Inheritance: Extending classes for reusability

In the World of programming, various concepts draw inispiration from real life examples, and inheritance is no execution.

Inheritance in real Life: Let's begin by examining the real-life example of Mukesh Ambani, who dedicated decades of hard working and invested millions of dollars in building the renowned Reliance Industries.

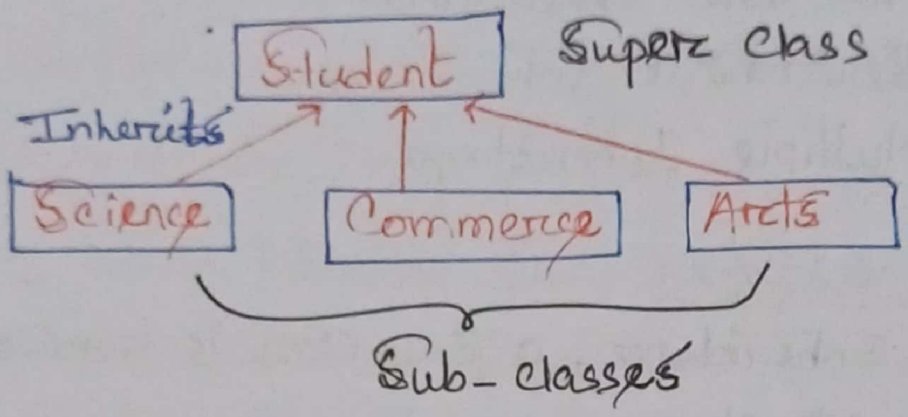
As time passes, there comes a point when Mukesh Ambani decides to retire. However, he has a son who will inherit the Reliance industry from his father. This means that the son doesn't have to start from scratch to build a company; instead, he can leverage what his father has already built ~~a company~~ and create his legacy upon it.

Understanding Inheritance in OOP:

Now we have a real life example to relate to, we can delve into the concept of inheritance in OOP. Inheritance allows us to establish relationships between classes, enabling the reuse of variables and functions from one class into another.

3. Hierarchical Inheritance

In this sort of inheritance, multiple subclass derived from single super class.



Code:

```
Class Student {
    public void methodStudent()
    {
        SOP("Student");
    }
}
Class Science extends Student {
    public void methodScience()
    {
        SOP("Science");
    }
}
Class Commerce extends Student {
    public void methodCommerce()
    {
        SOP("Commerce");
    }
}
```

```
class Arts extends Student {
    public void methodArts()
    {
        SOP("Arts");
    }
}
class Test {
    public static void main(String args[]){
        Science sci = new Science();
        Commerce com = new Commerce();
        Arts art = new Arts();

        sci.methoStudent();
    }
}
```

4. Multiple Inheritance

Java does not support multiple inheritances due to ambiguity. For example, consider the following program:

Code:

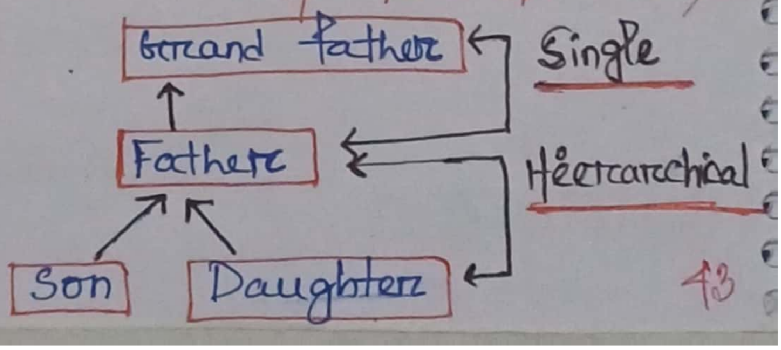
```
class Wishes {
    void message(){
        SOP("Hi");
    }
}
```

```
class Birthday {
        void message() {
            SOP("Happy Birthday");
        }
}
public class Demo extends Wishes, Birthday
{
        public static void main (String args){
            Demo ob = new Demo();
            // Can't decide which classes
            // message() method will be invoked.
            ob.message();
        }
}
```

The above code gives error because the compiler can't decide which message method is invoked.

* Hybrid Inheritance

Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.



Super Keyword

→ Super Keyword is used to refer immediate Super class object.

⊛ Usage of Super Keyword

1. It is used to call Super class instance Variable
2. It is used to Call Super Class Method.
3. It is used to Call Super class Constructor.

1. Super is used to refer immediate parent Class instance Variable

```
public class Variable{
        int num = 59;
}
public class Check extends Variable{
        int num = 60;
        void display(){
                SOPln(num);
                SOPln(super.num);
        }
}
```

2. Super can be used to invoke parent class method.

public class

```
public class A {
        void Display() {
            SOP("Inside A class");
        }
}
public class B extends A {
        @Override
        Void Display() {
            SOP("Inside B class");
            Super. Display();
        }
}
public class Test {
        public Static void main (String[] args) {
            B obj = new B();
            obj. Display();
}
```

3. Super is used to invoke parent class Constructor

```
Class Animal {
        Animal(){
            SOP("animal is created");
        }
}
Class Dog extends Animal {
        Dog(){
            Super();
            SOPln("Dog is Created");
        }
}
Class Test {
    public Static void main(String args[]){
        Dog d = new Dog();
    }
}
```

Method Overriding

* What is method overriding?

→ Declaring a method in subclass which is already present in superclass is known as method overriding.

* Why do we need method overriding?

- Code reuse
- Run time polymorphism.

* What are the rules for method overriding?

1. ~~Name~~ The method must have the same name as in the parent class
2. The method must have the same parameters as in the parent class.
3. A method declared as final or static can't be overridden.
4. Constructor can't be overridden.

Code:

```
public class Person{
        String name;
        int age;
        void Display(){
                Sopln(name);
                Sopln(age);
        }
}
public class Teacher extends Person{
        String qualification;
        @Override
        void Display(){
                Sopln(name);
                Sopln(age);
                Sopln(qualification);
        }
}
public class Test{
        public static void main(String[] args){
                Teacher t1 = new Teacher();
                t1.name = "Jalal";
                t1.age = 21;
                t1.qualification = "ABC";
                t1.Display();
        }
}
```

* Can We override Static method ?

→ No, Static method Can't be overridden. Because, Static method is bound to class, on the other hand method is Bound to objects.

* Can We override Java main method ?

→ No, because main is a Static method.

Final Keyword

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

1. Variable
2. method
3. class.

1. Final Variable

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

Code :

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

Output : Compile Time Error.

2. Final Method

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

Code:

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

Output: Compile time Error.

3. Final Class

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

* Is final method inherited ?

- Yes, final method is inherited but we cannot override it.