OBJECT ORIENTED PROGRAMMING

Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

CLASS FUNDAMENTALS

- A class, defines a new data type.
- Once defined, this new type can be used to create objects of that type.
- Thus, a class is a *template for an object*, *and an object is an instance of a class*.
- *Because an object is an instance* of a class, the two words *object and instance used interchangeably.*

THE GENERAL FORM OF A CLASS

- A class is declared by use of the **class** keyword.
- Classes may contain only code or only data, most real-world classes contain both.
- Syntax:

```
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 GENERAL FORM OF A CLASS

- The data, or variables, defined within a class are called *instance variables*.
- *The code is* contained within *methods*.
- *Collectively, the methods and variables defined within a class are* called *members of the class*.
- Variables defined within a class are called instance variables because 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.

A SIMPLE CLASS

- a **class** declaration only creates a template;

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

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

- mybox will be an instance of Box. Thus, it will have "physical" reality.
- To access class variables, use the *dot (.) operator*.
- *The dot operator links the name of the* object with the name of an instance variable.

```
mybox.width = 100;
```

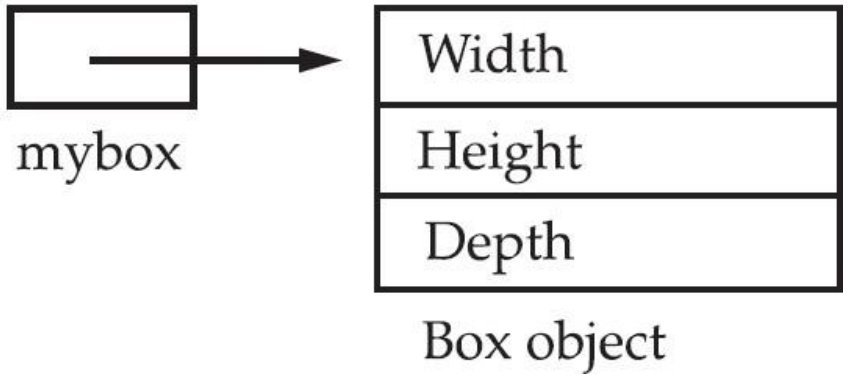
EXAMPLE

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

DECLARING OBJECTS

- 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*.
  - Box mybox; // declare reference to 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**.
  - mybox = new Box(); // allocate a Box object
- 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.

OBJECTS

Statement	Effect
Box mybox;	
mybox = new Box();	

A CLOSER LOOK AT NEW

- the new operator dynamically allocates memory for an object.
  - *class-var = new classname( );*
- *class-var is a variable of the class type being created.*
- *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.
- If no explicit constructor is specified, then Java will automatically supply a default constructor.
- A class is a logical construct. An object has physical reality.

ASSIGNING OBJECT REFERENCE VARIABLES

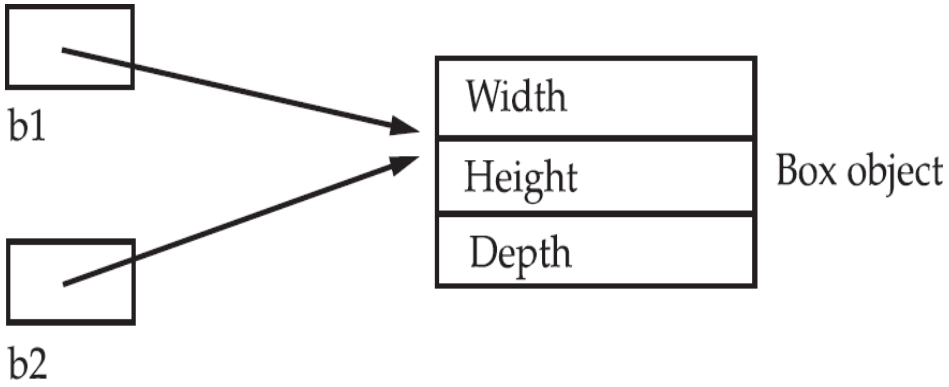
- Object reference variables act differently when an assignment takes place.

```
Box b1 = new Box();

Box b2 = b1;
```

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

// ...
b1 = null;
```



- Here, **b1 has been set to null, but b2 still points to the original object.**
- *When you assign one object reference variable to another object reference variable, you are not creating a copy of the object, you are only making a copy of the reference.*

INTRODUCING METHODS

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

- *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. This can be any legal identifier* other than those already used by other items within the current scope.
- 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.
- *value is the value returned.*

ADDING METHOD

- Method With parameters
- Method Without Parameters
- Method Return a value
- The type of data returned by a method must be compatible with the return type specified by the method.
- The variable receiving the value returned by a method **must** also be compatible with the return type specified for the method.

CONSTRUCTORS

- Automatic initialization of an object is performed through the use of a constructor.
- A *constructor initializes an object immediately upon creation*.
- It 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 immediately after the object is created, before the **new operator completes.**
- Constructors have no return type, not even **void.** This is because the implicit return type of a class constructor is the class type itself.
- The constructor's job to initialize the internal state of an object.

CONSTRUCTORS

- Constructors are
  - Automatic / Default Constructors
    - Box mybox1 = new Box();
    - The default constructor automatically initializes all instance variables to zero.
  - Parameterized Constructors
    - Box mybox1 = new Box(10, 20, 15);
  - Copy Constructors

PARAMETERIZED CONSTRUCTORS

- These constructors take parameters, allowing you to initialize the instance variables with specific values during object creation.

```
Box(double w, double h, double d) {
    // This is a parameterized constructor
    // It initializes width, height, and depth with the provided values
    width = w;
    height = h;
    depth = d;
}
```

DEFAULT OR AUTOMATIC CONSTRUCTOR

- A default constructor is one that is automatically provided by the compiler if no constructor is explicitly defined within a class.
- This default constructor initializes instance variables to default values (e.g., zero for numeric types, false for boolean, and null for object references).
- It's sometimes referred to as the "default" constructor because it's automatically provided if none other is defined.

```
Box() {
    // This is a default constructor
    // It initializes width, height, and depth to default values
}
```

COPY CONSTRUCTORS

- These constructors create a new object by copying values from another object of the same class. It's useful for creating a new object with the same state as an existing one. For example:

```
Box(Box originalBox) {
    // This is a copy constructor
    // It creates a new Box object with the same dimensions as originalBox
    width = originalBox.width;
    height = originalBox.height;
    depth = originalBox.depth;
}
```

EXAMPLE
DEFAULT CONSTRUCTOR

Output:

```
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0
```

```
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 BoxDemo {
   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);
   }
}
```

EXAMPLE PARAMETERIZED CONSTRUCTOR

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

   // Constructor with parameters
   Box(double w, double h, double d) {
      width = w;
      height = h;
      depth = d;
   }

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

```
class BoxDemo {
   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);
   }
}
```

Output:

```
Volume is 3000.0
Volume is 162.0
```