Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf

yousuf@juniv.edu

Class and Object

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.

- An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

- **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.

Class declaration

- A *class* is a template for manufacturing objects. You declare a class by specifying the class keyword followed by a non-reserved identifier that names it. A pair of matching open and close brace characters ({ and }) follow and delimit the class's body.

```
class identifier
{
    // class body
}
```

```
class Book
{
    // class body
}
```

- A class's body is populated with fields, methods, and constructors. Combining these language features into classes is known as *encapsulation*.

Multi-class applications and main()

- A Java application is implemented by one or more classes. Small applications can be accommodated by a single class, but larger applications often require multiple classes. In that case one of the classes is designated as the *main class* and contains the main() entry-point method.

```
class A
{
}

class B
{
}

class C
{
   public static void main(String[] args)
   {
      System.out.println("Application C entry point");
   }
}
```

Object in Java

- An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical.
- An object has three characteristics:
  - **state:** represents data (value) of an object.
  - **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
  - **identity:** Object identity is typically implemented via a unique ID.
- **Object is an instance of a class.** Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

A Simple Class

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

- As stated, a class defines a new type of data. In this case, the new data type is called **Box. will use this name to declare objects of type Box. It is important to remember** that a class declaration only creates a template; it does not create an actual object.
- To actually create a **Box object, you will use a statement like the following:**

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

The new keyword is used to allocate memory at runtime

A Simple Class

- each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class. Thus, every **Box object will contain its own copies of the instance variables width, height,** and **depth.**

- To access these variables, you will use the *dot (.) operator. The dot operator* links the name of the object with the name of an instance variable.

  For example, **mybox.width = 100;**

Declaring Objects

- 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.
-
- This reference is, more or less, the address in memory of the object allocated by **new. This** reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.

Declaring Objects

- To declare an object of type **Box:**

  Box mybox = new Box();

- This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

  Box mybox; // declare reference to object

  mybox = new Box(); // allocate a Box object

```
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);
  }
}
```

Here is a complete program that uses the **Box class**

Simple Example of Object and Class

```
1.class Student{
2. //defining fields
3. int id;//field or data member or instance variable
4. String name;
5. //creating main method inside the Student class
6. public static void main(String args[]){
7.  //Creating an object or instance
8.  Student s1=new Student();//creating an object of Student
9.  //Printing values of the object
10.  System.out.println(s1.id);//accessing member through reference variable
11.  System.out.println(s1.name);
12. }
13.}
```

Output:
0
null

The new keyword is used to allocate memory at runtime

Java Program to demonstrate having the m ain method in another class

```
1.class Student{
2. int id;
3. String name;
4.}
5.//Creating another class TestStudent1 which contains the main
 method
6.class TestStudent1{
7. public static void main(String args[]){
8.  Student s1=new Student();
9.  System.out.println(s1.id);
10.  System.out.println(s1.name);
11. }
12.}
```

It is a better approach than previous one.

3 Ways to initialize object

- There are 3 ways to initialize object in java.
  - By reference variable
  - By method
  - By constructor

3 Ways to initialize object

Initialization through reference

```
1.class Student{
2. int id;
3. String name;
4.}
5.class TestStudent3{
6. public static void main(String args[]){
7.  //Creating objects
8.  Student s1=new Student();
9.  Student s2=new Student();
10.  //Initializing objects
11.  s1.id=101;
12.  s1.name="Rahim";
13.  s2.id=102;
14.  s2.name="Samia";
15.  //Printing data
16.  System.out.println(s1.id+" "+s1.name);
17.  System.out.println(s2.id+" "+s2.name);
18. }
19.}
```

Output:
101 Rahim
102 Samia

3 Ways to initialize object

Initialization through method

```
1.class Student{
2. int rollno;
3. String name;
4. void insertRecord(int r, String n){
5.  rollno=r;
6.  name=n;
7. }
8. void displayInformation(){System.out.println(rollno+" "+name);}
9.}
10.class TestStudent4{
11. public static void main(String args[]){
12.  Student s1=new Student();
13.  Student s2=new Student();
14.  s1.insertRecord(101,"Rahim");
15.  s2.insertRecord(102,"Samia");
16.  s1.displayInformation();
17.  s2.displayInformation();
18. }
19.}
```

Output:
101 Rahim
102 Samia

```
1.class Employee{
2.    int id;
3.    String name;
4.    float salary;
5.    Employee (int i, String n, float s) {
6.        id=i;
7.        name=n;
8.        salary=s;
9.    }
10.    void display(){System.out.println(id+" "+name+" "+salary);}
11.}
12.public class TestEmployee {
13.public static void main(String[] args) {
14.    Employee e1=new Employee(101,"Rahim",45000);
15.    Employee e2=new Employee(102,"Samia",25000);
16.    Employee e3=new Employee(103,"Karim",55000);
17.    e1.display();
18.    e2.display();
19.    e3.display();
20.}
21.}
```

3 Ways to initialize object

Initialization through constructor

Creating multiple objects by one type only

- Each object has its own copies of the instance variables. This means that if you have two **Box objects, each has its own copy of depth, width, and height.**

- **It** is important to understand that changes to the instance variables of one object have no effect on the instance variables of another.

Creating multiple objects by one type only

```
1.class Rectangle{
2. int length;
3. int width;
4. void insert(int l, int w){
5.  length=l;
6.  width=w;
7. }
8. void calculateArea(){System.out.println(length*width);}
9.}
10.class TestRectangle1{
11. public static void main(String args[]){
12.  Rectangle r1=new Rectangle();
13.  Rectangle r2=new Rectangle();
14.  r1.insert(11,5);
15.  r2.insert(3,15);
16.  r1.calculateArea();
17.  r2.calculateArea();
18.}
19.}
```

Output:
55
45

Creating multiple objects by one type only

```
// 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);
  }
}
```

Assigning Object Reference Variables

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

- You might think that **b2 is being assigned a reference to a copy of the object referred to** by **b1. That is, you might think that b1 and b2 refer to separate and distinct objects.**

- However, this would be wrong. Instead, after this fragment executes, **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.**

Introducing Methods

- The general form of a method:

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

- Here, *type specifies the type of data returned by the method. This can be any valid type,* including class types that you create. If the method does not return a value, its return type must be **void. The name of the method is specified by *name.***
- The *parameter-list is a sequence of type and identifier pairs separated by commas.*

```
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();
      }
    }
```

Volume method does not return a value

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

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

class BoxDemo4 {
  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;

    // 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);
  }
}
```

Volume method returns a value

```
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);
      }
    }
```

Adding a method that takes parameters

Method Overloading in Java

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.

- Different ways to overload the method
- There are two ways to overload the method in java
  - By changing number of arguments
  - By changing the data type

- 1) Method Overloading: changing no. of arguments

```
1.class Adder{
2.static int add(int a,int b)
3.                {return a+b;}
4.static int add(int a,int b,int c)
5.               {return a+b+c;}
6.}
7.class TestOverloading1{
8.public static void main(String[] args){
9. System.out.println(Adder.add(11,11));
10.System.out.println(Adder.add(11,11,11));
11.}
12.}
```

Output:
22
33

- 2) Method Overloading: changing data type of arguments

```
1.class Adder{
2.static int add(int a, int b)
3.              {return a+b;}
4.static double add(double a, double b)
5.             {return a+b;}
6.}
7.class TestOverloading2{
8.public static void main(String[] args){
9.System.out.println(Adder.add(11,11));
10.System.out.println(Adder.add(12.3,12.6));
11.}}
```

Thank you