



ሐምሌ, 2013 ዓ.ም

Object Oriented Programming



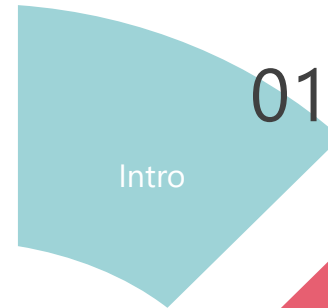
Chapter Three

Wollo University, Kombolcha Institute of Technology.
College of Informatics.
Department of Information System.

By Daniel G.

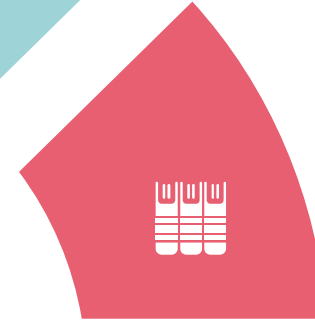
Contents

Object Oriented Programming
chapter 3 will include this topics.



01

Concept of inheritance



02

Super classes and
subclasses



03

Protected members



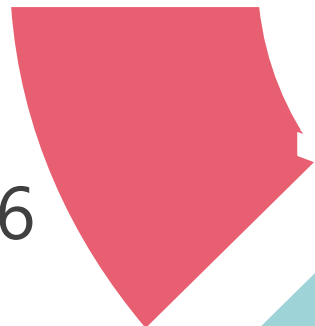
04

Overriding methods



05

Using this() and super()



06

Use of final with
inheritance

Basics questions In this chapter

1. *What is inheritance and it's type?*
2. *What are super class and subclass ?*
3. *What is Protected members in java ?*
4. *What is method Overriding means?*
5. *What does it mean this() and super() ?*
6. *What are Use of final with inheritance ?*



What is inheritance and it's type ?

- ❖ **Inheritance** is the capability of a class (subclass) to use the properties (data fields) and methods of another class (Super Classes) while adding its own functionality.
- ❖ **For example**, a **child** inherits the traits of his/her **parents**. With inheritance, we can reuse the fields and methods of the **existing** class. Hence, inheritance facilitates Reusability and is an important concept of **OOPs**.
- ❖ **Elements** to be **inherited** from **parent** class are **protected** and **public** Members.

- ❖ The **class** that inherits the properties is known as the **sub-class** or the **child class**.
- ❖ The **class** from which the properties are inherited from is known as the **superclass** or the **parent class**.
- ❖ *Inheritance **Syntax** in java*

```
class Child_class extends Parent_class
{
    //methods
    //fields
}
```

Types of inheritance

❖ There are Various types of **inheritance** in Java

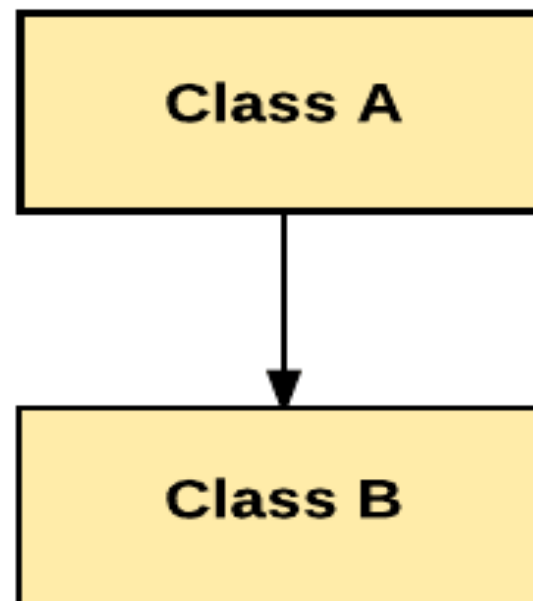
1. **Single** Inheritance
2. **Multiple** Inheritance
3. **Multilevel** Inheritance
4. **Hierarchical** Inheritance
5. **Hybrid** Inheritance

1. Single Inheritance

- ❖ In **Single Inheritance** one class extends another class (one class only).
- ❖ The **diagram** shows that, **Class B** extends only **Class A**. Class **A** is a **super class** and Class **B** is a **Sub-class**

```
class A
{
    //methods and fields
}

class B extends A
{
    //methods and fields
}
```



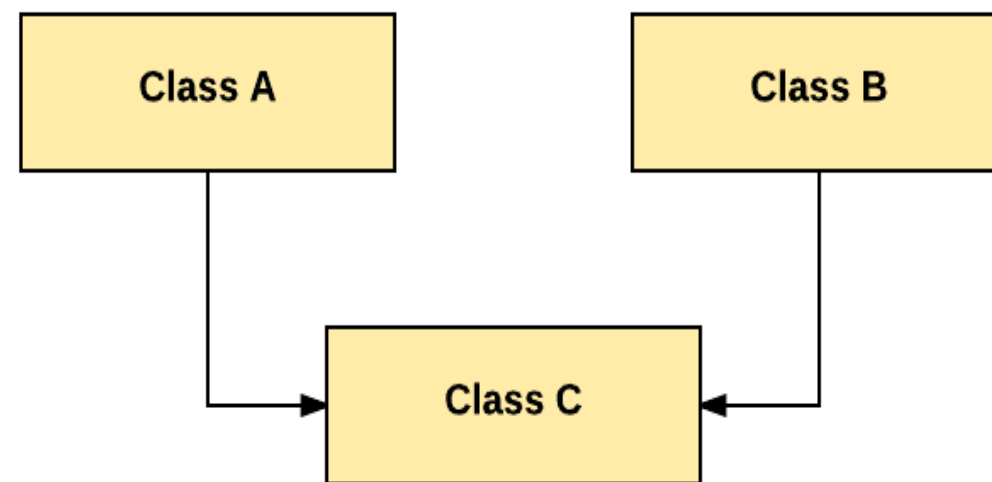
2. Multiple Inheritance

- ❖ In **Multiple Inheritance**, one class extending more than one class. Java does not support multiple inheritance.
- ❖ In this **diagram**, Class C extends Class A and Class B both.

```
class A{
    //methods and fields
}

class B{
    //methods and fields
}

class C extends A,B{
    //methods and fields
}
```



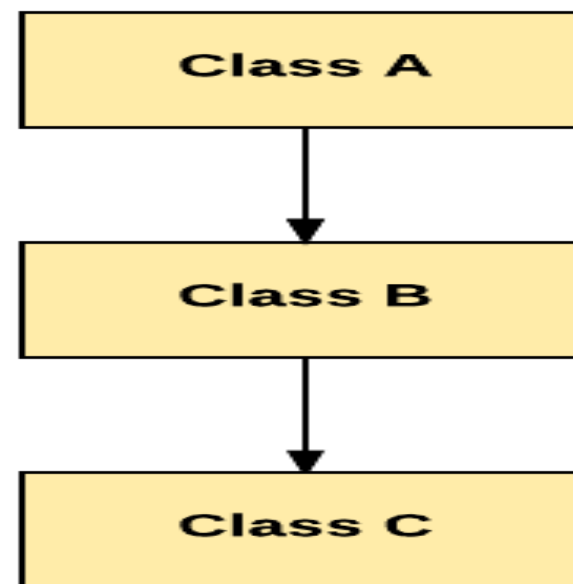
3. Multilevel Inheritance

- ❖ In **Multilevel Inheritance**, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.
- ❖ In this **diagram** Class C is subclass of B and B is a of subclass Class A.

```
class A{
    //methods and fields
}

class B extends A{
    //methods and fields
}

class C extends B{
    //methods and fields
}
```

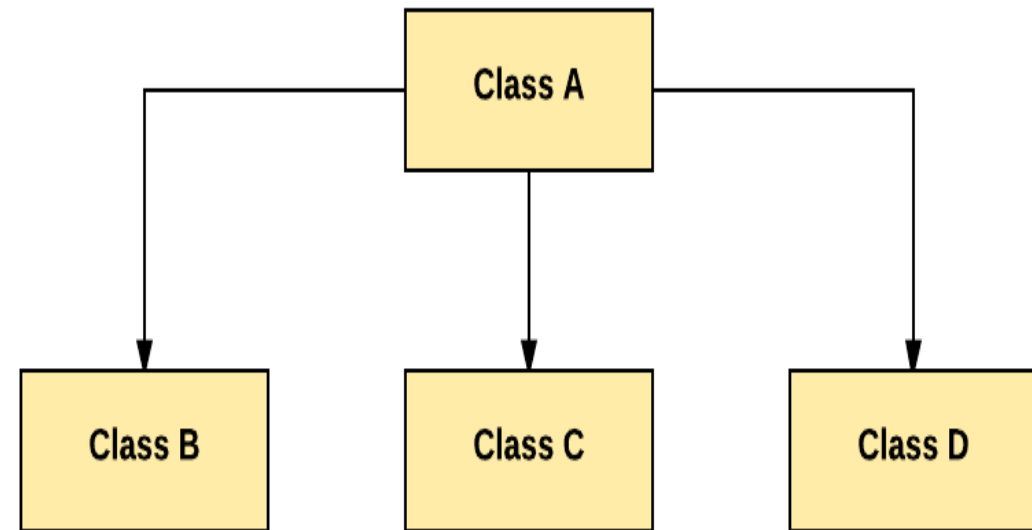


4. Hierarchical Inheritance

ሐምሌ, 2013 ዓ.ም

- ❖ In **Hierarchical Inheritance**, one class is inherited by many sub classes.
- ❖ In this **diagram** Class B, C, and D inherit the same class A.

```
class A{  
    //methods and fields  
}  
  
class B extends A{  
    //methods and fields  
}  
  
class C extends A{  
    //methods and fields  
}  
  
class D extends A{  
    //methods and fields  
}
```

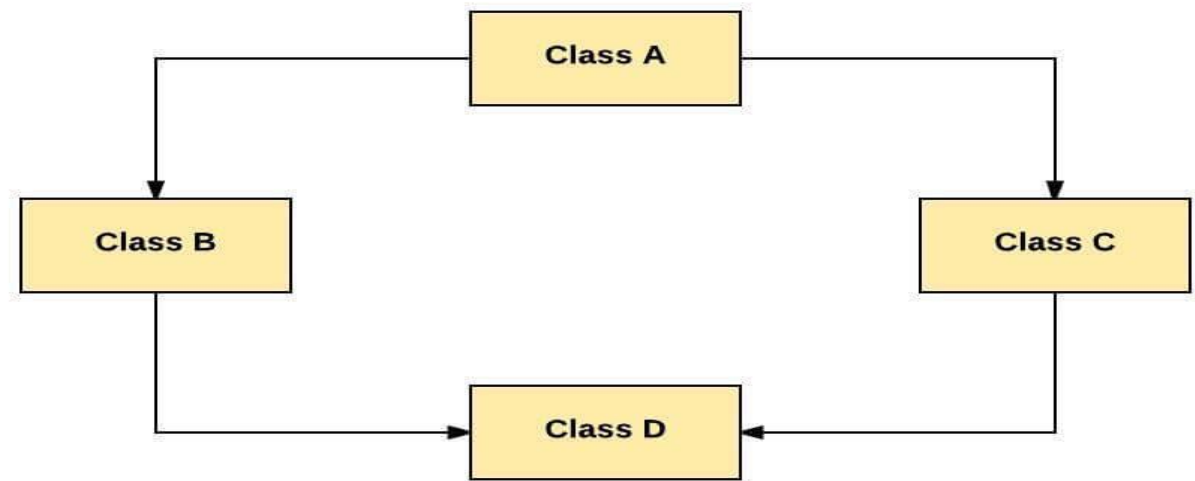


5. Hybrid Inheritance

ሐምሌ, 2013 ዓ.ም

- ❖ **Hybrid inheritance** is a combination of Single and Multiple inheritance.
- ❖ In this **diagram**, all the public and protected members of Class A are inherited into Class D, first via Class B and secondly via Class C.

```
class A{  
    //methods and fields  
}  
  
class B extends A{  
    //methods and fields  
}  
  
class C extends A{  
    //methods and fields  
}  
  
class D extends A{  
    //methods and fields  
}
```



Super classes and subclasses

ሐምሌ, 2013 ዓ.ም

- ❖ A **subclass** is a class that derives from another class. A **subclass** inherits state and behavior from all of its ancestors.
- ❖ The term **superclass** refers to a class's direct ancestor as well as all of its ascendant classes.
- ❖ In Java, it is possible to **inherit** attributes and methods from one class to another. We group the "**inheritance concept**" into two categories:
 - ✓ **subclass** (**child**) - the class that inherits from another class
 - ✓ **superclass** (**parent**) - the class being inherited
- ❖ To **inherit** from a class, use the **extends** keyword.

Example

ሐምሌ, 2013 ዓ.ም

```
class Vehicle { //super class
    protected String brand = "Ford"; // Vehicle
attribute
    public void honk() { // Vehicle method
        System.out.println("Tuut, tuut!");
    }
}
class Car extends Vehicle { //sub class
    private String modelName = "Mustang"; // Car
attribute
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.honk();
    }
}
```

- ❖ **Variables, methods, and constructors**, which are declared **protected** in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
- ❖ The **protected access modifier** cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Cont....

ሐምሌ, 2013 ዓ.ም

```
Protected class Rectangle { // class can't declared as protected
    private int x, y, w, h;
    public String data() {
        return "welcome";
    }
}

class DecoratedRectangle extends Rectangle {
    private int borderWidth;
}
```

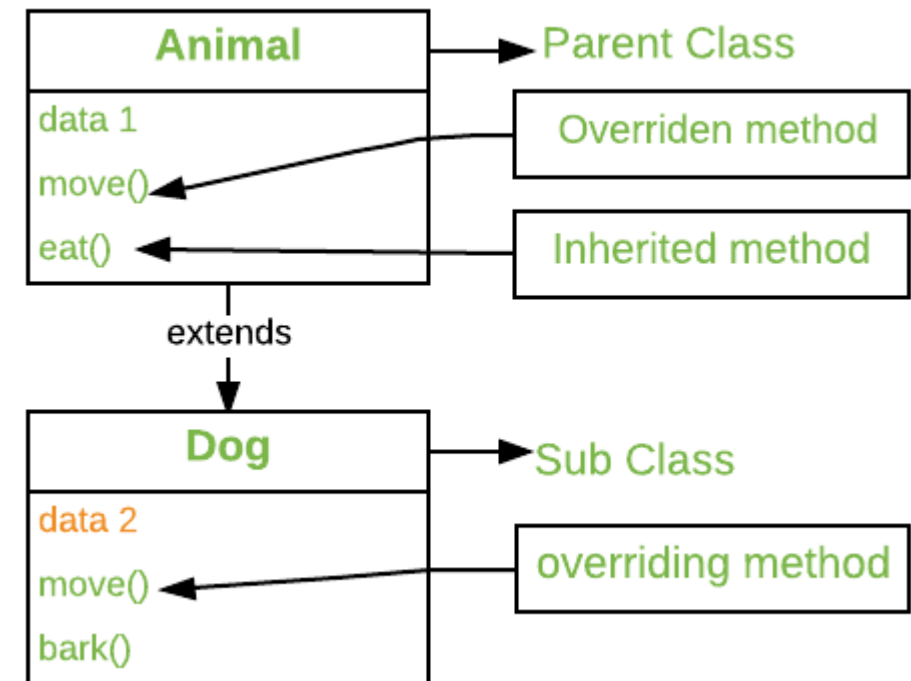
- ❖ In any **object-oriented programming language**, **Overriding** is a feature that allows a **subclass** or **child** class to provide a specific implementation of a method that is already provided by one of its **super-classes** or **parent classes**.
- ❖ When a **method** in a **subclass** has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.

- ❖ **Method overriding** is one of the way by which java achieve Run Time Polymorphism.
- ❖ The version of a method that is executed will be determined by the **object** that is used to invoke it. If an object of a **parent** class is used to invoke the method, then the version in the **parent** class will be executed, but if an object of the **subclass** is used to invoke the method, then the version in the **child** class will be executed.

- ❖ The ability of a subclass to **override** a **method** allows a class to inherit from a superclass whose behavior is "close enough" and then to modify behavior as needed.

❖ NB

- ✓ **Final** methods can not be overridden
- ✓ **Static** methods can not be overridden
- ✓ **private** methods are not overridden



Cont....

ሐምሌ, 2013 ዓ.ም

```
class Parent {
    private void m1() {
        System.out.println("From parent m1()");
    }
    protected void m2() {
        System.out.println("From parent m2()");
    }
}

class Child extends Parent {
    private void m1{()} // unique to Child class
    System.out.println("From child m1()");
}
@Override // overriding method, with more accessibility
public void m2() {
    System.out.println("From child m2()");
}
}
```

```
class Main {  
    public static void main(String[] args) {  
        Parent obj1 = new Parent();  
        obj1.m2();  
        Parent obj2 = new Child();  
        obj2.m2();  
    }  
}
```

Using this() and super()

ሐምሌ, 2013 ዓ.ም

- ❖ The **super** keyword refers to the objects of immediate **parent** class. Before learning super keyword you must have the knowledge of [inheritance in Java](#) so that you can understand simply how to use and what is the importance of **super** in java

❖ The **use** of super keyword

- ✓ To **access** the data members of **parent** class when both **parent** and **child** class have member with same name
- ✓ To **explicitly** call the no-arg and parameterized **constructor** of **parent** class
- ✓ To **access** the method of parent class when **child** class has **overridden** that method

Cont....

ሐምሌ, 2013 ዓ.ም

```
class Superclass {  
    int num = 100;  
}
```

```
class Subclass extends Superclass{
```

```
    int num = 110;
```

```
    void printNumber() {
```

```
        System.out.println(num);
```

```
    }
```

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

```
    Subclass obj= new Subclass();
```

```
    obj.printNumber();
```

```
}
```

```
}
```



There is no way you can access the num variable of parent class without using super keyword.

Cont....

ሐምሌ, 2013 ዓ.ም

```
class Superclass {  
    int num = 100;  
}
```

```
class Subclass extends Superclass{
```

```
    int num = 110;
```

```
    void printNumber() {
```

Now you can access **num** of parent class

```
        System.out.println(super.num);
```

```
    }
```

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

```
    Subclass obj= new Subclass();
```

```
    obj.printNumber();
```

```
}
```

```
}
```


❖ The **this keyword** refers to the **current** object in a **method** or **constructor**.

The most common use of the **this keyword** is to eliminate the confusion between class attributes and parameters with the **same name** (because a class attribute is shadowed by a **method** or **constructor** parameter).

this can also be used to:

- ✓ Invoke **current class constructor**
- ✓ Invoke **current class method**
- ✓ Return the **current class object**
- ✓ Pass an argument in the method call
- ✓ Pass an argument in the constructor call

Cont....

ሐምሌ, 2013 ዓ.ም

```
public class Main {  
    int x;  
    // Constructor with a parameter  
    public Main(int x) {  
        this.x = x;  
    }  
    // Call the constructor  
    public static void main(String[] args) {  
        Main myObj = new Main(5);  
        System.out.println("Value of x = " + myObj.x);  
    }  
}
```

- ❖ final is a keyword in java used for **restricting** some functionalities. We can declare variables, methods and classes with final keyword.
- ❖ When a **class** is declared as **final** then it cannot be **subclassed** i.e. no any other class can extend it.
- ❖ When a **method** is declared as **final** then it cannot be overridden by subclasses.

```
final class A {  
    // methods and fields  
}  
// The following class is illegal.  
class B extends A {  
    // ERROR! Can't subclass A  
}
```

ሐምሌ, 2013 ዓ.ም

THANK YOU

Any questions?
Feel free !