JAVA PROGRAMMING

Chap 3 : Inheritance

- Inheritance is an important pillar of OOP(Object-Oriented Programming).
- It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- In Java, inheritance means creating new classes based on existing ones.
- A class that inherits from another class can reuse the methods and fields of that class.
- In addition, you can add new fields and methods to your current class as well.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- Important terminologies used in Inheritance:
  - **Class:** Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
  - **Super Class/Parent Class/Base Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
  - **Sub Class/Child Class/Derived Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
  - **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

- Why do we need Inheritance in Java?
  - **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
  - **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which java achieves Run Time Polymorphism.
  - **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.
- **Syntax for inheritance**
- The **extends keyword** is used for inheritance in java. Using the extends keyword indicates you are derived from an existing class. In other words, "extends" refers to increased functionality.
- Syntax :

```
class derived_class extends base_class
{
  //methods and fields
}
```

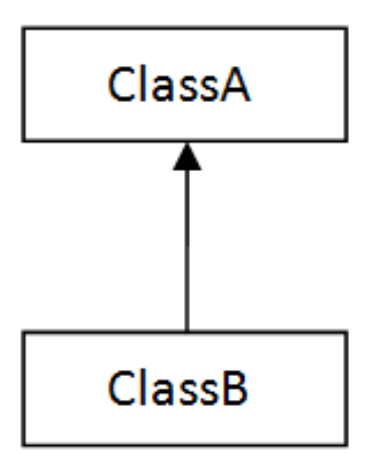
- Types of Inheritance :
  - Single Inheritance
  - Multilevel Inheritance
  - Hierarchical Inheritance

Single Inheritance

Here only one class is derived from another class.

```
import java.util.*;
class Data {                    // Base class Data
  float r;
  public void read(float x)
 {
   r=x;
 }}
class Area extends Data      // creating child
class Area
{
  private float a;
  public void calculate()
  {
    a=3.14f*r*r;
  }
  public void display()
  {
    System.out.println("Area is : "+a);
  }}
```

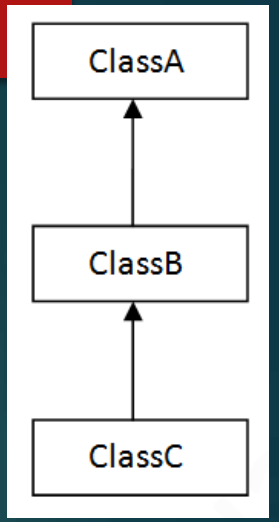
```
class Main
{
  public static void main(String args[])
  {
    float rad;
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Radius : ");
    rad=sc.nextFloat();
    Area ar=new Area();
    ar.read(rad);
    ar.calculate();
    ar.display();
  }}
```




```
Enter Radius :
10
Area is : 314.0
```

Multilevel Inheritance

Here one class is derived from a class which is in turn derived from another class.




```
import java.util.*;
class Data {                    // Base class Data
  float r;
  public void read(float x)
 {
   r=x;
 }}

class Area extends Data        // creating
child class Area of parent class Data
{
  protected float a;
  public void calculate()
  {
    a=3.14f*r*r;
  }
  public void display()
  {
    System.out.println("Area is : "+a);
  }}
```

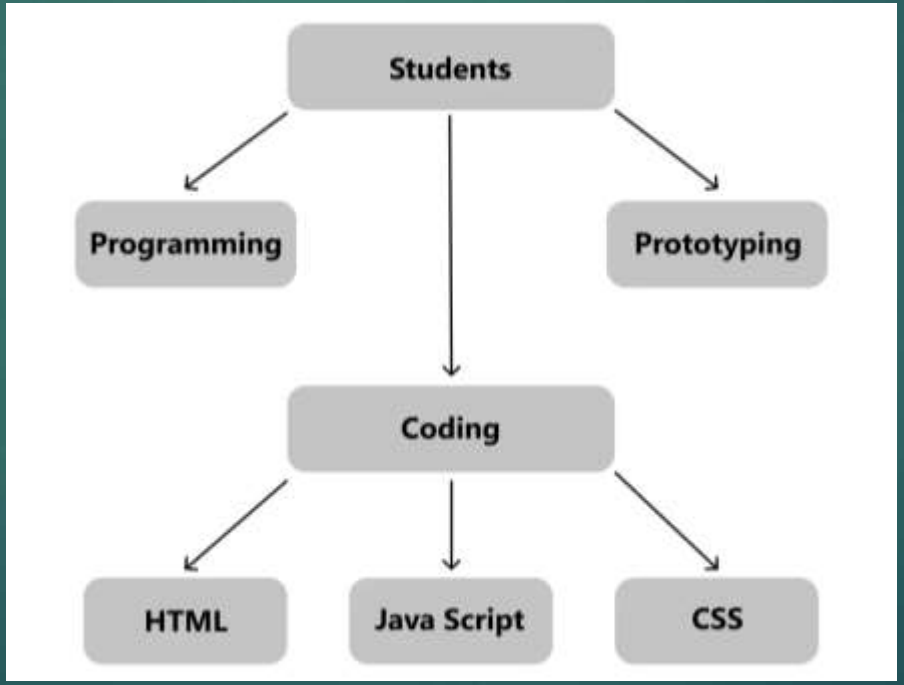
```
class Volume extends Area    // creating child
class Volume of parent class Area
{
  private float vol;
  public void compute()
  {
    vol=a*r*4/3;
  }
  public void show()
  {
    System.out.println("Volume is : "+vol);
  }}

class Main
{
  public static void main(String args[])
  {
    float rad;
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Radius : ");
    rad=sc.nextFloat();
```

```
Volume v=new Volume();
    v.read(rad);
    v.calculate();
    v.display();
    v.compute();
    v.show();
  }
}
```

```
Enter Radius :
10
Area is : 314.0
Volume is : 4186.6665
```

Hierarchical Inheritance

- When multiple classes are derived from a class and further more classes are derived from these derived classes
- one class serves as a superclass (base class) for more than one subclass
- Example from codes



Hierarchical Inheritance

```
// parent class
class Employee {
 double salary = 50000;

 void displaySalary() {
  System.out.println("Employee Salary:
Rs."+salary);
 }
}

// child class 1
class FullTimeEmployee extends
Employee{
 double hike = 0.50;

 void incrementSalary() {
   salary = salary + (salary * hike);
   System.out.println("Full Time Employee
Salary after increement : "+salary);
 }
}
```

```
// child class 2
class InternEmployee extends Employee{
 double hike = 0.25;

 void incrementSalary() {
   salary = salary + (salary * hike);
   System.out.println("Intern Salary after increement : "+salary);
 }}

class Main {
 public static void main(String[] args) {
   // object created
   FullTimeEmployee emp1 = new FullTimeEmployee();
   InternEmployee emp2 = new InternEmployee();
   System.out.println("Salary of a full-time employee before
incrementing:");
   emp1.displaySalary();
   emp1.incrementSalary(); // salary incremented of Full time Employee
   System.out.println("Salary of an intern before incrementing:");
   emp2.displaySalary();
   emp2.incrementSalary();  // salary incremented of Intern
    }}
```

Hierarchical Inheritance

```
Salary of a full-time employee before incrementing:
Employee Salary: Rs.50000.0
Full Time Employee Salary after increement : 75000.0
Salary of an intern before incrementing:
Employee Salary: Rs.50000.0
Intern Salary after increement : 62500.0
```

Method Overriding

- If a base class and derived class have a method with same name but different parameter list, then it is called as method overloading.
- But, **If a base class and derived class have a method with same name and same parameter list, then it is called as method overriding.**
- Usage of Java Method Overriding
  - Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
  - Method overriding is used for runtime polymorphism
- Rules for Java Method Overriding
  - The method must have the same name as in the parent class
  - The method must have the same parameter as in the parent class.
  - There must be an IS-A relationship (inheritance).

Method Overriding

```
class Bank{
int getRateOfInterest()
{
return 0;
}}

class SBI extends Bank{
int getRateOfInterest()
{
return 8;
}}

class ICICI extends Bank{
int getRateOfInterest()
{
return 7;
}}

class AXIS extends Bank{
int getRateOfInterest()
{
return 9;
}}
```

```
class Main{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}}
```

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Final class and “final” keyword

- The final keyword can be preceded to any member of a class or to the class itself.
- Making anything final has following implications –
  - If the field member is declared final then the variable value cannot be changed i.e. it becomes constant.
  - If a method is declared as final then that method cannot be overridden
  - If a class is declared as final then that class cannot have any sub class i.e. no class can be derived from a final class

Final class and “final” keyword

- **Final variable :** If you make any variable as final, you cannot change the value of final variable(It will be constant).
- Example of final variable : There is a final variable speedlimit, we are going to change the value of this variable, but it can't be changed because final variable once assigned a value can never be changed.

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

```
Main.java:4: error: cannot assign a value to final variable speedlimit
    speedlimit=400;    // changing value of speedlimit
    ^
1 error
```

Final class and “final” keyword

- **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()                  // overriding final method run()
   {
      System.out.println("running safely with 100kmph");
   }
}
  class Main{
   public static void main(String args[]){
   Honda h= new Honda();
   h.run();
   }
}
```

```
Main.java:10: error: run() in Honda cannot override run() in Bike
   void run()                  // overriding final method run()
        ^
  overridden method is final
1 error
```

Final class and “final” keyword

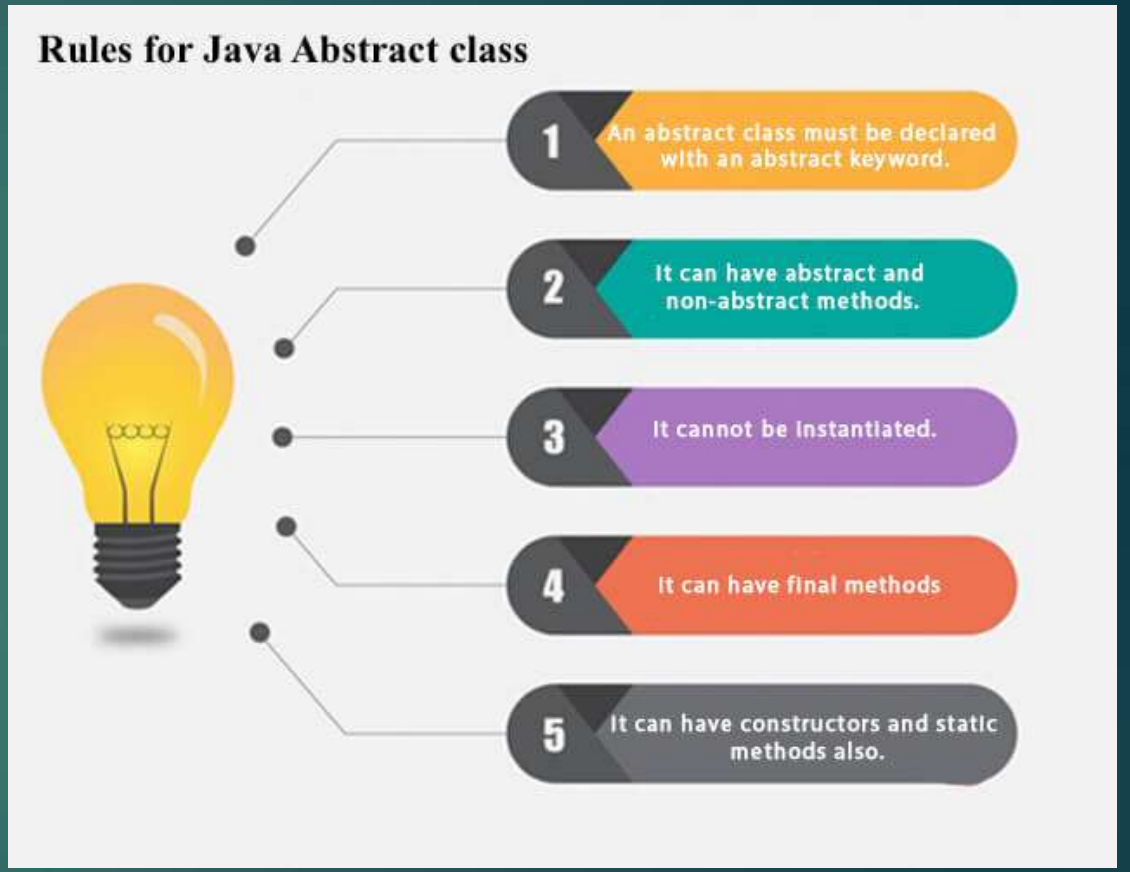
- **Final class :** If you make any class as final, you cannot extend it.

```
final class Bike{
}

 class Honda extends Bike{          // inheriting from final class bike
   void run()
   {
       System.out.println("running safely with 100kmph");

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

```
Main.java:4: error: cannot inherit from final Bike
 class Honda extends Bike{
                     ^
1 error
```

Java Abstract class and method

- Abstract classes are used to declare common characteristics of subclasses.
- Abstract classes are declared with keyword ***abstract*** followed by class definition. They provide template for subclasses.
- ***No object can be made of abstract class. It can be used as a base class for other classes that are derived from the abstract class.***
- An abstract class can contain fields and methods. It can have abstract and non-abstract methods (method with the body).
- ***Methods of abstract class that has only declaration and no definition*** **are known as** ***Abstract Methods.***
- ***Abstract methods must be overridden.***
- If a class has any abstract method, the class becomes abstract and must be declared as abstract.



Java Abstract class and method

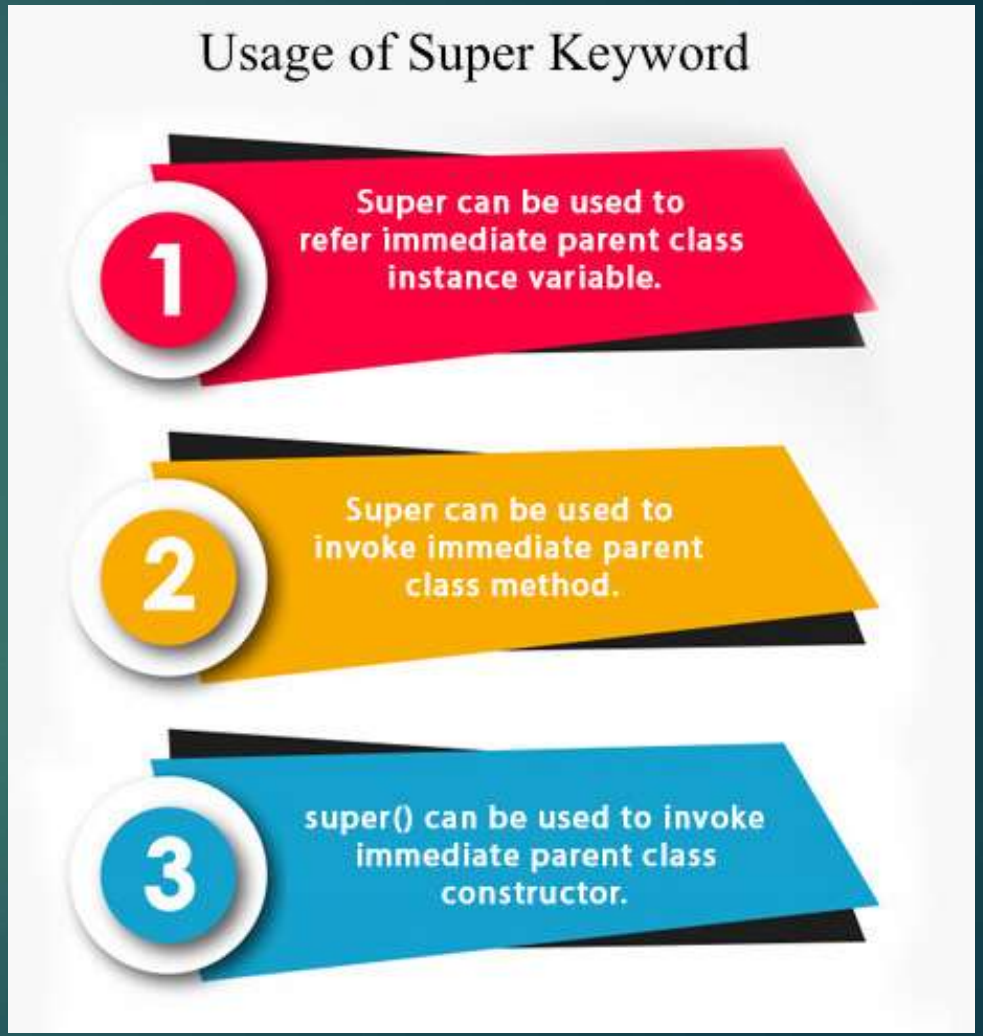
```
import java.util.*;
abstract class Abst {          //abstract class
  protected float r,vol;
  public void read(float x)  // non-abstract method
 {
   r=x;
 }
 public abstract void calculate();   // abstract method
 public void display()
 {
     System.out.println("Volume = "+vol);
 }}

class Sphere extends Abst {
    public void calculate()     // overriding abstract
method
    {
        vol=3.14f*r*r*r*4/3;
    }}

class Hemisphere extends Abst {
    public void calculate()    // overriding abstract method
    {
        vol=3.14f*r*r*r*2/3;
    }}
```

```
class Main
{
  public static void main(String args[])
  {
    float rad;
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Radius : ");
    rad=sc.nextFloat();
    Sphere s=new Sphere();
    s.read(rad);
    s.calculate();
    System.out.println("For Sphere : ");
    s.display();
    Hemisphere h=new Hemisphere();
    h.read(rad);
    h.calculate();
    System.out.println("For Hemisphere : ");
    h.display();
}
}
```

```
Enter Radius :
10
For Sphere :
Volume = 4186.6665
For Hemisphere :
Volume = 2093.3333
```

The *super* keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- If you want to access a member of a base class from the derived class, then the ***super*** keyword is used.
- This is especially to access the constructors and methods of base class.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.



The *super* keyword

```
class Parent{
public Parent(int a)
{
   System.out.println("Inside the
constructor of Parent class : "+a);
}

public void display(int x)
 {
    System.out.println("x = "+x);
 }
}
```

```
class Child extends Parent{
public Child(int a)
{
   super(a);
   System.out.println("Inside the
constructor of Child class : "+a);
}

public void display(int y)
{
   super.display(y);
   System.out.println("y = "+y);
}}
```

```
class Main{
public static void main(String args[])
 {
Child c = new Child(3);
c.display(5);
  }
}
```

```
Inside the constructor of Parent class : 3
Inside the constructor of Child class : 3
x = 5
y = 5
```

Programming Practice Questions

Design a class hierarchy for a library system. Create a base class *Book* with attributes *title* and *author*, and a constructor that initializes these attributes. Then, create a derived class *Ebook* that extends Book with additional attributes *file_size* and *format* and *Printed_book* that extends Book with attributes *edition* and *no of copies sold*. Ensure that the constructor of Ebook and Printed_book properly initializes attributes from both the base class and the derived class. (*use super keyword*)

Create an abstract class *Appliance* with abstract methods *switch_on()* and *switch_off()*. Derive two classes *WashingMachine* and *Refrigerator* from Appliance and implement the abstract methods with specific behaviors for each appliance. Ensure that the abstract class cannot be instantiated directly and that the derived classes provide concrete implementations for the abstract methods.

Design a class hierarchy for a university system. Create a base class *Person* with attributes *name* and *age*, and methods like *getDetails() and display_details()*. Then, create derived classes *Student* and *Professor* that inherit from Person and add specific attributes and methods. For example, *Student* might have a *studentID* and *major* field, while *Professor* might have a *facultyID*, *salary* and *department*. Override getDetails() and display_details() methods to input and display all the details of student and professor.

Implement a class hierarchy for a banking system. Create a base *class Account* with attributes *accountNumber* and *balance*, a constructor to initialize these attributes and methods *deposit(), withdraw() and getBalance()*. Create a derived class *SavingsAccount* that adds an *interestRate* attribute and provides its own constructor that initializes all attributes. Also calculate the interest and add it to the balance using deposit method().

Programming Practice Questions

- You are tasked with designing a system for a university that includes different types of people: *Professor, Student, and Staff*. All these types of people share some common attributes but also have their unique attributes. You need to use *hierarchical inheritance* to model this system, ensuring that constructors are properly used to initialize the base and derived classes. Create a Base Class *Person* having Fields *name and id* and a constructor to initialize them. It will also have *getdetails() and display()* methods to input and display the details. Create Derived Classes *Professor, Student and Staff* that inherits from Person. *Professor* has an additional field *department*. Initialize name, id, and department using the base class constructor. Class *Student* has additional field *major and gpa*. Initialize name, id, major, and gpa using the base class constructor. Class *Staff* has additional field *designation*. Initialize name, id, and position using the base class constructor. Create instances of Professor, Student, and Staff classes and print their details using *display()* method.
- Design a class Hierarchy for a zoo. Create a base *class Animal* which has data members *Name and Species*. It has methods *getHabitat()* to return default habitat description and *display()* to display the details of the animal. *Class Mammal* is derived from Animal which has field *hasFur* of type *Boolean*. A *class Lion* inherits class Mammal and has data member *endangered* of type *Boolean*. Override *display() and getHabitat()* methods to display animal details and Habitat description of Mammal as well as Lion. Use appropriate constructors to initialize all the data members.
- Design an abstract class `Character` for a game to serve as the base class for various types of characters. Each character type has specific actions but shares common functionality such as name, attacking and defending. This class accepts the `character_name` through `constructor` and has abstract methods `attack()` and `defend()` that defines how the character attacks or defends. `Class Warrier` and `Mage` inherits class Character and overrides attack() method to display what weapon is used by the character to attack (eg: sword, fireball, axe, javelin, etc.) and defend() method to display the technique used by the character to defend (eg: uses shield, uses magic spell, counter attack, etc.).