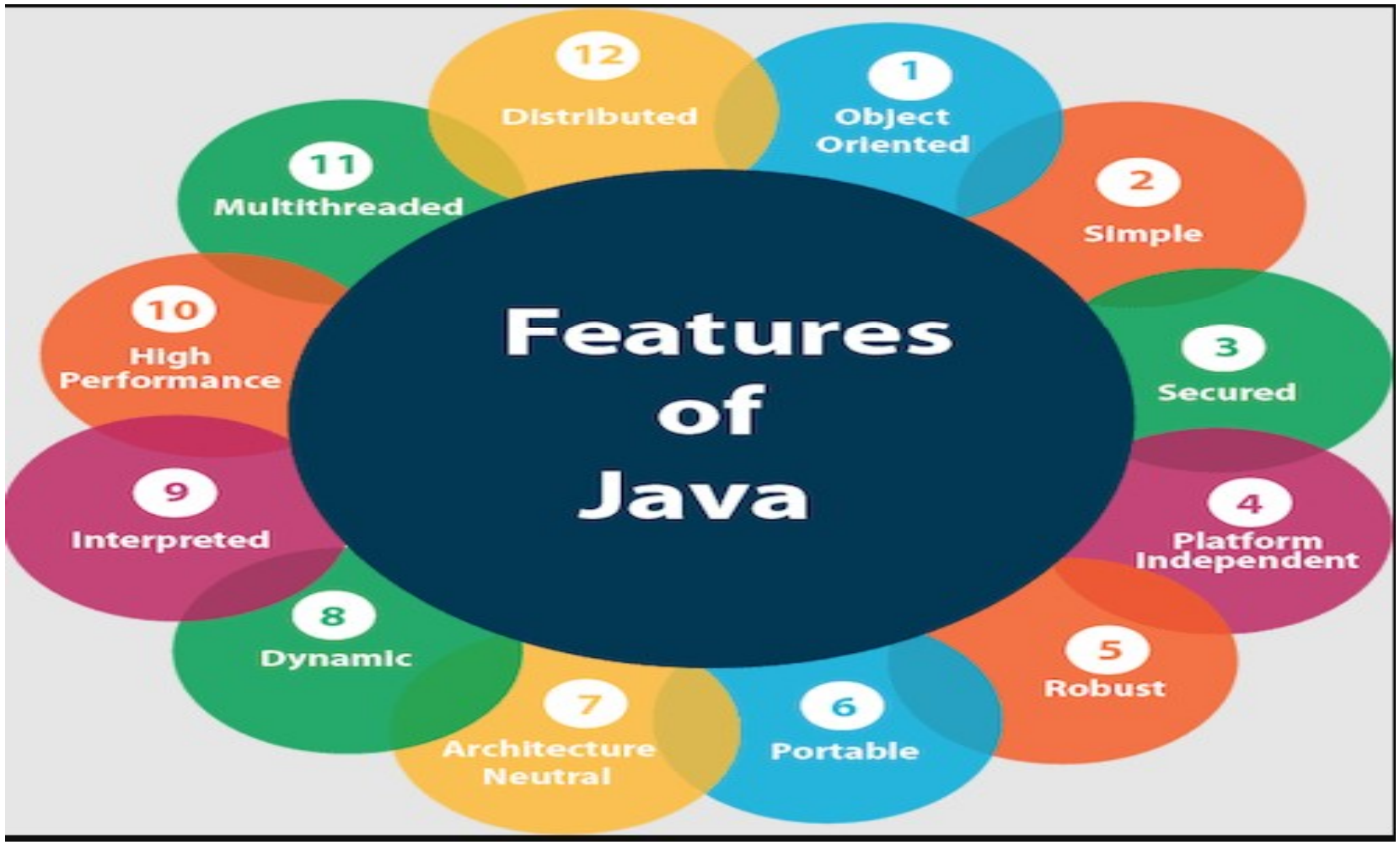
Java Programming-II

By:- Jagdish Chandra Pandey

H.O.D(I.T.), G.P. Dwarahat/ Officiating Principal, G.P. Chaunaliya

Features of java



Features of Java

1. Simple:- Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Features of Java

2. Object-oriented:-

Java is an object-oriented programming language. Basic concepts of OOPs are:-

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Features of Java

3. Platform Independent :-

1. Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

2. There are two types of platforms software-based and hardware-based. Java provides a software-based platform. The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:-

- Runtime Environment
- API(Application Programming Interface)

3. Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

Features of Java

4. Secured:- Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:-

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**
- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Features of Java

5. Robust:- The English meaning of Robust is strong. Java is robust because:-

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Features of Java

6. Portable:-

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

7. High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Features of Java

8. Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI (Remote Method Invocation) and EJB (Enterprise Java Bean) are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

9. Multi-threaded

- A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

9. Dynamic

- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

1-Dimensional arrays

1. The general form of 1-D array declaration is:-

   type var-name[];

   eg:- int month_days[];

   Although this declaration establishes that month_days is an array variable, no array actually exists. In fact, the value of month-days is set to null. To link month_days with an actual, physical array of integers, you must allocate one using new and assign it to month_days.

   i.e. array_var = new type[size];

   month_days = new int[12];

2. It is possible to combine the declaration of array variable with the allocation of array itself as:-

   int month_days[] = new int[12];

3. Arrays can be initialized when they are declared.

1-Dimensional arrays

```
class AutoArray
      {
      public static void main(String args[])
    {
       int months_days[] =
          { 31, 28, 31, 30, 31, 31, 30, 31, 30, 31};
     System.out.println("April has " +
      month_days[3] + "days");
    }
 }
```

Multi-Dimensional arrays

```
class TwoDArray
  {
    public static void main(String args[])
    {
      int twoD[][] = new int[4][5];
      int I,j,k = 0;
      for(i=0; i<4; i++)
       for(j =0; j<5; j++)
         twoD[i][j] = k;
           k++;
       for(i = 0; i<4; i++)
       {
         for(j = 0; j<5; j++)
          {
            System.out.println(twoD[i][j] + " ");
          }
         System.out.println();
       }
    }
  }
```

Output:-

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

Multi-Dimensional arrays

1. When you allocate memory for multidimensional array, you need only specify the memory for the first(leftmost) dimensions separately.

2. The advantage of individually allocating the second dimension arrays is that when we allocate dimensions manually we need not to allocate the same number of elements for each dimension. Since multidimensional arrays are actually arrays of arrays, the length of each array is under our control.

Multi-Dimensional arrays

```
class TwoDAgain
     {
    public static void main(String args [ ])
     {
      int twoD[][] = new int[4][];
      twoD[0] = new int[1];
      twoD[1] = new int[2];
      twoD[2] = new int[3];
      twoD[3] = new int[4];
  int i,j,k = 0;
  for(i=0; i<4; i++)
for(j = 0; j<i+1; j++)
{
twoD[i][j] = k;
K++;
}
for(i=0; i<4; i++)
{
for(j=0; j<i+1; j++)
{
System.out.print( twoD[i][j] + " ");
}
System.out.println();
}
}
}
```

Output

0
1 2
3 4 5
6 7 8 9

Alternative array declaration syntax

Second form that is used to declare an array:-

type [] var-name;

following two declaration are equivalent

```
int a[] = new int[3];
   ||
int [] a1 = new int[3];
```

Similarly

```
char two d1[][] = new char[3][4];
        ||
char [][] two d =  new char[3][4];
```

This alternative declaration form offers convenience when declaring several arrays at same time.

for ex:-

```
int[] num1, num2, num3;
        ||
int num1[], num2[], num3[];
```

Strings & Pointers

1. The string type is used to declare string variables.

2. eg:- String str = " this is a test";
   System.out.println (str);

3. Java does not support or allow pointers(or more precisely java does not support pointers that can be accessed and/or modified by programmers).

Class

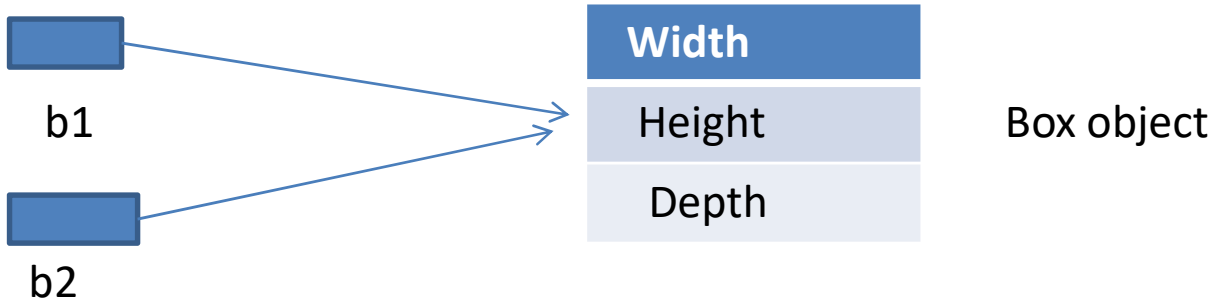
```
Class Box {
      double width;
      double height;
      double depth;
     }
Class BoxDemo {
      public static void main(String args[])
{
   Box mbox = new box();
    double vol;
    mybox.width = 10;
    mybox.height = 20;
    mybox.depth = 15;
  vol = mybox.width*mybox.height*mybox.depth;
   System.out.println("Volume is" +vol);
}
}
```

Note:-
The file that contain this program is called BoxDemo.java

Class

1. Class declaration and implementation of methods are stored in same place and not defined separately.

2. Box b1 = new Box();
   Box b2 = b1;
   After this fragment is executed b1 and b2 not refer to separate and distinct objects, 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 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.



3. Although b1 and b2 both refer to the same object they are not linked in any other way.
   for ex:- if b1 has been set to null i.e. b1 = null; then b2 still points to original object.

Difference between parameter and argument

- int square(int i)

```
{
   return i*i;
  }
int x = square(5);
```

A parameter is a variable defined by a method that receives a value when the method is called for ex:- in square(), i is a parameter whereas an argument is a value that is passed to a method when it is invoked, for ex:- square(5) passes 5 as an argument.

Constructor

```
class Box {
    double width;
     double height;
     double depth;
      Box()   //   Constructor function
          {
            System.out.println("Constructing Box");
             width = 10;
              height = 10;
               depth = 10;
              }
         double volume ()  //  method or function of class Box
            {
              return width*height*depth;
              }
               }

        class  BoxDemo6 {
              public static void main( String args[])
              {
                 Box mybox1 = newBox();
                  Box mybox2 = newBox();
                  double vol;
                  vol = mybox1.volume();
                  System.out.println("volume is " + vol);
                  vol = mybox2.volume();
                  System.out.println("volume is"  +vol);
}
}
```

Output:-

Constructing Box
Constructing Box
Volume is 1000.00
Volume is 1000.00

A constructor in Java is **a block of code similar to a method that's called when an instance of an object is created**. ... A constructor doesn't have a return type. The name of the constructor must be the same as the name of the class. Unlike methods, constructors are not considered members of a class.

Parameterized Constructor

```
class Box {
   double width;
    double height;
     double depth;
 Box( double w, double h, double d) // parameterized constructor
   {
     width = w;
      height = h;
      depth = d;
     }
double volume()
  {
     return width*height*depth;
    }
    }
class BoxDemo7
{
public static void main(String args[])
  {
      Box mybox1 = new Box(10,20,15);
      Box mybox2 = new Box(3,6,9) ;
      double vol;
      vol = mybox1.volume();
      System.out.println("volume is" +vol);
     vol = mybox2.volume();
     System.out.println("Volume is" +vol);
}
```

"this " keyword

We can have local variables, including formal parameters to methods , which overlap with the name of class instance variables. However when a local variable has the same name as instance variable, the local variable hides the instance variable. This is why width, height and depth were not used as the names of parameters to the Box() constructor inside the Box class. If they had been, then width would have referred to the formal parameter, hiding the instance variable width. But this lets you refer directly to object, we can use it to resolve any name-space collisions that might occur between instance variables and local variables.

for ex:-

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

Garbage Collection

1. Java handles deallocation for us automatically. The technique that accomplishes this is called garbage collection.

2. When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by object can be reclaimed.

3. Garbage collection only occurs spordically (if at all) during the execution of our program. It will not occur simply because one or more objects exist that are no longer used.

The “finalize()” method

1. Sometimes an object will need to perform some action when it is destroyed. for ex:- if an object is holding some non-java resource such as file-handle or character font, then we 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.

2. To add a finalizer to a class, you simply define the finalize() method. The java runtime calls that method whenever it is about to recycle an object of that class. Inside the finalize() method , you will specify those actions that must be performed before an object is destroyed.

3. The finalize() method has general form:-

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

4. finalize() is only called just prior to garbage collection. It is not called when an object goes out-of-scope i.e. we cannot know when or even if finalize() will be executed. Therefore our program should provide other means of releasing system resources etc. used by the object.

A stack class

```
class stack {
  int stck[] = new int[10];
  int tos;
  stack() {
   tos = -1;
   }
   void push(int item)
   {
     if (tos == 9)
       System.out.println(“stack is full”);
         else
           stck[++tos] = item;
     }
 int pop()
   {
    if (tos < 0)
   {
   System.out.println(“Stack underflow”);
   return 0;
   }
  else
        return stck [tos--];
    }
}

class Teststack
{
Public static void main(Strings args[])
{
 stack mystack1 = new stack();
 stack mystack2 = new stack();
 for (int i = 10; i < 20; i++)
   mystack1.push(i);
 for (int i = 10; i<20; i++)
    mystack2.push(i);
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());
}
}
```

Output:-

Stack in mystack1:
9
8
7
6
5
4
3
2
1

Stack in mystack2:
19
18
17
16
15
14
13
12
11
10

Returning objects

```
class Test
{
 int a;
 Test(int i)
{
 a= i;
}
Test incrbyten()
{
Test temp = new Test(a+10);
return temp;
}
}
class Retob
{
public static void main(String args[])
{
Test ob1 = new Test(2);
Test ob2;
ob2 = ob1.incrbyten();
System.out.println("ob2.a after second increases : + ob2.a);
}
}
```

Note:-

Since all objects are dynamically allocated using new we don't need to worry about an object going out-of-scope because the method in which it was created terminates. The object will continue to exist as long as there is a reference to it somewhere in our program.

Recursion

```
class Factorial
{
int fact(int n)
{
int result;
if (n == 1) return 1;
 result = fact(n-1)*n;
return result;
}
}
class Recursion
{
public static void main(String args[])
{
Factorial f = new Factorial();
System.out.println("Factorial of 3 is "  + f.fact(3));
}
}
```

1. Recursive versions of many routines may execute a bit more slowly than iterative equivalent because of added overhead of additional function calls.

2. Many recursive calls to a method could cause a stack overrun. Because storage for parameters and local variables is on stack and each new call creates a new copy of these variables, it is possible that stack could be exhausted.

3. The main advantage to recursive methods is that they can be used to create clearer and simpler version of several algorithm.

Continue……