**JAVA Notes**

MSc-IT (2nd Semester)

Standard College of Girls

Compiled By: Umair Yaqub

Java is an Object-Oriented Language. As a language that has the Object Oriented feature, Java supports the following fundamental concepts:

* Polymorphism
* Inheritance
* Encapsulation
* Abstraction
* Classes
* Objects
* Instance
* Method
* Message Parsing

In this chapter, we will look into the concepts Classes and Objects.

**Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class. If we consider the real-world we can find many objects around us, Cars, Dogs, Humans, etc. All these objects have a state and behavior. If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging, running.

**Class -** A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

A sample of a class is given below:

public class Dog{

String breed;

int ageC;

String color;

void barking(){ }

void hungry(){ }

void sleeping(){ }

}

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.
* **Class variables:**Class variables are variables declared with in a class, outside any method, with the static keyword.

# Java - Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

**extends** is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

class Super{

.....

}

class Sub extends Super{

.....

.....

}

## Sample Code:

Below given is an example demonstrating Java inheritance. In this example you can observe two classes namely Calculation and My\_Calculation. Using extends keyword the My\_Calculation inherits the methods addition() and Subtraction() of Calculation class.

class Calculation{

int z;

public void addition(int x, int y){

z=x+y;

System.out.println("The sum of the given numbers:"+z);

}

public void Substraction(int x,int y){

z=x-y;

System.out.println("The difference between the given numbers:"+z);

}

}

Public class My\_Calculation extends Calculation{

public void multiplication(int x, int y){

z=x\*y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]){

int a=20, b=10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Substraction(a, b);

demo.multiplication(a, b);

}

}

## Types of inheritance:

There are various types of inheritance as demonstrated below.

# Java Tutorial

## IS-A Relationship:

IS-A is a way of saying : This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

public class Animal{ }

public class Mammal extends Animal{ }

public class Reptile extends Animal{ }

public class Dog extends Mammal{ }

Now, based on the above example, In Object Oriented terms, the following are true:

* Animal is the superclass of Mammal class.
* Animal is the superclass of Reptile class.
* Mammal and Reptile are subclasses of Animal class.
* Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say:

* Mammal IS-A Animal
* Reptile IS-A Animal
* Dog IS-A Mammal
* Hence : Dog IS-A Animal as well

With use of the extends keyword the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass.

## HAS-A relationship:

These relationships are mainly based on the usage. This determines whether a certain class **HAS-A**certain thing. This relationship helps to reduce duplication of code as well as bugs.

public class Vehicle{}

public class Speed{}

public class Van extends Vehicle{

private Speed sp;

}

This shows that class Van HAS-A Speed. By having a separate class for Speed, we do not have to put the entire code that belongs to speed inside the Van class., which makes it possible to reuse the Speed class in multiple applications.

## The super keyword:

The **super** keyword is similar to **this** keyword following are the scenarios where the super keyword is used.

* It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
* It is used to **invoke the superclass** constructor from subclass.

### Sample Code:

This section provides you a program that demonstrates the usage of the **super** keyword.

In the given program you have two classes namely *Sub\_class* and *Super\_class*, both have a method named display() with different implementations, and a variable named num with different values. We are invoking display() method of both classes and printing the value of the variable num of both classes, here you can observe that we have used super key word to differentiate the members of super class from sub class.

class Super\_class{

int num=20;

//display method of superclass

public void display(){

System.out.println("This is the display method of superclass");

}

}

public class Sub\_class extends Super\_class {

int num=10;

//display method of sub class

public void display(){

System.out.println("This is the display method of subclass");

}

public void my\_method(){

//Instantiating subclass

Sub\_class sub=new Sub\_class();

//Invoking the display() method of sub class

sub.display();

//Invoking the display() method of superclass

super.display();

//printing the value of variable num of subclass

System.out.println("value of the variable named num in sub class:"+ sub.num);

//printing the value of variable num of superclass

System.out.println("value of the variable named num in super class:"+ super.num);

}

public static void main(String args[]){

Sub\_class obj = new Sub\_class();

obj.my\_method();

}

}

# Java - Overriding

We talked about super classes and sub classes. If a class inherits a method from its super class, then there is a chance to override the method provided that it is not marked final.

The benefit of overriding is: ability to define a behaviour that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.

In object-oriented terms, overriding means to override the functionality of an existing method.

## Example:

Let us look at an example.

class Animal{

public void move(){

System.out.println("Animals can move");

}

}

class Dog extends Animal{

public void move(){

System.out.println("Dogs can walk and run");

}

}

public class TestDog{

public static void main(String args[]){

Animal a = new Animal(); // Animal reference and object

Animal b = new Dog(); // Animal reference but Dog object

a.move();// runs the method in Animal class

b.move();//Runs the method in Dog class

}

}

This would produce the following result:

Animals can move

Dogs can walk and run