Object Oriented Programming with Java (Subject Code: BCS-403)

Unit 1

Lecture 8

Lecture 8

- **Inheritance**
- **Super Class**
- **Sub Class**
- **Access Specifies**

Inheritance in Java

- 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.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

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.

Super Class/Parent Class:

The class whose features are inherited is known as a superclass(or a base class or a parent class).

Sub Class/Child Class:

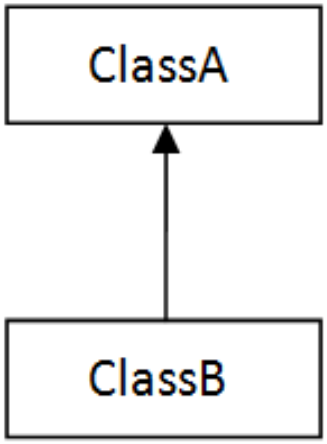
The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

```
import java.io.*;
 // Base or Super Class
class Employee {
    int salary = 60000;
}
 // Inherited or Sub Class
class Engineer extends Employee {
    int benefits = 10000;
}
```

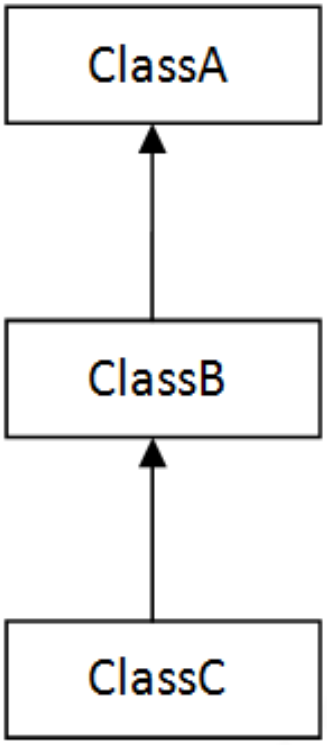
```
class MyMain {
    public static void main(String args[])
    {
        Engineer E1 = new Engineer();
        System.out.println("Salary : " + E1.salary
                        + "Benefits : " + E1.benefits);
    }
}
```

```
Salary : 60000
Benefits : 10000
```

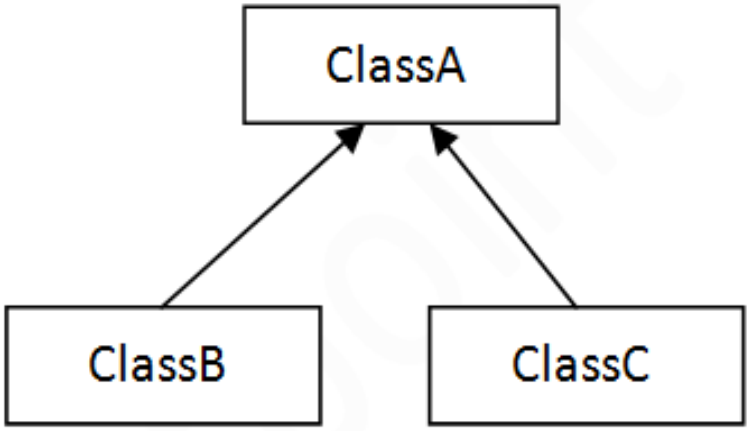
Types of inheritance in java



1) Single



2) Multilevel



3) Hierarchical

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, 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 and 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.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Compile Time Error

Java Modifier Types

Modifiers are keywords that you add to those definitions to change their meanings. The Java language has a wide variety of modifiers, including the following:

- Java Access Modifiers
- Non Access Modifiers

Access Control Modifiers:

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

The four access levels are:

- Visible to the package. the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Private access modifier

The private access modifier is accessible only within class.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

- If you make any class constructor private, you cannot create the instance of that class from outside the class.

Note: A class cannot be private or protected except nested class.

default access modifier

- If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.

```
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
```

protected access modifier

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```
Output:Hello

public access modifier

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Access Modifier	within class	within package	outside package by subclass only
Private	Y	N	N
Default	Y	Y	N
Protected	Y	Y	Y
Public	Y	Y	Y

Non Access Modifiers:

Java provides a number of non-access modifiers to achieve many other functionality.

- The *static* modifier for creating class methods and variables
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.