Introduction

- The mechanism of deriving a new class from an old one is called *Inheritance*.
- The old class is known as the '***Base Class'*** or '***Super Class'*** or '***Parent Class'.***
- The new class is called '***Sub-Class'*** or '***Derived Class'*** or '***Child Class'***.
- Inheritance allows sub-class to inherit all the variables and methods of their parent class.
- Inheritance represents the IS-A relationship, also known as '*Parent-Child'* relationship.

Syntax

class *subclassname* extends *superclassname*
{
    variable declaration;
    method declaration;
}

- The keyword extends signifies that the properties of the superclassname are extended to the subclassname
- The subclass will now contain its own variables and methods as well those of the superclass.

Types of Inheritances

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance

Single Inheritance

- Only one super class.

Example

```
class A
{
        …….
        …….
}
class B extends A
{
        …….
        …….
}
```

Example Program: single.java

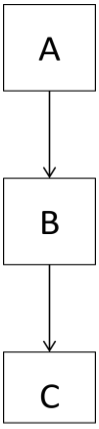
```
class A
{
        void displayA()
        {
                System.out.println("I am A class function");
        }
}
class B extends A
{
        void displayB()
        {
                System.out.println("I am B class function");
        }
}
class single
{
        public static void main(String args[])
        {
                B obj1=new B();
                obj1.displayA();
                obj1.displayB();
        }
}
```

Output

I am A class function
I am B class function

Multilevel Inheritance

- Derived from a derived class.



Example

```
class A
{
        …….
        …….
}
class B extends A
{
```

```
        …….
        …….
}
class C extends B
{
        …….
        …….
}
```

Example Program: multilevel.java

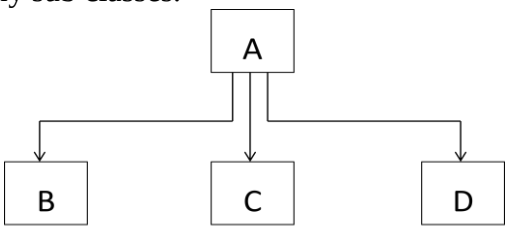
```
class A
{
        void displayA()
        {
                System.out.println("I am A class function");
        }
}
class B extends A
{
        void displayB()
        {
                System.out.println("I am B class function");
        }
}
class C extends B
{
        void displayC()
        {
                System.out.println("I am C class function");
        }
}
class multilevel
{
        public static void main(String args[])
        {
                B obj1=new B();
                C obj2=new C();
                obj1.displayA();
                obj1.displayB();
                obj2.displayA();
                obj2.displayB();
                obj2.displayC();
        }
}
```

Output

```
I am A class function
I am B class function
I am A class function
I am B class function
I am C class function
```

Hierarchical Inheritance

- One super class, many sub classes.



Example

```
class A
{
        …….
        …….
}
class B extends A
{
        …….
        …….
}
class C extends A
{
        …….
        …….
}
class D extends A
{
        …….
        …….
}
```

Example Program: hierarchical.java

```
class A
{
        void displayA()
        {
                System.out.println("I am A class function");
        }
}
class B extends A
{
        void displayB()
        {
                System.out.println("I am B class function");
        }
}
class C extends A
{
        void displayC()
        {
                System.out.println("I am C class function");
        }
```

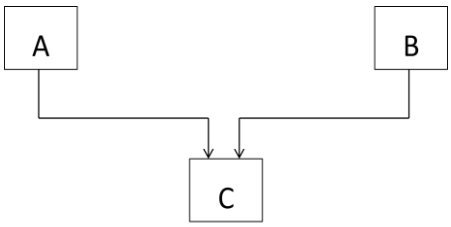
```
}
class hierarchical
{
        public static void main(String args[])
        {
                B obj1=new B();
                C obj2=new C();
                obj1.displayA();
                obj1.displayB();
                obj2.displayA();
                obj2.displayC();
        }
}
```

Output

```
I am A class function
I am B class function
I am A class function
I am C class function
```

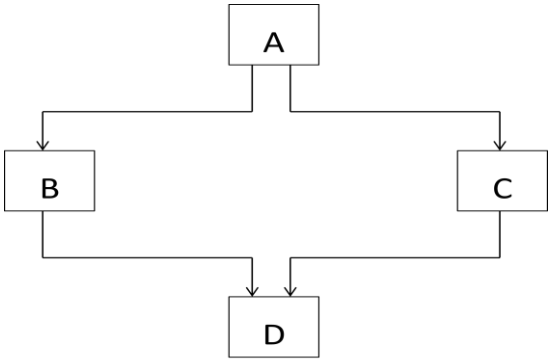
Multiple Inheritance

- Several super classes.
- Java does not directly implement multiple inheritance.
- Multiple inheritance is supported through interface only.



Hybrid Inheritance

- Combination of two or more types of inheritance.
- Java does not directly implement hybrid inheritance.
- Hybrid inheritance is supported through interface only.



Why multiple inheritance is not supported in java?

- Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A & B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.
- Since compile time errors are better than runtime errors.
- So, in order to reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Method Overriding

- In OOP, overriding means to override the functionality of an existing method.
- If subclass has the same method as declared in the parent class, it is known as method overriding.
- It is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is an example of Runtime Polymorphism.

Rules of Method Overriding

- Method must have same name as in the parent class.
- Method must have same arguments as in the parent class.
- If a method cannot be inherited, then it cannot be overridden.

Example Program: overriding.java

```
class A
{
	void display()
	{
		System.out.println("I am A class function");
	}
}
class B extends A
{
	void display()
	{
		System.out.println("I am B class function");
	}
}
class overriding
{
	public static void main(String args[])
	{
		A obj1=new A();
		obj1.display();
	}
}
```

Output

```
I am A class function
```

Method Overloading

- If a class have multiple methods by same name but different parameters, it is known as Method Overloading.
- It is used when objects are required to perform similar tasks but using different parameters.
- It is used to increase the readability of the program.
- Method overloading is an example of Compile Time Polymorphism.

Example Program: overriding.java

```
class A
{
	void add(int x,int y)
	{
		int result=x+y;
		System.out.println("Result = "+result);
	}
	void add(int x,int y,int z)
	{
		int result=x+y+z;
		System.out.println("Result = "+result);
	}
	void add(String x,String y)
	{
		String result=x+y;
		System.out.println("Result = "+result);
	}
}
class overloading
{
	public static void main(String args[])
	{
		A obj1=new A();
		obj1.add(6,4);
		obj1.add(5,6,4);
		obj1.add("Good ","Morning");
	}
}
```

Output

```
Result = 10
Result = 15
Result = Good Morning
```

Method Overloading Vs Method Overriding

Method Overloading	Method Overriding
It is used to increase the readability of the program.	It is used to provide the specific implementation of the method that is already provided by its parent class.

Method Overloading	Method Overriding
It is performed within class.	It is performed in two classes that have IS-A (inheritance) relationship.
Method parameter must be different.	Method parameter must be same.
It is an example of Compile Time Polymorphism.	It is an example of Run Time Polymorphism.
Method return type can be same or different.	Method return type must be same.

Questions asked in semester paper

Question-How run time polymorphism is achieved in Java?

[2011-2012]

Question-What do you mean by function overloading? Explain with examples.

[2006-2007]