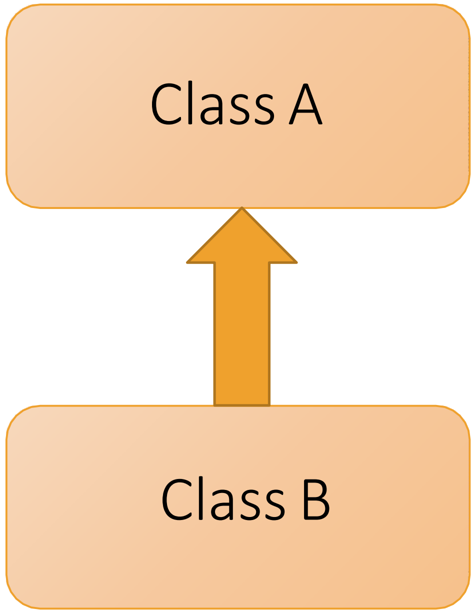
Unit 4
Inheritance and Interfaces

What is Inheritance?

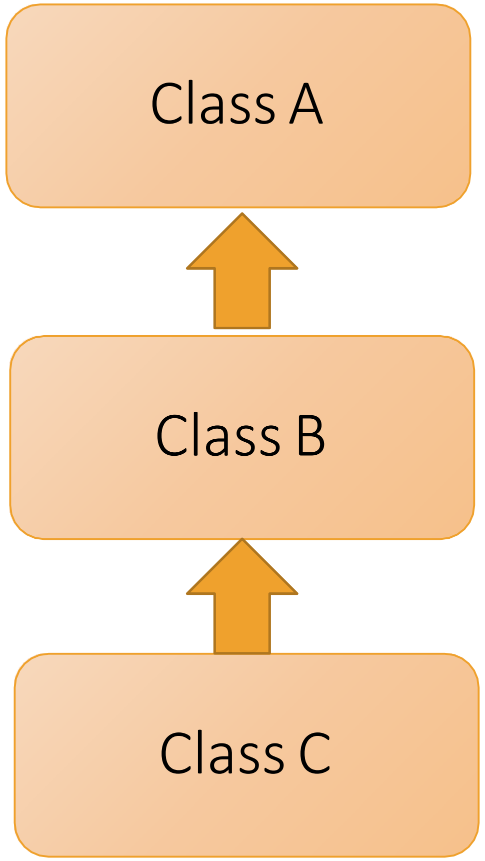
- It is a mechanism in which one class object acquires all properties of parent class.
- Syntax:

```
class parent
{
.....
}
class child extends parent
{
....
}
```

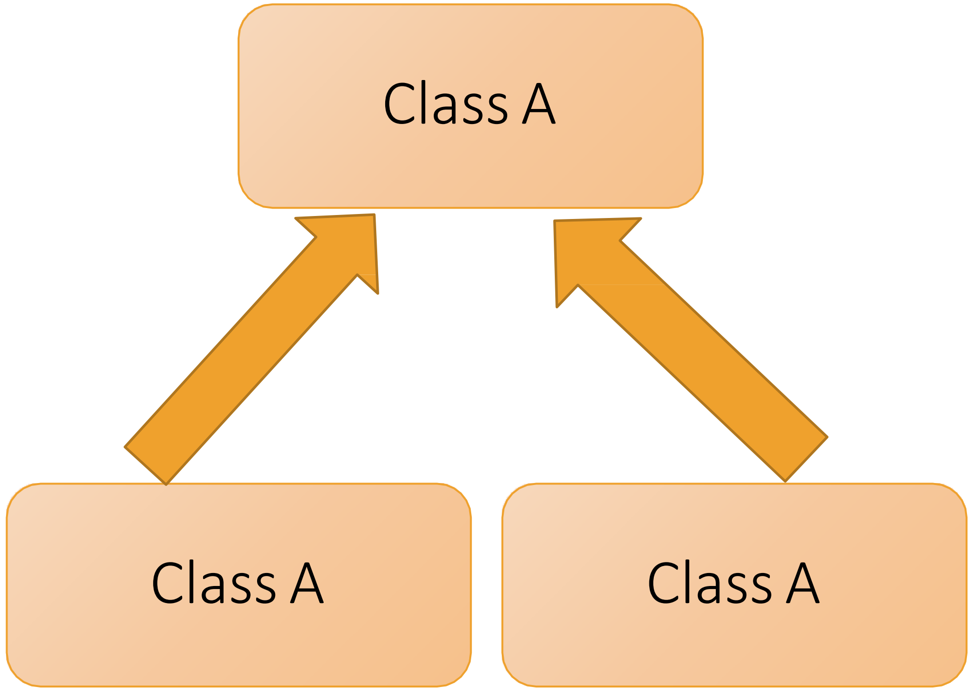
Types of Inheritance



Single Inheritance



Multi Level Inheritance



Hierarchical Inheritance

Syntax:

Single Inheritance

```
Class A
{
// Base class
}
Class B extends A
{
// Derived Class
}
```

Multilevel Inheritance

```
Class A
{
// Base class
}
Class B extends A
{
// Derived Class of A
}
Class C extends B
{
// Derived class of B
}
```

Hierarchical Inheritance

```
Class A
{
// Base class
}
Class B extends A
{
// Derived Class
}
Class C extends A
{
//Derived class
}
```

Example of Single Inheritance

```
class Faculty {
String collegeName = “MEFGI";
String designation = “Assit.prof";

   void Print(){
          System.out.println("Teaching");
   }
}
class JavaFaculty extends Faculty
 {
   String Subject = “JAVA";
```

```
public static void main(String args[]){

JavaFaculty obj = new JavaFaculty();
System.out.println(obj.Subject);

System.out.println(obj.collegeName);
System.out.println(obj.designation);
          obj.Print();
//Sub class Object Inherits Properties and
Behavior of Base Class
  }
}
```

Example of Multilevel Inheritance

```
class Faculty {
String collegeName = “MEFGI";
void Print(){
            System.out.println("Teaching");
}}
class JavaFaculty extends Faculty
 {
  String Subject = “JAVA";
}
class JavalabFaculty extends JavaFaculty
{
String lab= “Lab-8";
}
```

```
public class Demo
{
public static void main(String args[]){

JavalabFaculty obj = new JavalabFaculty();
System.out.println(obj.lab);

System.out.println(obj.collegeName);
System.out.println(obj.Subject);
obj.Print();
//Sub class Object Inherits Properties and
Behavior of Base Classes
   }
}
```

Example of Hierarchical Inheritance

```
class Faculty {
String collegeName = “MEFGI";
void Print(){
            System.out.println("Teaching");
}}
class JavaFaculty extends Faculty
 {
  String Subject1 = “JAVA";
}
class OODPFaculty extends Faculty
{
String Subject2= “OODP";
}
```

```
public class Demo
{
public static void main(String args[]){
OODPFaculty obj = new OODPFaculty();

System.out.println(obj.collegeName);
System.out.println(obj.Subject2);
obj.Print();
//Sub class Object Inherits Properties and
Behavior of Base Class
   }
}
```

Super Keyword

The **super** keyword in Java is used to refer immediate parent class object.

Use of super Keyword

- Used to refer immediate parent class instance variable.
- Used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

Example

- Refer immediate parent class instance variable.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
}
class Puppy extends Dog
{
String color="brown";
```

```
void printColor(){
System.out.println(color);// prints brown
System.out.println(super.color);//prints black
System.out.println(((Animal)this).color);//prints
white
 }}


class Demo{
public static void main(String args[]){
Puppy p=new Puppy();
p();
}}
```

Example

- Invoke immediate parent class method.

```
class Animal{

void eat()

{System.out.println("eating...");}

}

class Dog extends Animal{

void eat()

{System.out.println("eating bread...");}

super.eat();

}
```

```
class Demo{

public static void main(String args[]){

Dog d=new Dog();

d.eat();

}}
```

Constructor and Inheritance

For Example:

```
 class Base {
   Base() {
   System.out.println("Base Class ");
   }
 }
  class Derived extends Base {
 Derived() {
 System.out.println("Derived Class ");
   }
 }
```

```
public class Main {
public static void main(String[] args)
{
    Derived d = new Derived();
     //Automatically call Base class Constructor
  }
}
```

Output:

```
Base Class
Derived Class
```

In java constructor of base class is default (without argument) then it will automatically called by derived class.

Constructor and Inheritance

```
class Base {
  int x;
  Base(int x1) {
    x = x1;
  }
}

class Derived extends Base {
  int y;
  Derived(int x1, int y1) {
    super(x1);
    y = y1;
  }
  void Display() {
    System.out.println("x = "+ x +"y = "+y);
  }
}

public class Main {
  public static void main(String[] args) {
    Derived d = new Derived(100, 200);
    d.Display();
  }
}
```

If constructor is parameterized then use "super()" to call parameters of base class to derived class.

But super() call must be a first line of derived class constructor.

Method Overriding

- Declaration of method in sub class which is already present in Super class is known as method overriding.

Rules for Method Overriding:

- Private and Final methods can not be overridden .
- The overriding method must have same Argument List and Return type .
- Static Method can not Override but Sub class can have re-declaration of Static Method.
- The access modifier for an overriding method can allow more, but not less access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private
- It is Used to Achieve Run Time Polymorphism.

Example (Method Overriding)

```
class Animal{
void eat()
{
System.out.println("eating...");}
}
class Dog extends Animal{
void eat()
{
System.out.println("eating bread...");
} }
```

```
class Demo{
public static void main(String args[]){
Animal a = new Animal();
a.eat();
Dog d=new Dog();
d.eat();
}}
```

Example

```
class Parent
{
void f1()
{
System.out.println("Parent Class");
}
}
class Child extends Parent
{
void f1()
{
super.f1();
System.out.println("Child Class");
}
```

```
public static void main(String arg[])
{
Child obj1 = new Child();
obj1.f1();


}
```

```
Output:
Parent Class
Child Class
```

Dynamic Method Dispatch (Run Time Polymorphism)

Process of accessing members of derived class through the object of Base class is called Dynamic Method Dispatch.

For Example:

```
class Parent
{
void f1()
{
System.out.println(“Parent Class”);
}
}
class Child extends Parent
{
void f1()
{
System.out.println(“Child Class”);
}
```

```
Public static void main(String arg[])
{
Parent obj1 = new Child();

//Object of base class but instantiate with Base Class

obj1.f1();

}
```

Output:
Child Class

Multiple Inheritance

- Multiple inheritance means class can inherit properties from more than one class.

But java doesn’t support Multiple Inheritance.

Why?

To simplify and to reduce complexity, java doesn’t support multiple inheritance.

For example:

class Z inherits properties of class X and class Y. if both class having same method and we want to call both same method which are in different base class, from child class then it will create ambiguity.

Multiple Inheritance

How to solve problem of multiple inheritance in Java?

Using Interface.

Interface

It is a blueprint of class or it is like an abstract class which contains abstract methods with empty bodies and static constants.

For Example:

```
interface  human
{
Public void sleeping();   //method which has no body.
Public void eating();
}
```

Interface key Points

1) We can’t instantiate an interface

2) Interface provides full abstraction as none of its methods have body.

3) “implements” keyword is used by classes to implement an interface.

4) Implementation in class of any method of an interface, it must be mentioned as public.

5) Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.

6) All the interface methods are by default abstract and public.

8) Variables declared in interface are public, static and final by default.

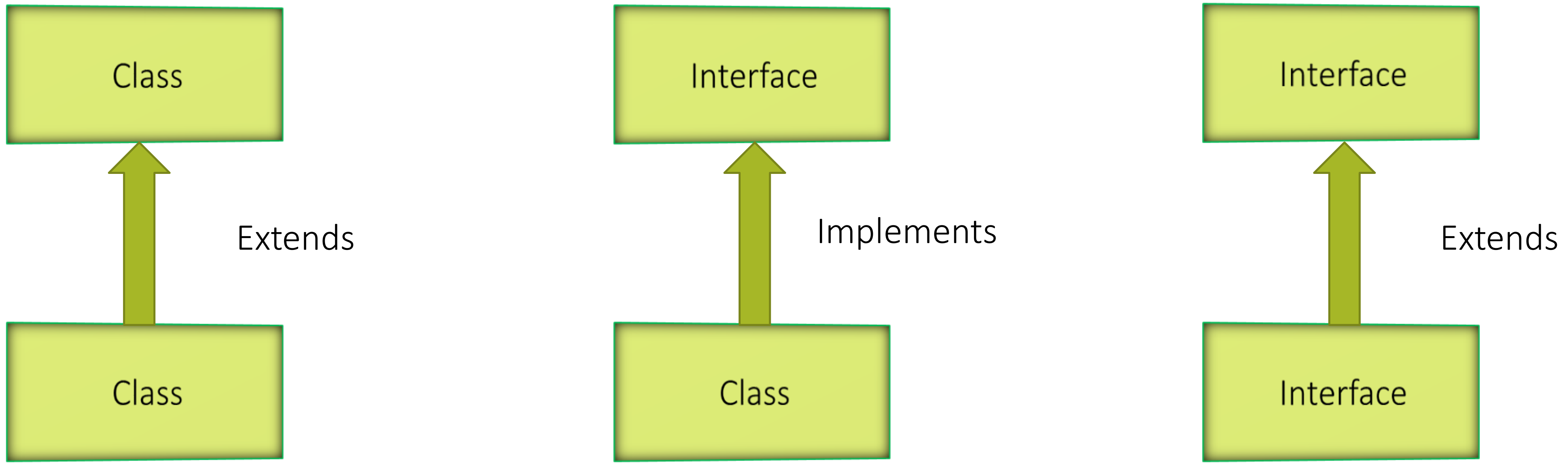
9) Interface variables must be initialized at the time of declaration otherwise throw error.

10) A class can implement any number of interfaces.

11) Variable names conflicts can be resolved by interface name.

12) A class cannot implement two interfaces that have methods with same name but different return type.

Relation between interface and class



Example

```
interface Test
{
void fun();   //method is by default public and abstract
}
class Demo implements Test
{
public void fun()
{            System.out.println("Hello");              }

public static void main(String args[])
{
Demo obj = new Demo();
obj.fun();
 }
}
```

Multiple Interface

```
interface Girl{
void f1();
}
interface Boy{
void f2();
}
class  Demo implements Girl,Boy
{
public void  f1(){System.out.println(“GIRL");}
public void  f2(){System.out.println(“BOY");}
```

```
public static void main(String args[])
{
Demo Obj=new Demo();
Obj.f1();
Obj.f2();
 }
}
```

Interface Inheritance

One Interface extends another Interface.

For Example:

```
Public interface Vehicle
{
void movefwd();
void movebwd();
}
Public interface Aircraft extends Vehicle
{
void up();
void down();
}
```

Interface Vehicle is used for all vehicle but if we want it for Aircraft also then we need to add two methods (up and down).

So, Aircraft extends Vehicle Interface.

Final Variable

It is used to prevent modification in a variable. A final variable can not modified after initialization.

For Example:

```
final int a=50;
a=100;  //Compilation error
```

Final Method

For Example:

```
class A
{
   final void f1()
   {          System.out.println(“Final Method");
   }
}
class B extends A
{
   void f1()
   {
             // Compile-error!
            System.out.println(“Not Allowed");
   }
}
```

It is used to prevent method overriding.

If method is final then derived class can not use same method.

Final Class

It is used to prevent Inheritance.

If class is final then it can’t extended by any other derived class.

For Example:

```
final class A
{
    // attributes and behaviors
}
class B extends A  // Compile-error!
{
}
```

This vs Super

This keyword	Super keyword
It keeps reference of same class	It keeps reference of base class
It is used to distinguish same class variable and local variable.	It is used to distinguish base class variable and derived class variable.
This() is used to call same class constructor	Super() is used to call base class constructor
It can be used to call same class method in case of method overloading.	It can be used to base class method in case of method overriding.

Method Overloading and Method Overriding

Method Overloading	Method Overriding
Multiple methods with same name in same class is called as method overloading	Methods having same name in base class as well as in derived class is called as method overriding.
Signature of method must be different	Signature of method may be same.
“this” is used to call specified method	“super” is used to call base class method.
It is concept of polymorphism	It is concept of inheritance.

Interface and Abstract Class

Interface	Class
Have all abstract method	Have abstract as well as normal method
Variables are static and final	Variables may normal
It implements class	It extends class
No Constructors	Have Constructors
It can not have base class	It can have base class
Can derive interface	Can’t derive interface

Questions??