

# ECAP615

## Programming in Java



Harjinder Kaur  
Assistant Professor

# Learning Outcomes



After this lecture, you will be able to

- learn the basic concept Inheritance,
- understand the different types of access specifiers,
- implementation of different types of inheritance.

# Access Specifiers

- There are two types of modifiers in Java:
  - ✓ access specifiers.
  - ✓ non-access specifiers.
- The access modifiers in Java specify the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and classes by applying the access modifier on them.

# Access Specifiers

- For classes, you can use either public or default.
- For attributes, methods and constructors, you can use the one of the following:
  - ✓ public
  - ✓ Private
  - ✓ Default
  - ✓ protected

# Access Specifiers

Access Modifier	Within class	Within package	Outside package by subclass only	Outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

# Private Access Specifier

```
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  
    }  
}
```

# Default Access Specifier

```
package pack;  
  
class A{  
  
    void msg(){System.out.println  
("Hello");}  
  
}
```

```
package mypack;  
import pack.*;  
class B{  
  
    public static void main(Stri  
ng args[]){  
        A obj = new A();  
        //Compile Time Error  
        obj.msg();  
        //Compile Time Error  
    }  
}
```

# Protected Access Specifier

```
package pack;  
  
public class A{  
  
    protected void msg()  
    {  
  
        System.out.println("Hello  
");  
    }  
}  
  
package mypack;  
import pack.*;  
  
class B extends A{  
  
    public static void main(String  
    args[]){  
  
        B obj = new B();  
        obj.msg();  
    }  
}
```

# Public Access Specifier

```
package pack;  
  
public class A{  
  
    public void msg()  
    {  
  
        System.out.println("Hello"  
    };  
  
}
```

```
package mypack;  
import pack.*;  
  
class B{  
    public static void main(String  
args[])  
    {  
        A obj = new A();  
        obj.msg();  
    }  
}
```

# Inheritance

- It is the mechanism in java by which one class is allowed to inherit the features of another class.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the parent class.

# Inheritance

- Inheritance represents the IS-A relationship which is also known as a *parent-child* relationship.

Syntax:

```
class Subclass-name extends Superclass-name
```

```
{
```

```
    //methods and fields
```

```
}
```

# Important Terms

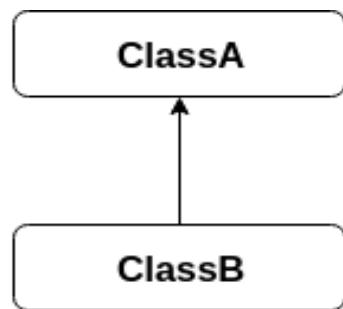
Class

Sub Class/Child Class

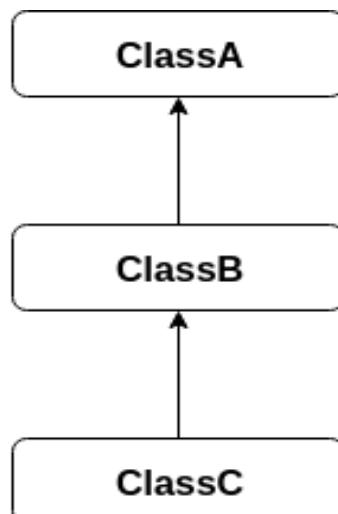
Super Class/Parent Class

Reusability

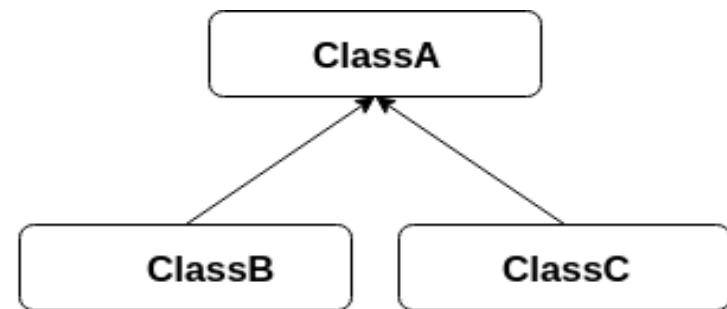
# Types of inheritance



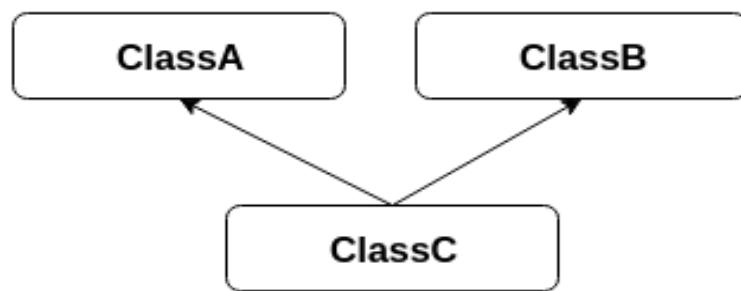
Single Inheritance



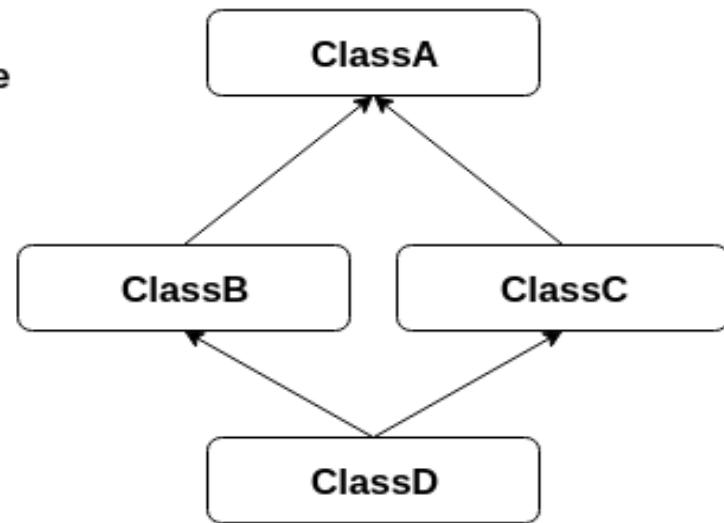
Multilevel Inheritance



Hierarchical Inheritance

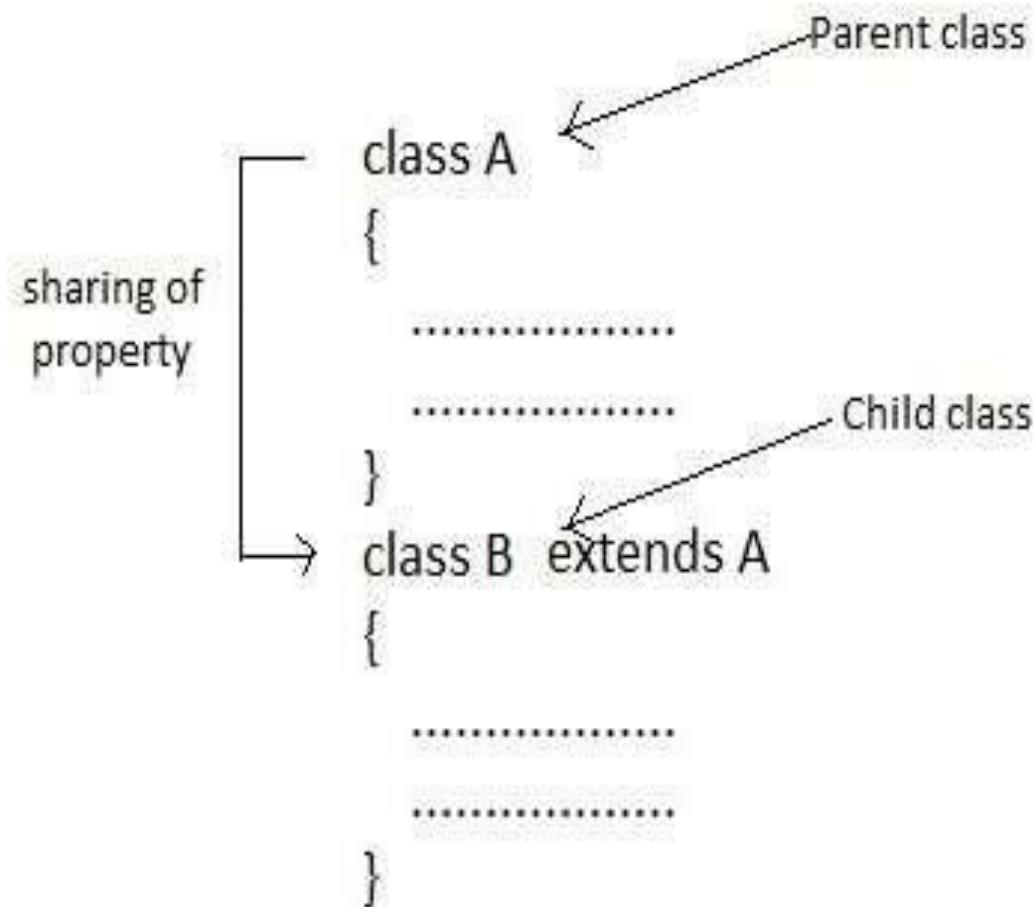


Multiple Inheritance



Hybrid Inheritance

# Single Inheritance



# Single Inheritance Example

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

```
class TestInheritance  
{  
public static void main(String arg  
s[])  
{  
Dog d=new Dog();  
d.bark();  
d.eat();  
}  
}
```

# Multilevel Inheritance

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

Diagram illustrating Multilevel Inheritance:

- Parent class:** Class A is the Parent class.
- Intermediate class:** Class B and Class C are Intermediate classes.
- Child class:** Class D is the Child class.

The code shows a hierarchy where Class B inherits from Class A, and Class C inherits from Class B. Class D inherits from Class C. The inheritance relationship is indicated by arrows pointing from the child class to the parent class in the code structure.

# Multilevel Inheritance Example

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

```
class Dog extends Animal{  
    void bark()  
    {System.out.println("barking...");}  
}
```

```
class BabyDog extends Dog{  
    void weep()  
    {System.out.println("weeping...");}  
}
```

```
class TestInheritance2{  
    public static void main(String  
args[])  
    {  
        BabyDog d=new BabyDog();  
        d.weep();  
        d.bark();  
        d.eat();  
    }  
}
```

# Multilevel Inheritance

```
class A {  
    .....  
}  
class B extends A {  
    .....  
}  
class C extends A {  
    .....  
}  
.....  
=====  
}  
.....  
=====  
class D extends A {  
    .....  
}  
=====  
}  
  
Parent class  
Child class  
Child class  
Child class
```

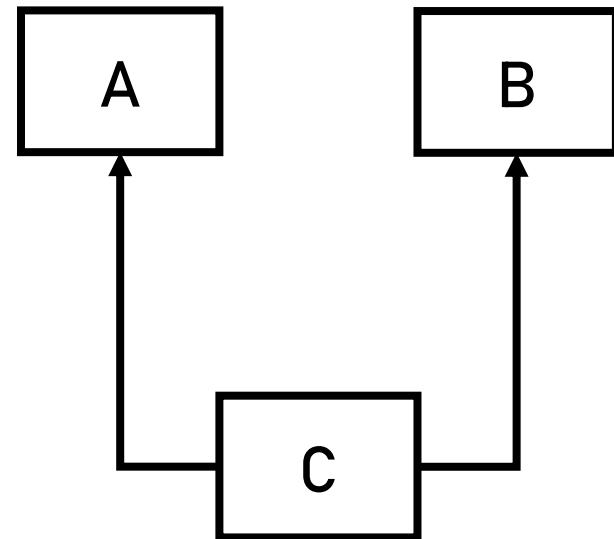
# Hierarchical Inheritance Example

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

```
class Cat extends Animal{  
    void meow()  
    {System.out.println("meowing...");}  
}  
  
class TestInheritance3{  
    public static void main(String args[])  
    {  
        Cat c=new Cat();  
        c.meow();  
        c.eat();  
    }  
}
```

# Multiple Inheritance

- When the child class extends from more than one superclass, it is known as multiple inheritance.
- However, Java does not support multiple inheritance.
- To achieve multiple inheritance in Java, we must use the interface.



Multiple Inheritance

# Multiple Inheritance using Interface

```
interface AnimalEat {  
    void eat();  
}  
  
interface AnimalTravel {  
    void travel();  
}  
  
class Animal implements AnimalEat,  
AnimalTravel  
{  
    public void eat() {  
        System.out.println("Animal is eating");  
    }  
}
```

```
public void travel() {  
    System.out.println("Animal is  
travelling");  
}  
}  
  
public class Demo {  
    public static void main(String  
args[]) {  
        Animal a = new Animal();  
        a.eat();  
        a.travel();  
    }  
}
```

# Hybrid Inheritance

- Hybrid Inheritance is a combination of both Single Inheritance and Multiple Inheritance.
- Since java doesn't support multiple inheritances with classes, hybrid inheritance is also not possible with classes.
- In java, we can achieve hybrid inheritance only through Interfaces.

# Important facts about inheritance

**Default superclass.**

**Superclass can only be one.**

**Inheriting Constructors.**

**Private member inheritance.**

**That's all for now...**