INTRODUCING CLASSES

- The class is the core of Java & it forms the basis for object-oriented programming in Java.

CLASS FUNDAMENTALS

- The most important thing to understand about a class is that it defines a new data type.
- Once defined, this new type can be used to create objects of that type.
- A class is a template for an object and an object is an instance of a class.

GENERAL FORM OF A CLASS

- When defining a class, the data that it contains and the code that operates on that data has to be specified.
- A class is declared by use of the ‘class’ keyword.
- General form:

```
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.
- The code is contained within methods.
- It is the methods that determine how a class data can be used.
- Java classes have a main( ) method only if that class is the starting point for the program.

A SIMPLE CLASS

- Here is a class called Box that defines 3 instance variables: width, height and depth

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

- A class defines a new type of data. In this case, the new data type is called Box.
- This name is used to declare objects of type Box.
- To create a Box object, the following statement is used.

  Box mybox = new Box( );
- 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, the dot (.) operator is used.
- The dot operator links the name of the object with the name of an instance variable.

  eg. mybox.width = 100;
- Program

- When the above program is compiled, two .class files have been created, one for Box and one for BoxDemo.
- The Java compiler automatically puts each class into its .class file.
- It is not necessary for both the Box and the BoxDemo class to actually be in the same source file.
- But to run this program, BoxDemo.class has to be executed.
- Changes to the instance variables of one object have no effect on the instance variables of another in the case of having more than one or two objects.

DECLARING OBJECTS

- Obtaining objects of a class is a two-step process
  - Declare a variable of the class type
  - Acquire an actual physical copy of the object and assign it to that variable, using the new operator.
- The new operator dynamically allocates memory for an object and returns a reference to it.
- This reference is then stored in the variable.
- This reference is more or less, the address in memory of the object allocated by new.

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

- The effect of these two lines of code is depicted below:

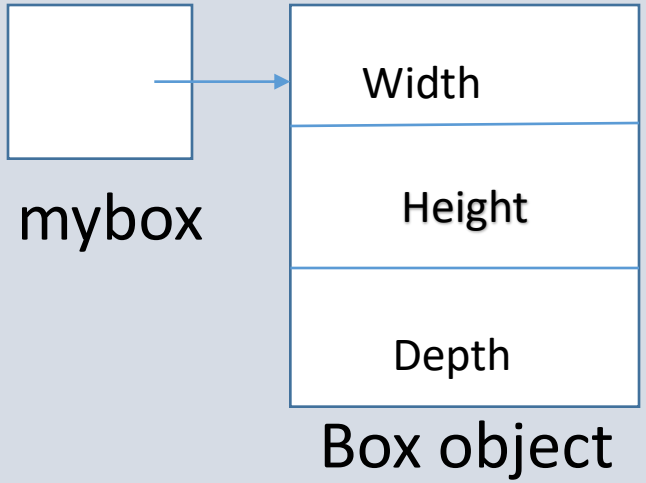
Statement	Effect
Box mybox;	
mybox = new Box( );	

fig: Declaring an object of type Box

NEW OPERATOR

- The new operator dynamically allocates memory for an object.
- General form:

```
class_var = new Classname( );
```

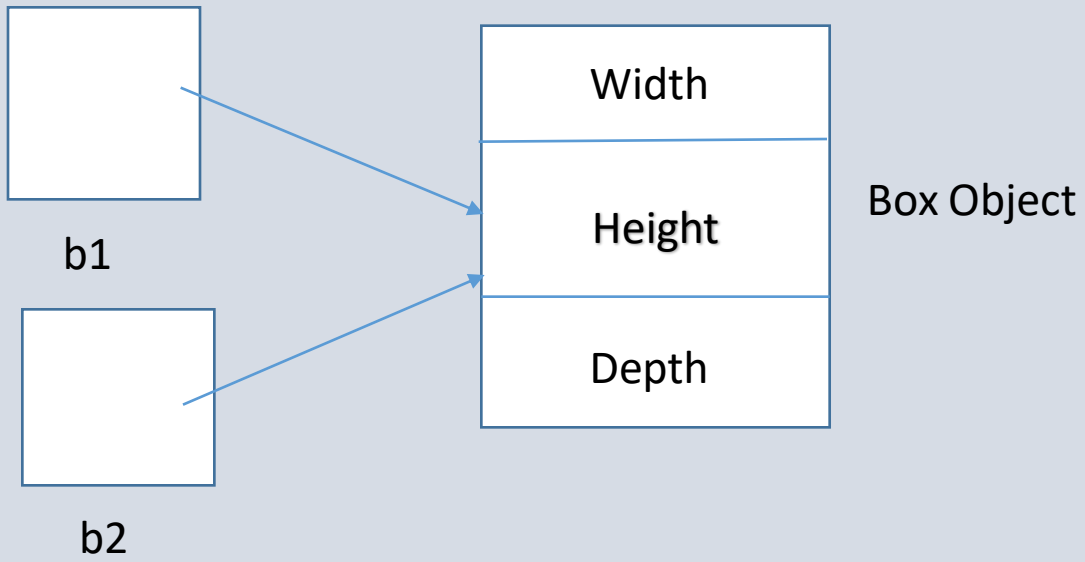
- The Classname followed by parenthesis specifies the constructor for the class.
- A constructor defines what occurs when an object of a class is created.
- If no explicit constructor is specified, then Java will automatically supply a default constructor.
- It is possible that new will not be able to allocate memory for an object because of insufficient memory. If this happens, a run-time exception will occur.

ASSIGNING OBJECT REFERENCE VARIABLES

- Consider the following fragment,

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

- When this fragment is executed, both b1 and b2 will refer to the same object.
- The assignment of b1 to b2 did not allocate any memory or copy any part of the original part.
- It simply makes b2 refer to the same object as does b1.
- Thus any changes made to the objet through b2 will affect the object to which b1 is referring, since they are the same object.



- Although b1 and b2 refer to the same object, they are not linked in any other way.

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

- The above assignment (b1 = null;) will simply unhook b1 from the original object, without affecting the object or affecting b2.

INTRODUCING METHODS

- General form of a method

```
type name(parameter_list)
{
        //body of method
}
```

- Here, type specifies the type of data returned by the method.
- If the method does not return any value, its return type must be void.
- 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.

- Methods that have a return type other than void return a value to the calling routine using the following form of the return statement

  return value;

- Here value is the value returned.

ADDING A METHOD TO THE BOX CLASS

- Methods are mostly used to access the instance variables defined by the class. In fact, methods define the interface to most classes.
- Methods can also be defined to be used internally by the class itself.
- Program

RETURNING A VALUE

- Program
- vol is a variable that will receive the value returned by volume( )
- 2 important things about returning values
  - The type of the 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.

ADDING A METHOD THAT TAKES PARAMETERS

- Parameters allow a method to be generalized.
- A parameterized method can operate on a variety of data and/or can be used in a number of slightly different situations.

- Eg

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

- square( ) will return the square of whatever value it is called with

```
x = square(5); // x equals 25
```

- **Parameter & Argument**
  - A parameter is a variable defined by a method that receives a value when the method is called.
  - An argument is a value that is passed to a method when it is invoked.
- Program

CONSTRUCTORS

- Java allows objects to initialize themselves when they are created.
- This automatic initialization 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.
- They have no return type, not even void, because the implicit return type of a class' constructor is the class itself.
- It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.
- Program

- When a constructor is not explicitly defined, Java creates a default constructor for a class, that automatically initializes all instance variables to zero.

PARAMETERIZED CONSTRUCTORS

- If we need to create Box objects of various dimensions, solution is to add parameters to the constructor.
- Program

The this keyword

- Java defines the this keyword, that can be used inside any method to refer to the current object.
- i.e., this is always a reference to the object on which the method was invoked.

```
Eg.    Box(double w, double h, double d){
            this.width = w;
            this.height = h;
            this.depth = d;
       }
```

- Inside Box( ), this will always refer to the invoking object.

Instance Variable Hiding

- It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.
- When a local variable has the same name as an instance variable, the local variable hides the instance variable.
- this keyword lets you refer directly to the object, you can use it to resolve any name collisions that might occur between instance variables and local variables.

```
Eg.    Box(double width, double height, double depth){
            this.width = width;
            this.height = height;
            this.depth = depth;
       }
```

- Using of this keyword overcomes instance variable hiding.

GARBAGE COLLECTION

- In C++, dynamically allocated objects must be manually released by use of a delete operator.
- Java handles deallocation automatically, when no references to an object exist, then that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. This technique is called Garbage collection.

The finalize( ) Method

- If an object is holding some non-Java resources such as a file handle or window character font, then these resources must be freed before that object is destroyed.
- Java provides a mechanism called finalization and by using this finalization, specific actions can be defined that will occur when an object is just to be reclaimed by the garbage collector.

- To add a finalizer to a class, the finalize( ) method has to be defined.
- Inside the finalize( ) method, specify those actions that must be performed before an object is destroyed.
- 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 runtime system calls the finalize( ) method on the object.
- General form

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

- The keyword protected is a specifier that prevent access to finalize( ) by code defined outside its class.
- finalize( ) is only called just prior to garbage collection.
- We cannot know when or even if finalize( ) will be executed, so our program should provide other means of releasing system resources.
- It must not rely on finalize( ) for normal program execution.

STACK CLASS PROGRAM

A Stack Class

- A stack stores data using first-in, last-out ordering.
- Stacks are controlled through two operations called push and pop.
- Program

METHOD OVERLOADING

- In Java, it is possible to define two or more methods within the same class that share the same name as long as their parameter declarations are different.
- In this case, the methods are said to be overloaded, and the process is referred to as method overloading.
- **Method overloading is one of the ways in which Java implements Polymorphism.**
- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Thus, overloaded methods must differ in the type and/or number of their parameters.

- Program
- When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method's parameters.
- Java's automatic type conversions can play a role in overload resolution.
- If an integer argument is passed to method of type double, then Java can automatically convert an integer into double, and this conversion can be used to resolve the call.
- Method overloading supports polymorphism by which Java implements the "one interface, multiple methods"
- In C, the function abs( ) returns the absolute value of an integer, labs( ) returns the absolute value of a long integer, fabs( ) returns the absolute value of a floating-point value.

- Java’s standard class library includes an absolute value method, called abs( ).
- This method is overloaded by Java’s Math class to handle all numeric types.
- Java determines which version of abs( ) to call based upon the type of argument.
- The value of overloading is that it allows related methods to be accessed by use of a common name.
- There is no rule stating that overloaded methods must relate to one another.
- However, you should not overload unrelated methods by using the same name.
- Closely related operations can be overloaded.

CONSTRUCTOR OVERLOADING

- In Java, it is possible to define two or more constructors with the same name but with different parameter declarations to handle different situations.
- In this case, the constructors are said to be overloaded and the process is referred to as constructor overloading.
- Program

USING OBJECTS AS PARAMETERS

- It is common to pass objects as parameters to methods.
- Program