPs (Object-Oriented Programming System)

- **Object** : real-world entity such as a pen, chair, table, computer, watch, etc.
- **Object-Oriented Programming**: a methodology or paradigm to design a program using classes and objects.
- **The popular object-oriented languages:**
  Java, C#, PHP, Python, C++, etc.

It simplifies software development and maintenance by providing some concepts:

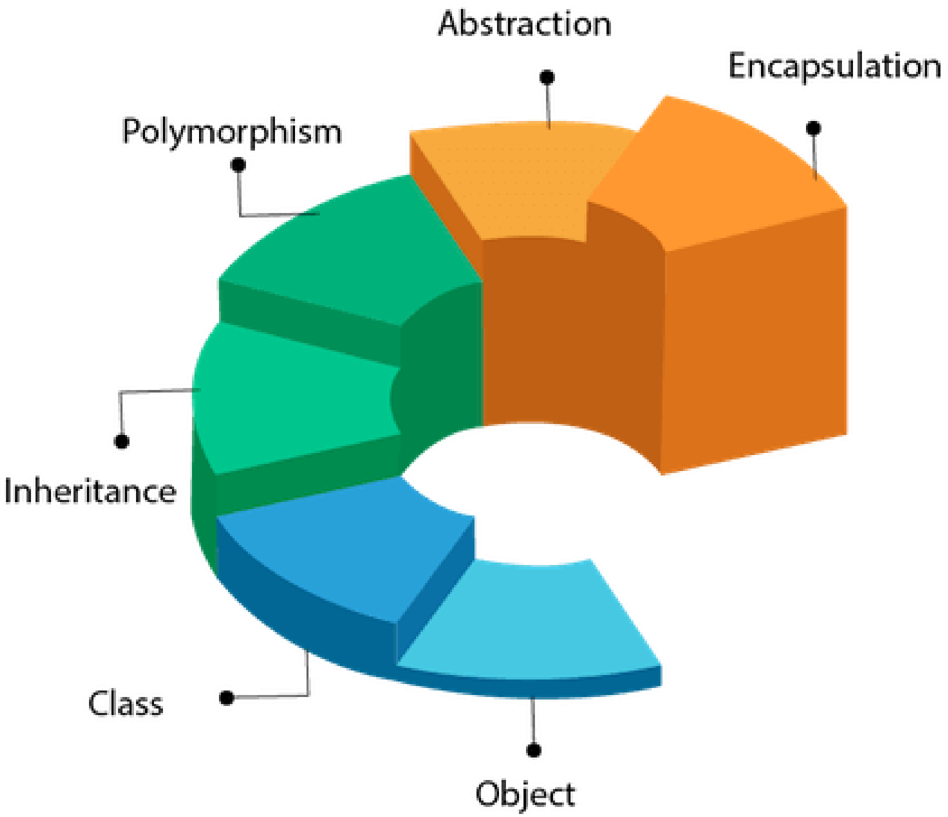
- ✓ **Object & Class,**
- ✓ **Inheritance &Polymorphism,**
- ✓ **Abstraction, & Encapsulation**

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

Coupling, *Cohesion*, *Association*, *Aggregation and Composition*

Java OOP Principal

OOPs (Object-Oriented Programming System)



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java OOP Principal : Object & Class

Object : Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class : Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java OOP Principal : Inheritance & Polymorphism

Inheritance : When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism: If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java OOP Principal : Abstraction& Encapsulation

Abstraction : Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation : Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

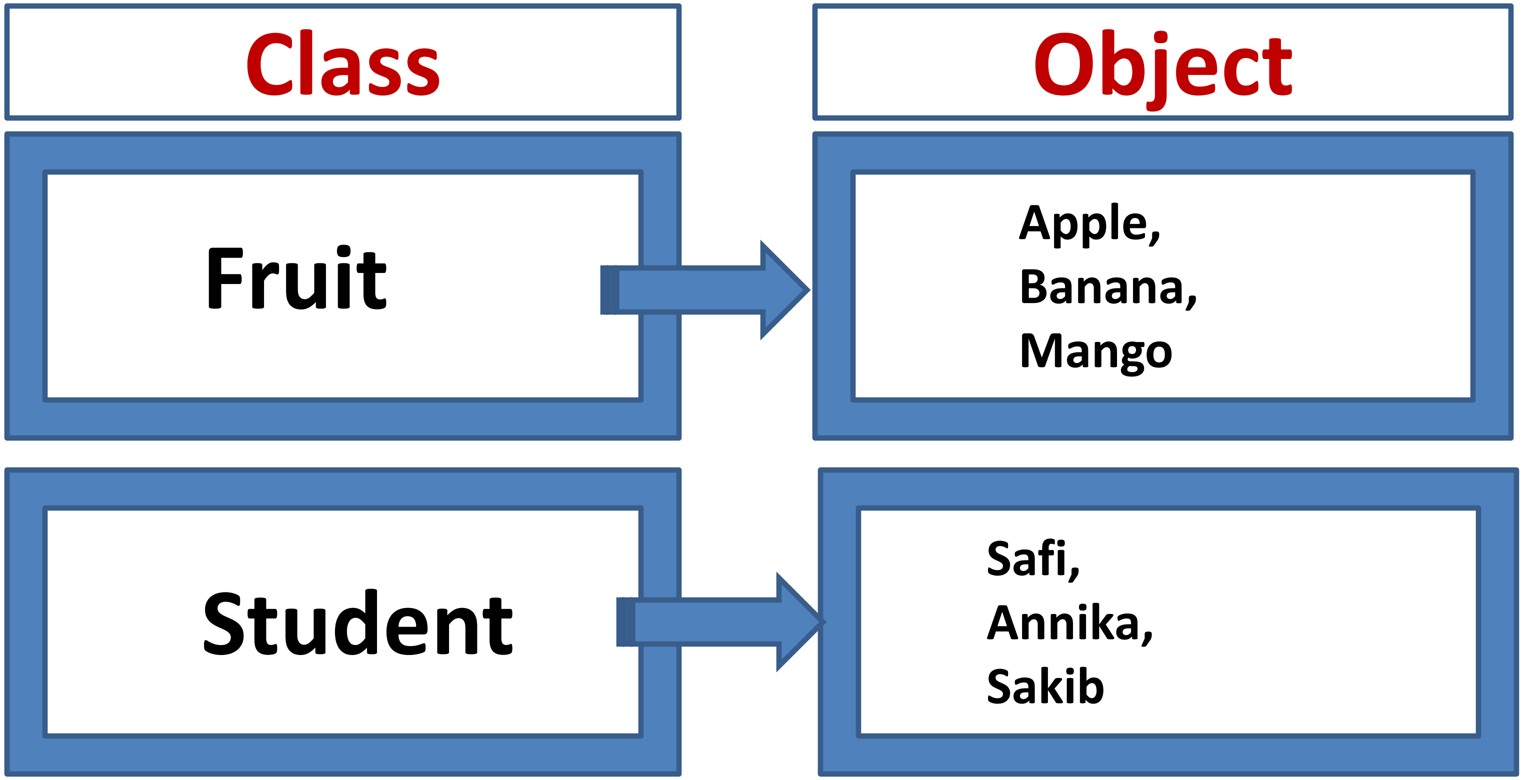
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Java - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:



Class

Object

Fruit

Apple,
Banana,
Mango

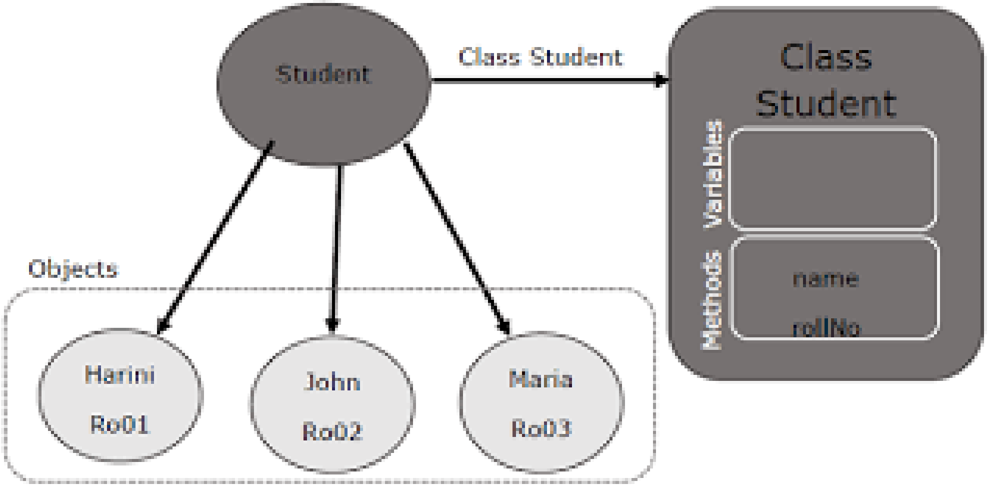
Student

Safi,
Annika,
Sakib

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Object and Class

- class is a *template* for an object
- an object is an *instance* of a class.

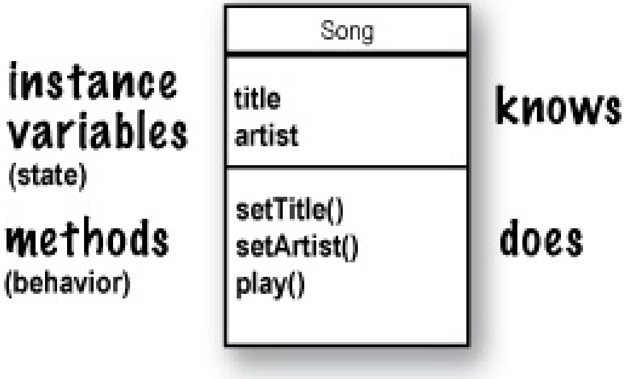


Because an object is an instance of a class, you will often see the two words *object* and *instance* used interchangeably

- When the individual objects are created, they inherit all the variables and methods from the class.

Class Member: Variables and methods

- The data, or variables, defined within a **class** are called *instance variables*.
- Methods operate on data member/variable .The code is contained within *methods*.



- Collectively, the methods and variables defined within a class are called *members* of the class.

Class Members: Example

```
Class Box{

        int height, width;
         double area;

        double Area(){
                area= height*width;
                System.out.println(“Area=“+area);
        }
}
```

height, width and area all are instance variable

Area() is a method

Object: Example

An object is created from a class. We have already created the class named Box, so now we can use this to create objects.

To create an object of Box, specify the class name, followed by the object name, and use the keyword new:

```
Class Box{
            int height, width;
               double area;
            double area(){
                        area= height*width;
            System.out.println("Area="+area);
            }
}
```

```
Box ob1= new Box();
Box ob2= new Box();
Box ob3= new Box();
```

Objects must be within a main function

Object Creation: Two Step Process

Obtaining objects of a class is a two-step process.

- **First,** *you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object.*
- **Second,** *you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the* ***new*** *operator. The* ***new*** *operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.*

Object Creation: Two Step Process

```
Class Box{
        // couple of lines of code
}
```

Step1: `Box mybox; // declare reference to object`

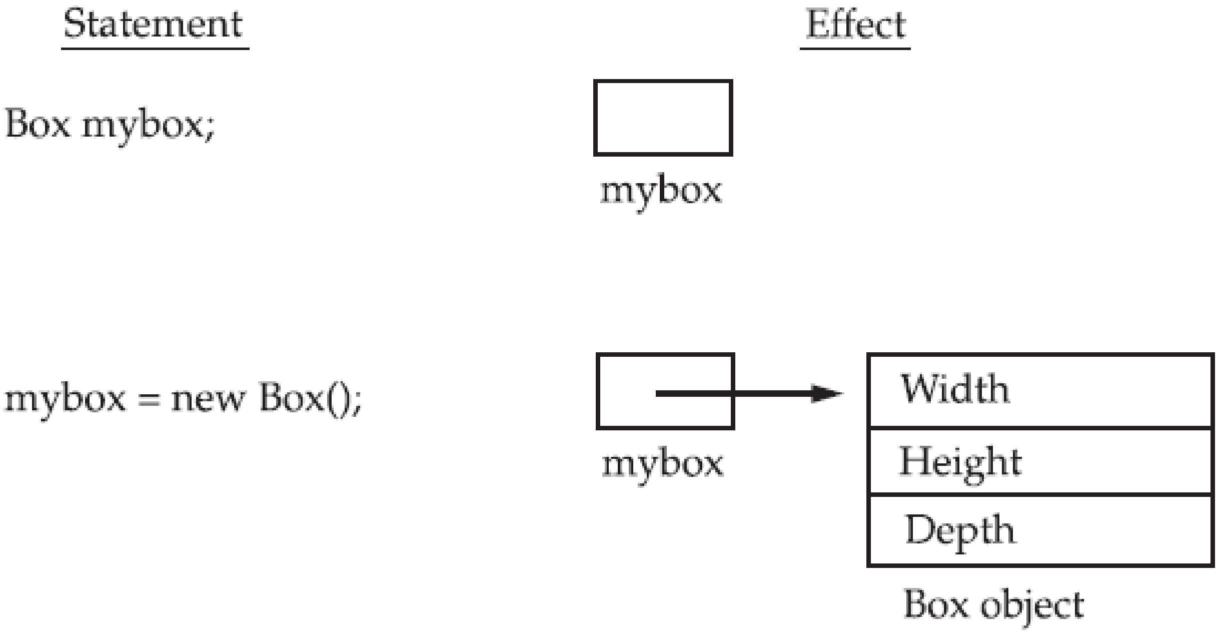
Step2: `mybox = new Box(); // allocate a Box object`

The Following statement combines the two steps just described.

```
        Box mybox = new Box();
```

Declaring an object and effect in memory

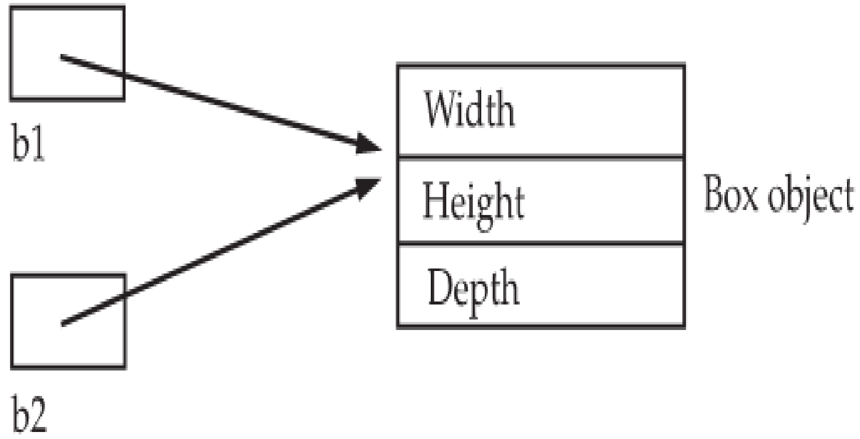
```
class Box{
        int Width;
        int  Height;
        int  Depth;
}
```



Assigning Object Reference Variables

```
class Box{
        int Width;
        int  Height;
        int  Depth;
}

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



Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Object Creation and assignment: An Exercise

Consider the following class declaration in java

```
Class Student{
    int roll;
    char sec;
    byte age;
    static String institution= “BAUST”;
}
```

Now, draw the memory mapping for the following set of instructions

```
Student st1;
st1= new Student ();
Student st2= new Student ();
Student st3=st2;
st1= null;
st3=st2=st1;
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Class Method : Simple and Parameterized

Methods are declared within a class, and that they are used to perform certain actions. Method can be parameterized or Non-parameterized

```
Class Box{

                int height, width;
                    double area;

             // Parameterized Method
                void set_dim(int h, int w){
                                 height=h;
                                width= w;
                }

              // Non-parameterized Method
                void area(){
                                area= height*width;
                                System.out.println(“Area=“+area);
                }
}
```

Parameterized Method

Non-Parameterized Method

Class Constructor : Default and Parameterized

- A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:
- Like method, constructor can be parameterized or Non-parameterized(default)

Default Constructor

```
Class Box{
        int height, width;

        Box(){   }


        Box(int h, int w){
                 height=h;
                width= w;
        }
}
```

Parameterized Constructor

Constructors & Methods : Example

```
class A{

        A(){ … }
        A(String n){  … }

        void show(){ …}
        void show(String n){ … }
        int min_maz(int n){ …}

}
```

Default & Parameterized Constructors

Simple & Parameterized Methods

Class Attribute : Access

You can access attributes by creating an object of the class, and by using the dot syntax (.):

- The following example will create an object of the Box class, with the name Obj. We use the x attribute on the object to print its value:

```
public class Box{
    int x = 5;
    public static void main(String[] args)
    {
          Box Obj= new Box();
          System.out.println(Obj.x);
    }
}
```

initialization of variable: Three ways

1. **Through Dot(.) operator**

```
Ex: Box ob1=new Box()
          ob1.height=2;
          ob1.width=3;
```

2. Through Constructor

```
Ex: Box ob2=new Box(5,6)
```

3. Through method

Ex: Box ob3=new Box()

```
      ob3.set_dim(10,20);
```

Example Class

```
Class Box{
     int height, width;
    Box(){…}
    Box(int h, int w){
          height=h;
          width= w;
     }
    void set_dim(int h, int w){
          height=h;
          width= w;
     }
}
```

Variable initialization : Through Dot (.) Operator

```
class  Box{
                int height;
                int width;
   public static void main(String args[]){
            Box  ob1= new Box();
                ob1.height=10;
                 ob1.weidth=20;
    }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

Variable initialization: Through Constructor

```
Class Box{
            int height, width;
                double area;
            Box(int h, int w){
                         height=h;
                        width= w;
            }
       Public static void main(String args[]){
            Box  ob1= new Box(10,20);
        }
}
```

Variable initialization: Through Method

```
class  Box{
             int height;
             int width;
             void set_dim(int h, int w){
                       height=h;
                      width= w;
             }
     public static void main(String args[]){
           Box  ob1= new Box();
               ob1.set_dim(10,20);
     }
}
```

initialization of variable: Three ways

```
Class Box{
          int height, width;
            double area;
          Box(){         }
          Box(int h, int w){
                    height=h;
                    width= w;
          }

          void set_dim(int h, int w){
                    height=h;
                    width= w;
          }

          void area(){
                    area= height*width;
          System.out.println("Area="+area);
          }
```

```
    Public static void main(String args[]){

    //1. Through Constructor
          Box  ob1= new Box(10,20);
            ob1.area()  // Area=200

    //2. Through dot(.) Operator
          Box  ob2= new Box();
            ob2.height=2
            ob2.width=3
            ob2.area() // Area=6

    //3. Through Method
         Box  ob3= new Box();
           ob3.set_dim(5,10)
           ob3.area()   // Area=50
    }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

initialization of variable: Constructor

```
Class Box{
            int height, width;
                double area;
            Box(){      }
            Box(int h, int w){
                         height=h;
                        width= w;
            }
            void area(){
                        area= height*width;
            System.out.println(“Area=“+area);
            }
```

```
      Public static void main(String args[]){

      //1. Through Constructor
             Box  ob1= new Box(10,20);
                ob1.area()  // Area=200

      //2. Through Constructor
             Box  ob2= new Box(5,10);
                ob2.area()  // Area=50

      //3. Through Constructor
            Box  ob3= new Box(2,3);
                ob3.area()  // Area=6}
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

this keyword: *this* Pointer

- ✓ 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.

```
class Box{
int width, height
void set_dim(double w, double h) {
      this.width = w;
      height = h;
   }
}
```

```
class Access{
              public static void main(String args[]){
                            Box ob1=new Box()
                                  ob1. set_dim(10,20)
                             Box ob2=new Box()
                                  ob2. set_dim(100,200)
               }
              }
```

this keyword: A redundant use of this

```
class Box{
int width, height
void set_dim(double w, double h) {
     this.width = w;
     this.height = h;
   }
}
```

```
class Access{
           public static void main(String args[]){
                      Box ob1=new Box()
                           ob1. set_dim(10,20)
                       Box ob2=new Box()
                           ob2. set_dim(100,200)
           }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

this keyword : name-space collision

```
class Box{
int width, height
void set_dim(double width, double height) {
        width = width;
        height = height ;
    }
}
```

```
class Access{
                public static void main(String args[]){
                                Box ob1=new Box()
                                        ob1. set_dim(10,20)
                                 Box ob2=new Box()
                                        ob2. set_dim(100,200)
                }
}
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

this keyword: Solution of name-space collisions.

Use ***this*** to resolve name-space collisions

```
class Box{
int width, height
void Box(double width, double height) {
      this.width = width;
      this.height = height ;
   }
}
```

```
class Access{
              public static void main(String args[]){
                           Box ob1=new Box()
                                 ob1. set_dim(10,20)
                            Box ob2=new Box()
                                 ob2. set_dim(10,20)
              }
}
```

Java Garbage Collection

- Since objects are dynamically allocated by using the **new** operator, you might be wondering how such objects are destroyed and their memory released for later reallocation.
- In some languages, such as C++, dynamically allocated objects must be manually released by use of a **delete** operator.

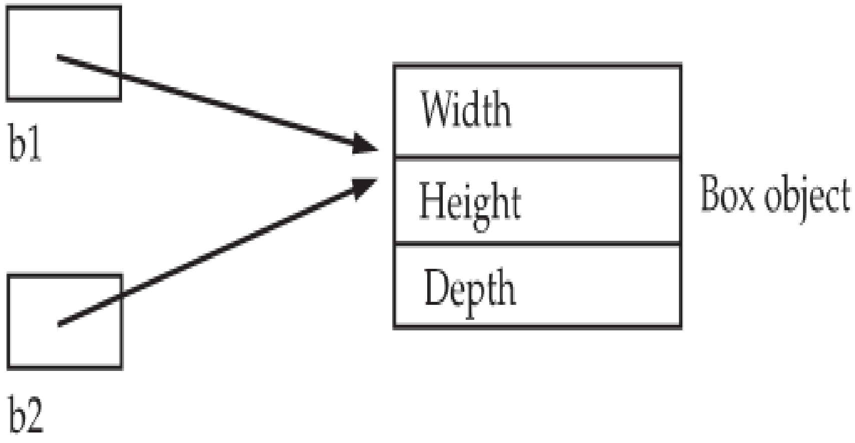
Java takes a different approach; it handles de-allocation for you ***automatically***.

- **The technique that accomplishes this is called *garbage collection*. It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.**

There is no explicit need to destroy objects as in C++.

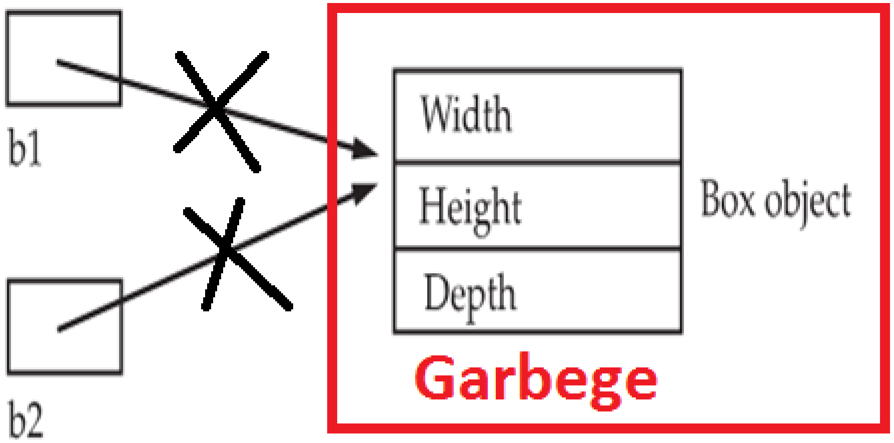
How Garbage is created?

```
class Box{
        int Width;
        int  Height;
        int  Depth;
}
```



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

b2 = b1 = null;
```



Java Garbage Collection : When?

- Garbage collection only occurs sporadically (if at all) during the execution of your program.
- It will not occur simply because one or more objects exist that are no longer used.
- Furthermore, different Java run-time implementations will take varying approaches to garbage collection.

For the most part, you should not have to think about it while writing your programs.

The finalize( ) Method

- Sometimes an object will need to perform some action when it is destroyed.
- For example, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed.
- To handle such situations, Java provides a mechanism called *finalization*. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

The finalize( ) Method

- To add a finalizer method to a class, you simply define the **finalize( )** method.
- Inside the **finalize( )** method, you will specify those actions that must be performed before an object is destroyed.

General form of The finalize( ) method:

```
protected void finalize( )
        {
                // finalization code here
        }
```

The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the Java run time calls the **finalize( )** method on the object.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The finalize( ) Method :Example

```
Class A{
 protected void finalize() throws
Throwable
    {
     System.out.println("Finalize “);
    }
  public static void main(String[] args)
   {
         A a1 = new A(10);
         A a2 = new A(20);
             a1 = a2;
       System.gc();
       System.out.println("done");
   }
}
```

Output:

```
Finalize
done
```

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST

The finalize( ) Method :Example

```
Class A{
 public A(int i)
   {       this.i = i;    }
   @Override
 protected void finalize() throws Throwable
   {
    System.out.println("Finalize Method, i = "+i);
   }
```

```
   public static void main(String[] args)
   {
      //Creating two instances of class A
      A a1 = new A(10);
      A a2 = new A(20);
      //Assigning a2 to a1
      a1 = a2;
 //Now both a1 and a2 will be pointing same object
 //An object earlier referred by a1 will become abandoned

//Calling garbage collector thread explicitly
      System.gc();
     //OR call Runtime.getRuntime().gc();
      System.out.println("done");
   }
}
```

The finalize( ) Method : In Practice

- It is important to understand that **finalize( )** is only called just prior to garbage collection.
- It is not called when an object goes out-of-scope, for example.
- This means that you cannot know when—or even if—**finalize( )** will be executed.

Therefore, your program should provide other means of releasing system resources, etc., used by the object.

*You must not rely on **finalize( )** for normal program operation.*

finalize( ) Method & Destructor

- If you are familiar with C++, then you know that C++ allows you to define a destructor for a class, which is called when an object goes out-of-scope.
- Java does not support this idea or provide for destructors.
- The finalize( ) method only approximates the function of a destructor.

Md. Mamun Hossain
B. Sc. (Engg.) & M. Sc. (Thesis) in CSE, SUST
Asst. Professor, Dept. of CSE , BAUST