INHERITANCE

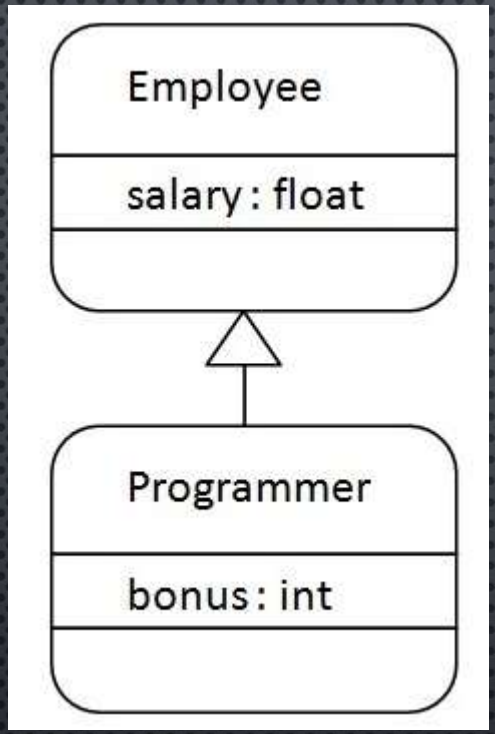
- **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviours of parent object.

- **Syntax of Java Inheritance**

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

- The **extends keyword** indicates that you are making a new class that derives from an existing class.
- The meaning of "extends" is to increase the functionality.

Java Inheritance Example



Programmer is the subclass and Employee is the superclass.
Relationship between two classes is **Programmer IS-A Employee**.
It means that Programmer is a type of Employee.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

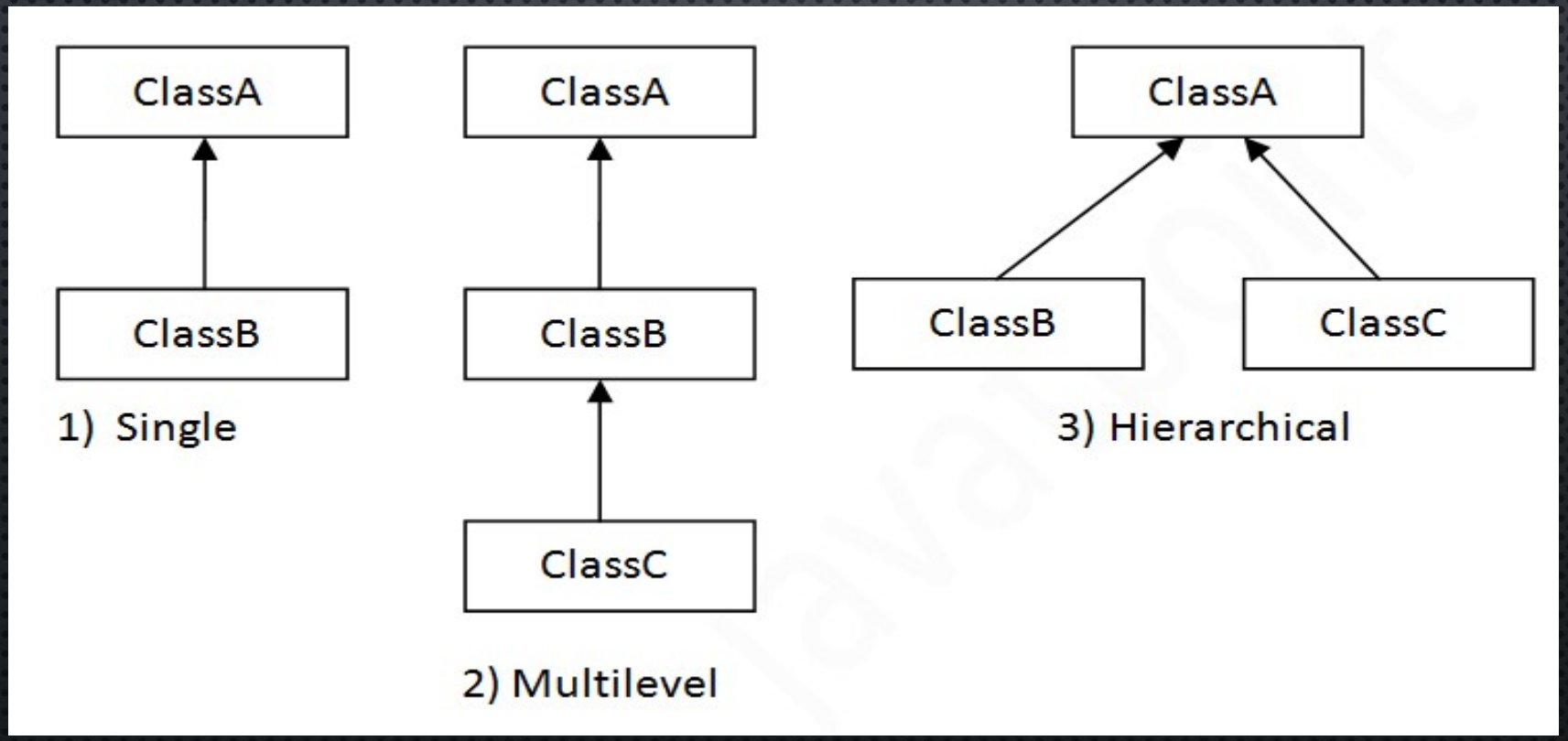
```
class Vehicle {
 String vehicleType;
 }
public class Car extends Vehicle {
 String modelType;
 public void showDetail() {
 vehicleType = "Car"; //accessing Vehicle class
member modelType = "sports";
System.out.println(modelType+" "+vehicleType);
 }
 public static void main(String[] args) {
 Car car =new Car();
 car.showDetail();
 }
 }
```

Types of inheritance in java

On the basis of class, there can be **three types of inheritance** in java:

single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.



```
class SingleInheritance{
int num1=10;
int num2=5;
}
public class MainInheritance extends
SingleInheritance{
 int num3=2;

 public static void main(String[] args){
 MainInheritance m=new MainInheritance();
int result=m.num1+m.num2+m.num3;
System.out.println("Result of child class is "+result);
 }
}
```

```
class WorkerDetail{
int code;
String name;
double salary;
double hra;
void setSalary(int c, String n, double s){
code=c;
name=n;
salary=s;
}
void showDetail(){
System.out.println("The employee code is "+code);
System.out.println("The employee name is "+name);
System.out.println("The employee salary is "+salary);
}
void getHRA(){
hra=salary*0.60;
System.out.println("HRA "+hra);
}
}
```

```
 class OfficerSal extends WorkerDetail{
double da;
void getDA(){
da=salary*0.98;
System.out.println("DA "+da);
}
}
class ManagerSal extends OfficerSal{
double ca,gross;
void getCA(){
ca=salary*0.20;
System.out.println("CA "+ca);
}
void getGross(){
gross=hra+da+ca+salary;
System.out.println("The gross salary is "+gross);
}
}
```

```
 public class MultiLevel{
public static void main(String args[]){
ManagerSal ms=new ManagerSal();
ms.setSalary(101,"Sreedar",75000);
System.out.println("Details of Manager is ");
ms.showDetail();
ms.getHRA();
ms.getDA();
ms.getCA();
ms.getGross();
}
}
```

```
class Fruit {
void fruitInfo() {
System.out.println("I am a fruit. ");
 }}
// class 'Mango' inherits from class 'Fruit‘
class Mango extends Fruit {
 public void mangoInfo() {
fruitInfo(); // calling base class function
System.out.println("My name is mango. ");
 }}
// class 'Apple' inherits from class 'Fruit‘
class Apple extends Fruit {
 public void appleInfo() {
 fruitInfo();// calling base class function
 System.out.println("My name is apple. ");   }}
public class HierarchicalInheritanceDemo {
 public static void main(String args[]) {
 Mango m = new Mango();
 Apple a = new Apple();
 m.mangoInfo();
 a.appleInfo();
 }}
```

super keyword

In Java, `super` keyword is used to refer to immediate parent class of a child class.

In other words **super** keyword is used by a subclass whenever it need to refer to its immediate super class.

```
class Parent {
 String name;
 }

 public class Child extends Parent {
 String name;

 public void details() {
 super.name = "Parent"; //refers to parent class member
 name = "Child";
 System.out.println(super.name+" and "+name);
 }

public static void main(String[] args) {
 Child cobj = new Child();
cobj.details();
 }
}
```

```
class Parent {
String name;
 public void details() {
 name = "Parent";
 System.out.println(name);
}
 }
 public class Child extends Parent {
String name;
 public void details() {
 super.details(); //calling Parent class details() method
 name = "Child";
 System.out.println(name);
 }
 public static void main(String[] args) {
 Child cobj = new Child();
 cobj.details();
 }
 }
```

```
class Parent {
 String name;
 Parent(String n) {
 name = n;
 }
 }
public class Child extends Parent {
String name;
 Child(String n1, String n2) {
 super(n1); //passing argument to parent class constructor
this.name = n2;
}
public void details() {
 System.out.println(super.name+" and "+name);
}
 public static void main(String[] args) {
 Child cobj = new Child("Parent","Child");
 cobj.details();
 }
 }
```

Note: When calling the parent class constructor from the child class using super keyword, super keyword should always be the first line in the constructor of the child class.

```
class Animal{
Animal(){
System.out.println("Animal is created");
}
}
class Dog extends Animal{
Dog(){
System.out.println("Dog is created");
}
}
class TestSuper{
public static void main(String args[]){
Dog d=new Dog();
}
}
```

```
class Person{
int id;
String name;
Person(int id,String name){
this.id=id;
this.name=name;
}
}

class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);  //reusing parent constructor
this.salary=salary;
}
void display(){System.out.println(id+" "+name+" "+salary);}
}

class TestSuper5{
public static void main(String[] args){
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}}
```

METHOD OVERRIDING

- **When a method in a sub class has same name, same number of arguments and same type signature as a method in its super class, then the method is known as overridden method.**
- **Method overriding is also referred to as runtime polymorphism.**

- **Advantage of method overriding**
- The main advantage of method overriding is that the class can give its own specific implementation to a inherited method **without even modifying the parent class code.**

- This is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code.

```
class Animal {
public void eat() {
 System.out.println("Generic Animal eating");
 } }
class Cat extends Animal{
//eat() method overridden by Cat class.
public void eat(){
System.out.println("Cat eats fish");
}
}
class Dog extends Animal {
 public void eat()
 //eat() method overridden by Dog class.
{ System.out.println("Dog eats meat");
 }
public static void main(String args[]){
Dog d=new Dog();
Cat c=new Cat();
d.eat();
c.eat();
}
 }
```

Method Overriding and Dynamic Method Dispatch

Method Overriding is an example of runtime polymorphism

When a parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method(parent class or child class) is to be executed is determined by the type of object.

This process in which call to the overridden method is resolved at runtime is known as dynamic method dispatch.

Q) What is Dynamic method dispatch?
Dynamic method dispatch is a mechanism by which a call to an overridden **method** is resolved at runtime. This is how **java** implements runtime polymorphism. When an overridden **method** is called by a reference, **java** determines which version of that **method** to execute based on the type of object it refer to.

```
class Animal {
public void eat() {
 System.out.println("Generic Animal eating");
 } }
class Cat extends Animal{
public void eat(){
System.out.println("Cat eats fish");
}
}
class Dog extends Animal {
 public void eat()
 //eat() method overridden by Dog class.
{ System.out.println("Dog eats meat");
 }
public void bark(){
System.out.println("Dog barks");
}
public static void main(String args[]){
Animal d=new Dog();
Animal c=new Cat();
d.eat();
c.eat();
d.bark();//gives compile time error
}
 }
```

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Difference between Overloading and Overriding

Method Overloading	Method Overriding
Parameter must be different and name must be same.	Both name and parameter must be same.
Compile time polymorphism.	Runtime polymorphism.
Increase readability of code.	Increase reusability of code.

RULES FOR METHOD OVERRIDING

- The argument list should be exactly the same as that of the overridden method.
- If the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden
- A subclass can override any superclass method that is not declared private or final.
- Constructors cannot be overridden.

Can we Override static method? Explain with reasons ?

No, we cannot override static method. Because static method is bound to class whereas method overriding is associated with object i.e at runtime.

FINAL KEYWORD IN JAVA

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- **variable**
- **method**
- **class**

Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike{
 final int speedlimit=90; //final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike obj=new  Bike();
 obj.run();
 }
}//end of class
```

o/p

Compile Time error

Java final method

If you make any method as final, you cannot override it.

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

o/p

Compile Time error

Java final class

If you make any class as final, you cannot extend it.

```
final class Bike{
  void run(){System.out.println("running...");}
}
class Honda extends Bike{
void run(){System.out.println("running safely with 100kmph");}
   public static void main(String args[]){
      Honda h= new Honda();
  h.run();

   }
}
```

Output: Compile Time Error

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it.

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda extends Bike{

   public static void main(String args[]){
      Honda h= new Honda();
  h.run();

   }
}
```

o/p

running…

Blank final variable

A final variable that is not initialized at the time of declaration is known as **blank final variable**.

We **must initialize the blank final variable in constructor** of the class otherwise it will throw a compilation error

```
class Demo{
final int MAX_VALUE; //Blank final variable
Demo(){ //It must be initialized in constructor
MAX_VALUE=100;
}
void myMethod(){
 System.out.println(MAX_VALUE);
}
 public static void main(String args[]){
Demo obj=new Demo();
 obj.myMethod(); } }
```