CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

MODULE II JAVA OBJECTS -1	L	T	P	EL
	3	0	4	3
Classes and Objects, Constructor, Destructor, Static instances, this, constants, Thinking in Objects, String class, Text I/O				
SUGGESTED ACTIVITIES : • Flipped classroom • Practical - Implementation of Java programs – using String class, Creating Classes and objects • EL – Thinking in Objects				
SUGGESTED EVALUATION METHODS: • Assignment problems • Quizzes				

- **procedural programming:** Programs that perform their behavior as a series of steps to be carried out.
- **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects

About ?

- The basic elements of a class
- How a class can be used to create objects?
- Learn about methods, constructors, and the **this** keyword.

Class

- Any concept to implement in a Java program must be encapsulated within a class.
- What is a class?
  - A class is a structure that defines the data and the methods to work on that data.
  - A new data type
  - A class is an encapsulation of attributes and methods.
  - A class is a logical construct or template.

Class Fundamentals

- Class
  - Defines a new data type
  - A class is to create an object or instance of the defined type
  - A class is a template for an object
  - A class is the blueprint of an object
  - A class describe object's properties and behaviors
- Example: Blueprint of a house (class) and the house (object)

Blueprint

```
public class Point
{
    int x;
    int y;
}
```

Instance

```
Point p=new Point();
```

Class

Properties : State/variables

Behaviors: Methods

Class Clock

Properties : State/variables

int Hour, minute, second;

Behaviors: Methods

int getMinutes();
int getSeconds();
int getHours();

Container class vs Definition class

- Container class:
  - Collection of static methods that are not bound to a specific objects.
  - Example: Math.sqrt(), Math.pow()
- Definition class:
  - A class that create new objects.
  - Example: Clock c1;

Object

- Object
  - An object is an instance of a class.
  - An entity that combines state and behavior
  - An object is a real world entity.

Characteristics of an Object:

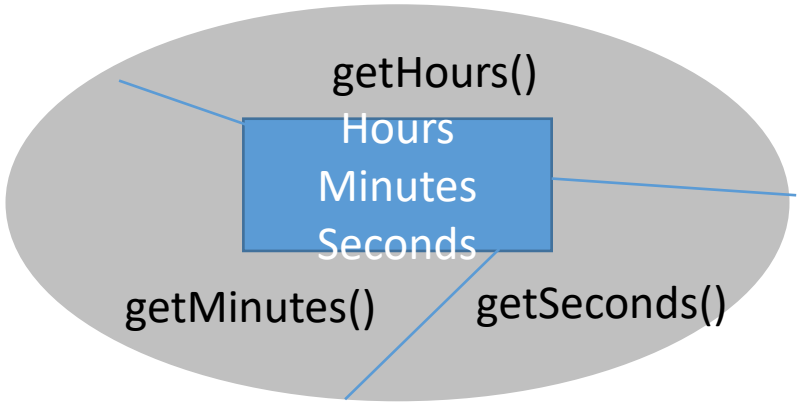
- State : represents the data (value) of an object.
- Behavior : represents the behavior (functionality) of an object.
- Identity : An object identity is typically implemented via unique ID. The value of the ID is not visible to the external user. However, JVM can identify each object uniquely.

Class Abstraction

- Abstract data type (ADT)
  - Abstraction refers to the act of representing essential features without including the background details or explanations.
  - Class is an ADT as it uses the concept of abstraction.
- Abstraction:
  - An abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
  - Allow user to understand its external behaviour(interface) but not its internal details.
  - Example: Buttons
  - Why? To manage complexity
  - abstraction is achieved by using Interfaces and Abstract classes.

Encapsulation

- State is *encapsulated* or *hidden*
  - The internal state of an object is not directly accessible to other parts of the program
  - Other parts of the program can only access the object using its interface.
  - Data-hiding: Data of a class is hidden from any other class and can be accessed only through any member function of it's own class in which they are declared.



General form of the 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
    }
}
```

- The data, or variables, defined within a class are called instance variables.
  - Each instance of the class (that is, each object of the class) contains its own copy of these variables.
  - Thus, the data for one object is separate and unique from the data for another.
- The code is contained within .defined within a class are called members of the class.
- Classes have a main method(), if that class is the starting point of the program

A simple class

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

To actually create a Box object, use a statement like the following:
Box mybox = new Box();
// create a Box object called mybox

Instance variables

A class called Box that defines three instance variables:
width, height, and depth.

Class name

A class defines a new type of data.
In this case, the new data type is called Box.
class name is used to declare objects of type Box.

A class declaration only creates a template;
It does not create an actual object.

Object

mybox will refer to an instance of Box. Thus, it will have “physical” reality.

Each time on creating an instance of a class means creating an object that contains its own copy of each instance variable defined by the class

Every Box object will contain its own copies of the instance variables width, height, and depth.

A simple class

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

To actually create a Box object, use a statement like the following:
Box mybox = new Box();
// create a Box object called mybox

mybox.width = 100;
//assign width variable of mybox the value 100

Dot operator

Use the dot operator to access both the instance variables and the methods within an object.
The dot operator links the name of the object with the name of an instance variable or methods.

```
/* A program that uses the Box class.

   Call this file BoxDemo.java
*/
class Box {
  double width;
  double height;
  double depth;
}

// This class declares an object of type Box.
class BoxDemo {
  public static void main(String args[]) {
    Box mybox = new Box();
    double vol;

    // assign values to mybox's instance variables
    mybox.width = 10;
    mybox.height = 20;
    mybox.depth = 15;

    // compute volume of box
    vol = mybox.width * mybox.height * mybox.depth;

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

Output:
Volume is 3000.0

Save the file that contains this program BoxDemo.java, because the main( ) method is in the class called BoxDemo, not the class called Box.
Compilation will create two .class files, one for Box and one for BoxDemo.
Each class can also be defined in its own file, called Box.java and BoxDemo.java,

```
// This program declares two Box objects.

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

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

    // assign values to mybox1's instance variables
    mybox1.width = 10;
    mybox1.height = 20;
    mybox1.depth = 15;

    /* assign different values to mybox2's
       instance variables */
    mybox2.width = 3;
    mybox2.height = 6;
    mybox2.depth = 9;

    // compute volume of first box
    vol = mybox1.width * mybox1.height * mybox1.depth;
    System.out.println("Volume is " + vol);

    // compute volume of second box
    vol = mybox2.width * mybox2.height * mybox2.depth;
    System.out.println("Volume is " + vol);
  }
}
```

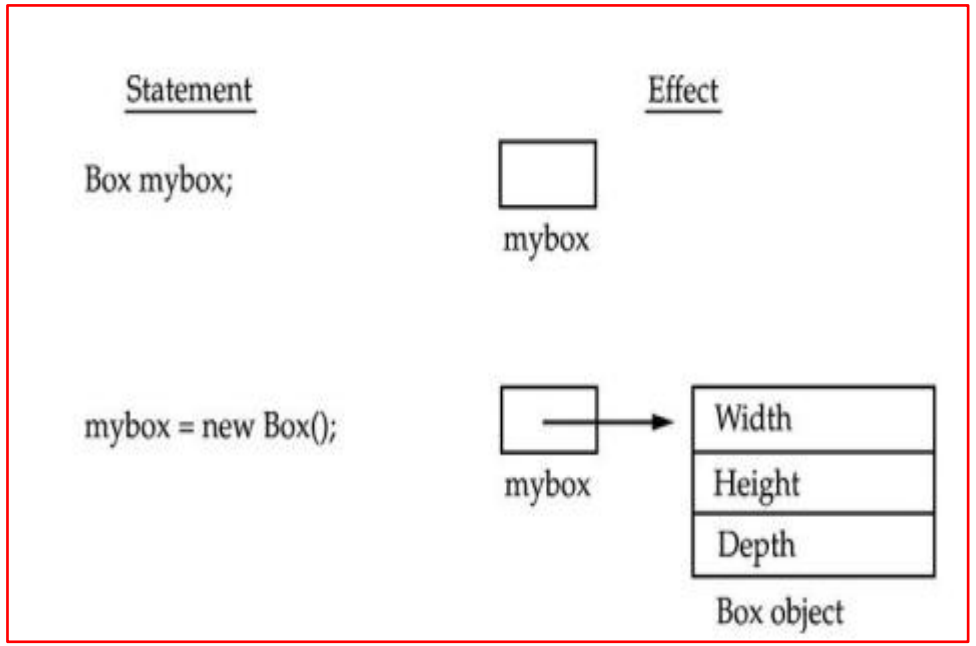
Every Box object will contain its own copies of the instance variables width, height, and depth.

Changes to the instance variables of one object have no effect on the instance variables of another.

Output :
Volume is 3000.0
Volume is 162.0

Declaring Objects

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



Declaring an object of type Box

Declare object

The first line declares mybox as a reference to an object of type Box.

At this point, mybox does not yet refer to an actual object.

The next line allocates an object and assigns a reference to it to mybox.

mybox simply holds the memory address of the actual Box object.

new operator

- The new operator dynamically allocates memory for an object.

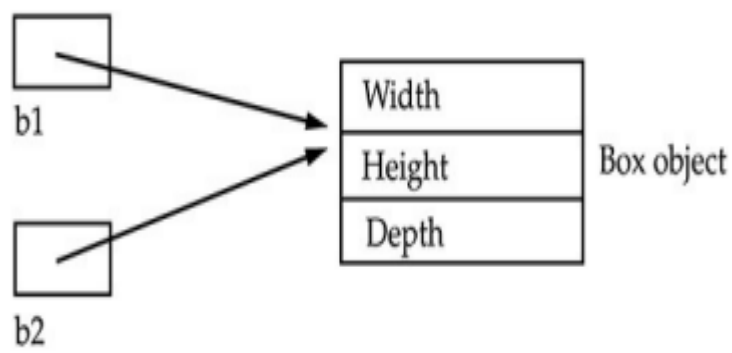
  class-var = new classname ( );

- The classname is the name of the class that is being instantiated.
- The class name followed by parentheses specifies the constructor for the class.
- A **constructor** defines what occurs when an object of a class is created.
- Most real-world classes explicitly define their own constructors within their class definition.
- However, if no explicit constructor is specified, then Java will automatically supply a default constructor.

Assigning Object Reference Variables

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

- b1 and b2 will both refer to the same object.
- The assignment of b1 to b2 did not allocate any memory or copy any part of the original object.
- It simply makes b2 refer to the same object as does b1.
- Thus, any changes made to the object through b2 will affect the object to which b1 is referring, since they are the same object.
- Assign one object reference variable to another object reference variable, is not creating a copy of the object, only making a copy of the reference.




```
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

- Classes usually consist of two things:
  - instance variables
  - methods.
- This is the general form of a method:

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

Type

Type specifies the type of data returned by the method.

This can be any valid type, including class types

If the method does not return a value, its return type must be void.

Name

The name of the method is specified by name.

This can be any legal identifier .

Parameter-list

The parameter-list is a sequence of type and identifier pairs separated by commas.

Parameters are essentially variables that receive the value of the arguments passed to the method when it is called.

If the method has no parameters, then the parameter list will be empty.

Return

Methods that have a return type other than void return a value to the calling routine using the return statement:

Adding a Method to the Box Class

Use methods to access the instance variables defined by the class.

In fact, methods define the interface to most classes.

This allows the class implementor to hide the specific layout of internal data structures behind cleaner method abstractions.

In addition to defining methods that provide access to data, we can also define methods that are used internally by the class itself.

Instance variable that is not part of the class, it must be accessed through an object, by use of the dot operator.
Instance variable that is part of the same class, that variable can be accessed directly.
The same thing applies to methods.

```
// This program includes a method inside the box class.

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

  // display volume of a box
  void volume() {
    System.out.print("Volume is ");
    System.out.println(width * height * depth);
  }
}

class BoxDemo3 {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();

    // assign values to mybox1's instance variables
    mybox1.width = 10;
    mybox1.height = 20;
    mybox1.depth = 15;

    /* assign different values to mybox2's
       instance variables */
    mybox2.width = 3;
    mybox2.height = 6;
    mybox2.depth = 9;

    // display volume of first box
    mybox1.volume();

    // display volume of second box
    mybox2.volume();
  }
}
```

No dot operator to access instance variable within the class

dot operator used to access instance variable outside the class

Output:
Volume is 3000.0
Volume is 162.0

Adding a Method That Takes Parameters

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

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

parameter

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

argument

- A parameterized method can operate on a variety of data.
- A non parameterized method use is very limited

parameter and argument:

- A parameter is a variable defined by a method that receives a value when the method is called.
- For example, in square( ), i is a parameter.
- An argument is a value that is passed to a method when it is invoked.
- For example, square(100) passes 100 as an argument.

```
// This program uses a parameterized method.

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

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

  // sets dimensions of box
  void setDim(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
}

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

    // initialize each box
    mybox1.setDim(10, 20, 15);
    mybox2.setDim(3, 6, 9);

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

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

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

setDim( )
method to set the dimensions of each box.

Constructors

- A constructor initializes an object immediately upon creation.
- Automatic initialization of object is performed through the use of a constructor.
- Constructor has the same name as the class in which it resides and is syntactically similar to a method.
- Once defined, the constructor is automatically called when the object is created, before the new operator completes.
- Constructor's have no return type, not even void.
  - The implicit return type of a class' constructor is the class type itself.

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

The parentheses are needed after the class name is to invoke the constructor for the class is being called.

```
/* Here, Box uses a constructor to initialize the
   dimensions of a box.
*/
class Box {
  double width;
  double height;
  double depth;

  // This is the constructor for Box.
  Box() {
    System.out.println("Constructing Box");
    width = 10;
    height = 10;
    depth = 10;
  }

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

class BoxDemo6 {
  public static void main(String args[]) {
    // declare, allocate, and initialize Box objects
    Box mybox1 = new Box();
    Box mybox2 = new Box();

    double vol;

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

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

Output:

Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0

Default vs defined constructor

- Java creates a default constructor for the class if a constructor is not explicitly defined for a class.
- The default constructor will initialize all non initialized instance variables to their default values.
  - zero, null, and false, for numeric types, reference types, and boolean.
- Once define your own constructor, the default constructor is no longer used.

```
/* Here, Box uses a parameterized constructor to
   initialize the dimensions of a box.
*/
class Box {
  double width;
  double height;
  double depth;

  // This is the constructor for Box.
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

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

class BoxDemo7 {
  public static void main(String args[]) {
    // declare, allocate, and initialize Box objects
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box(3, 6, 9);

    double vol;

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

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

Parameterized Constructor

Non-parameterized constructor allow all boxes to have the same dimensions.

Parameterized constructor is a way **to construct Box objects of various dimensions.**

Output:
Volume is 3000.0
Volume is 162.0

The this Keyword

- **this keyword** allows a method to refer to the object that invoked it.
- **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.
- use this anywhere as a reference to an object of the current class' type.

```
// This is the constructor for Box.
Box(double w, double h, double d) {
  width = w;
  height = h;
  depth = d;
}
```

```
// A redundant use of this.
Box(double w, double h, double d) {
  this.width = w;
  this.height = h;
  this.depth = d;
}
```

this keyword:Instance Variable Hiding

- It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.
- There can be Overlap of local variables, including formal parameters to methods with the names of the class' instance variables.
- However, when a local variable has the same name as an instance variable, **the local variable hides the instance variable**.
- this directly refers to the object,
- use this to resolve any namespace collisions that might occur between instance variables and local variables.

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

A 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;
  }

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

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

```
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());
  }
}
```

This program generates the following output:

```
Stack in mystack1:
9
8
7
6
5
4
3
2
1
0
Stack in mystack2:
19
18
17
16
15
```

```
14
13
12
11
10
```

Garbage collections

- In java, since objects are dynamically allocated by using the new operator, such objects are destroyed and their memory released for later reallocation automatically.
- In some languages, such as traditional C++, dynamically allocated objects must be manually released by use of a delete operator.
- 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 need to explicitly destroy objects.
- Garbage collection only occurs sporadically (if at all) during the execution of your program.