Unit II

Class: syntax, instance variable, class variables, methods, constructors, overloading of constructors and methods. Arrays, Strings and Vectors.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. In Java everything is encapsulated under classes. Class is the core of Java language. Class can be defined as a template/ blueprint that describe the behaviors /states of a particular entity. A class defines new data type. Once defined this new type can be used to create object of that type. Object is an instance of class. You may also call it as physical existence of a logical template class.

A class is declared using **class** keyword. A class contain both data and code that operate on that data. The data or variables defined within a **class** are called **instance variables** and the code that operates on this data is known as **methods**. Thus, the instance variables and methods are known as class members. **class** is also known as a user defined datatype.

Rules for Java Class

- A class can have only public or default(no modifier) access specifier.
- It can be either abstract, final or concrete (normal class).
- It must have the class keyword, and class must be followed by a legal identifier.
- It may optionally extend one parent class. By default, it will extend java.lang.Object.
- It may optionally implement any number of comma-separated interfaces.
- The class's variables and methods are declared within a set of curly braces {}.
- Each **.java** source file may contain only one public class. A source file may contain any number of default visible classes.
- Finally, the source file name must match the public class name and it must have a .java suffix.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

Class & Object in Java	Variable in Java

class demo
{
void show() // method
{
System.out.println("hello");
}
public static void main (String args[])
{
demo d=new demo(); // create object
d.show(); // function call
}
}
	class demo
	{
	int a=10,b=20; // Global Var
	void adding()
	{
	int c; //local var
	c=a+b;
	System.out.println(c);
	}
	public static void main (String args[])
	{
	demo d=new demo();
	d.adding();
	}
	}

A class can contain any of the following variable types.

- **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables** − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class. A

variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

- **Class variables** − Class variables are variables declared within a class, outside any method, with the static keyword.

Method in Java

A **method** is a block of code which only runs when it is called.You can pass data, known as parameters, into a method.Methods are used to perform certain actions, and they are also known as **functions**.

Create a Method

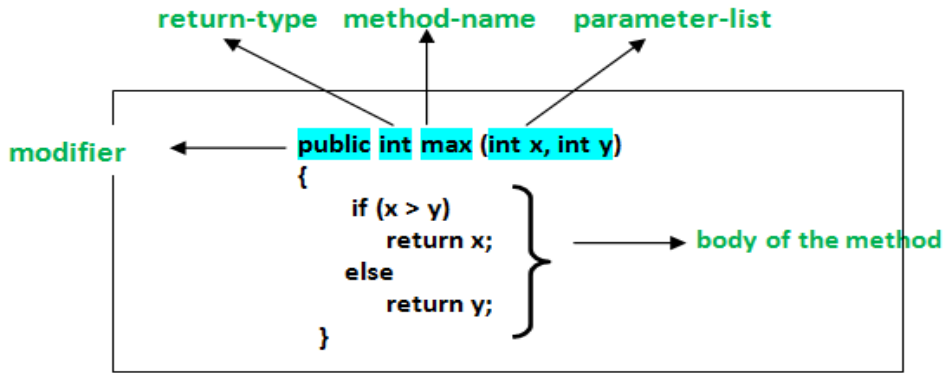
A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Call a Method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method.Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.



In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

No Arg No return	**Arg with No return**

class demo
{
void adding()
{
int a=20,b=30,c;
c=a+b;
System.out.println(c);
}
public static void main(String args[])
{
demo d=new demo();
d.adding();
}
}
	class demo
	{
	void adding(int a,int b)
	{
	int c;
	c=a+b;
	System.out.println(c);
	}
	public static void main(String args[])
	{
	demo d=new demo();
	d.adding(10,20);
	}
	}

Arg with Return	

class demo
{
void adding(int a,int b)
{
return a+b;
}
public static void main(String args[])
{
demo d=new demo();
System.out.println(d.adding(10,20));
}
}	

Constructor

a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Default Constructor	**Parameterized constructor**

class demo
{
demo()
{
System.out.println("Default Const.");
}
public static void main(String args[])
{
demo D=new demo();
}
}
	class demo
	{
	int a;
	demo(int x)
	{
	a=x;
	System.out.println(a);
	}
	public static void main(String args[])
	{
	demo D=new demo(100); } }

Constructor Overloading in Java

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Constructor overloading

```
class demo
{
int a,b;
demo() // Zero Argu
{
 a=b=0;
show();
}
demo(int x)  // Single Argu
{
a=b=x;
show();
}
demo(int x,int y)  // Multi Arg– Const. Overloading
{
a=x;
b=y;
show();
}
void show()
{
System.out.println(a+" "+b);
}
public static void main(String args[])
{
demo D1=new demo();
```

```
demo D2=new demo(10);
demo D3=new demo(10,20);
}
}
```

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.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Method overloading

```
class demo
{

void adding(int a,int b) // 2 argu
{
int c;
c=a+b;
System.out.println(c);
}

void adding(int a,int b,int c) // 3 argu
{
int d;
d=a+b+c;
System.out.println(d);
}
public static void main(String args[])
{
demo d1=new demo();
d1.adding(20,30);
d1.adding(10,20,30);
}
}
```

Array in Java

An array is a collection of similar data types. Array is a container object that hold values of homogenous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.

An array can be either primitive or reference type. It gets memory in heap area. Index of array starts from zero to size-1.

Features of Array

- It is always indexed. Index begins from 0.
- It is a collection of similar data types.
- It occupies a contiguous memory location.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

1 – D Array	2-D Array

class demo
{
public static void main(String args[])
{
int a[]={8,3,4,5};
for(int i=0;i<a.length;i++)
{
System.out.println(a[i]);
}
}
}
	class demo
	{
	public static void main(String args[])
	{
	int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
	for(int i=0;i<3;i++)
	{
	for(int j=0;j<3;j++)
	{
	System.out.print(arr[i][j]+" ");
	}
	System.out.println();
	}
	}
	}

Multi-Dimensional Array

A multi-dimensional array is very much similar to a single dimensional array. It can have multiple rows and multiple columns unlike single dimensional array, which can have only one full row or one full column.

Java Vector

Java Vector class comes under the java.util package. The vector class implements a growable array of objects. Like an array, it contains the component that can be accessed using an integer index.

Vector is very useful if we don't know the size of an array in advance or we need one that can change the size over the lifetime of a program.

Vector implements a dynamic array. It is similar to ArrayList, but with two differences −

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

The Vector class implements a growable array of objects. Vectors basically fall in legacy classes but now it is fully compatible with collections.

- Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index
- They are very similar to ArrayList but Vector is synchronised and have some legacy method which collection framework does not contain.
- It extends **AbstractList** and implements **List** interfaces.

Vector	Output

import java.util.*;
class demo
{
public static void main(String args[])
{
// create default vector
Vector v = new Vector();
 	
v.add(1);
v.add(2);
v.add("bca");
v.add("mca");
 	
System.out.println("Vector is " + v);
}
}
	[1, 2, bca, mca]