Marwadi University

Unit – 4
Inheritance, Interface and Packages

Prepared By
Prof. Ravikumar Natarajan
Assistant Professor, CE Dept.

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

Contents

Marwadi University

- Inheritance basics
- Super keyword
- Multilevel hierarchy
- Overriding methods
- Dynamic method dispatch
- Abstract class
- Using final with inheritance
- Object class
- Interfaces
- Packages: defining and importing
- Access protection

Inheritance basics

Marwadi University

- It is defined as the process where derived class can borrow the properties of base class.
- Inheritance can be achieved by using the "extends" keyword.

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

Advantages:

i. Code reusability
ii. Extensibility
iii. Data hiding
iv. Overriding

Inheritance basics

Marwadi University

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

Example:

```
class A            //Base class //Super class // Parent class
{
}
class B extends A        //Derived class //Sub class //Child class
{
..//Use of class A properties
}
```

Inheritance Types

1. **Single inheritance -** It has one parent per derived class. | Whenever a class inherits another class, it is called single inheritance.

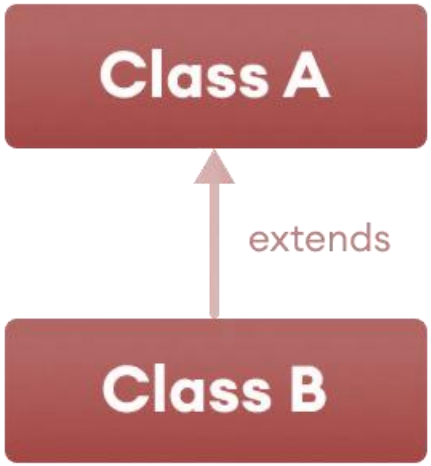
2. Multilevel inheritance – when a derived class is derived from base class which itself is derived again is called multilevel inheritance. | Multilevel inheritance is a part of single inheritance where more than two classes are in the sequence.

3. Hierarchical inheritance - If a class has more than one derived classes This is known as hierarchal inheritance.

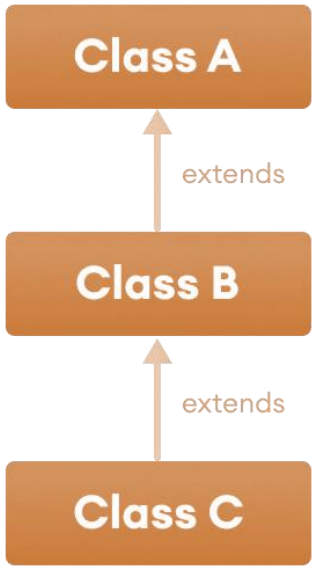
4. Multiple inheritance - If a class inherits the data member and member function from more than one base class, then this type of inheritance is called multiple inheritance. (Java will not support directly but it is possible using "interface")

5. Hybrid inheritance – when two or more types of inheritances (Single and multiple inheritance) are combined together then it is called hybrid inheritance.

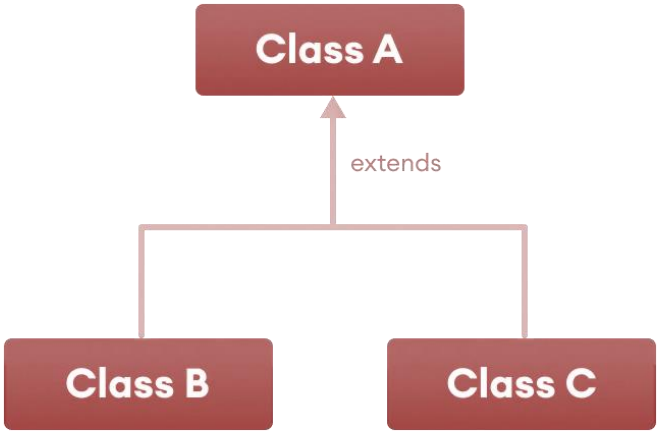
Inheritance Types



Single

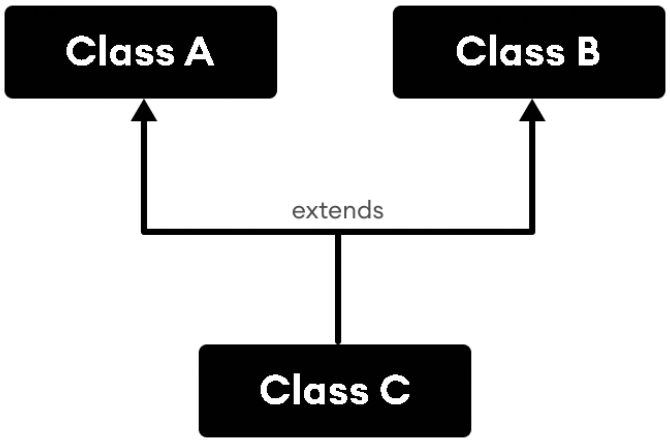


Multilevel

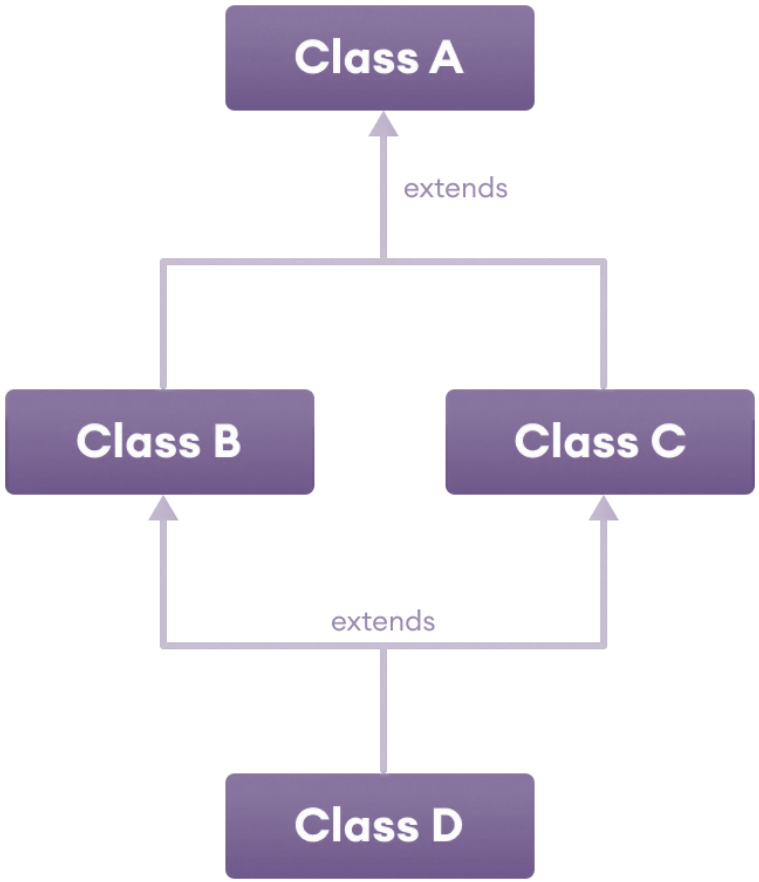


Hierarchical

Inheritance Types



Multiple



Hybrid

Single Inheritance

```
//Single inheritance – Example
class a             //base class
{       void disp()     //method 1
        {
                System.out.println("Hello");   }}
class b extends a       //derived class
{       void disp1()    //method 2
        {
        System.out.println(" Good morning");   }}
class singleinh //main class
{       public static void main(String args[])          //main function
        {
        b B1 = new b(); // reference variable B1, Creating object to allocate memory space
        B1.disp1();     //calling method
        B1.disp();      }}
```

Multilevel Inheritance

Marwadi University

```
//Multilevel inheritance – Example
class a
{       void disp()     {  System.out.println("Helllo");        }}
class b extends a
{       void disp1()    {  System.out.println("Good morning");      } }
class c extends b
{       void disp2()    {  System.out.println("Welcome to MU");  } }

class multilevelinh{
        public static void main(String args[])          {
        c C1 = new c();
        C1.disp2();
        C1.disp1();
        C1.disp();
        }}
```

Hierarchical Inheritance


```
//Hierarchical inheritance – Example
class a
{       void disp()     {  System.out.println("Helllo");        }}
class b extends a
{       void disp1()    {  System.out.println("Good morning");      } }
class c extends a
{       void disp2()    {  System.out.println("Welcome to MU");  } }

class hierarInh{
        public static void main(String args[])          {
        c C1 = new c();
        C1.disp2();
        C1.disp1();// Question?
        C1.disp();
        }}
```

Multiple Inheritance

Marwadi University

Why Interface should be used in multiple inheritance? Consider the example

```
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 occurs

Interface

Marwadi University

- Since abstract class allows concrete methods as well, it does not provide 100% abstraction.
- You can say that it provides partial abstraction.
- Interfaces are used for 100% abstraction (full abstraction)

Syntax:

```
modifier interface InterfaceName {

/** Constant declarations */
/** Abstract method signatures */
}
```

Multiple Inheritance

Marwadi University

```
//Multiple inheritance example
interface mobileFeature1 {
        int  pixel=720;
        public void camera();
}
interface mobileFeature2 {
        int color=200;
        public void design();
}
class Mobile implements mobileFeature1, mobileFeature2 //two interfaces
{       public void camera() {
                int  color=pixel*200;
                System.out.println("Suported Colors: "+color);
        }
        public void design() {
                int pixel=color*300;
                System.out.println("Supported Pixel "+pixel);
        }
}
```

```
class MultipleInterface{
        public static void main(String args[]){
                Mobile obj = new Mobile();
                obj.camera();
                obj.design();
} }
```

Hybrid Inheritance

```
class A                 //class
{           int a=10; //variable with value
assigned
}

interface B         //Interface
{int b=20;
}
//Multiple inh.
class C extends A implements B      {
int c;
int mul()
            //method1
{           c=a*b;
            return c;
}
}
```

```
class D extends C                   //Single inh.
{void sum()                                 //method2
            { System.out.println("Adding all 3
variables");
            int d=a+b+mul();
            //Calc
            System.out.println(d);
            }}
class hybridinh       //main class
{
public static void main(String[] args)     //main fun.
            { C obj1 = new C();
            //Obj. creation
            D obj2 = new D();
            System.out.println("Multiplying two
variables");
            System.out.println(obj1.mul());
            //Print stmt
            obj2.sum();
            //Calling method
            }}
```

Using Super Keyword

Marwadi University

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Conditions where super key word used:

1. Only used in subclass constructor and methods.
2. Call to super must first statement in subclass constructor.
3. Parameters must be in same order.

Super keyword can be used for the following:

1. Use of super with **variables**
2. Use of super with **methods**
3. Use of super with **constructors**

Using Super Keyword

Marwadi University

```
//Use of super with variables:
class Vehicle
{    int maxSpeed = 120; }

class Car extends Vehicle
{    int maxSpeed = 180;
     void display()     {
        System.out.println("Maximum Speed: " + super.maxSpeed);
    }}

class Test
{    public static void main(String[] args) {
        Car small = new Car();
        small.display();
    } }
```

Using Super Keyword

Marwadi University

```
//Use of super with methods
class Person
{   void message()   {
      System.out.println("This is person class");
   } }
 class Student extends Person
{   void message()
   {      System.out.println("This is student class");  }
    void display()
   {       super.message();
       message();      }}
 class Test
{   public static void main(String args[])   {
      Student s = new Student();
       s.display();       }}
```

Using Super Keyword

Marwadi University

```
//Use of super with constructors
class Person {
   Person()   {
      System.out.println("Person class Constructor"); }
   Person(int i)    {  System.out.println("Person class Const. with param:”+i);
   }}
 class Student extends Person {
   Student()    {     System.out.println("Student class Constructor");   }
   Student(int i)    {
       super(i);
       System.out.println("Student class Const. with param");
   }}
 class Main {
   public static void main(String[] args)    {
      Student s = new Student(5);      } }
```

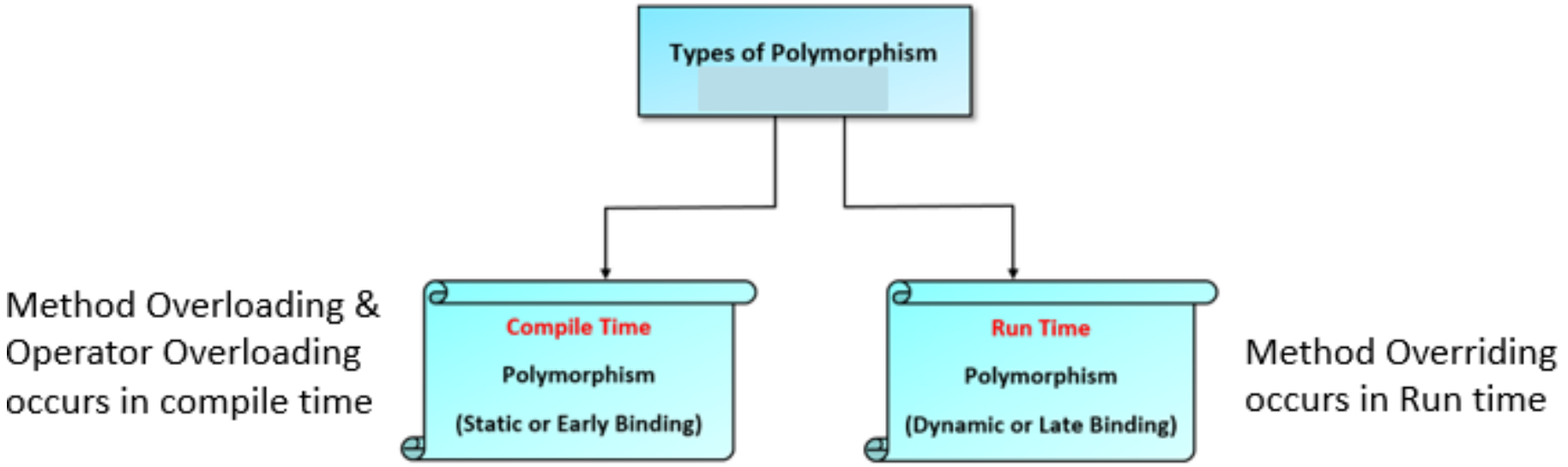
Overriding Methods

Marwadi University

- Discussed in unit 3

Dynamic Method Dispatch

- Discussed in unit 3



Abstraction

Marwadi University

- Class abstraction is the separation of class implementation from the use of a class.
- The details of implementation are encapsulated and hidden from the user, which is called as class encapsulation.

- **Ways to achieve Abstraction**
- There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Abstract Class

Marwadi University

Hiding the internal implementation of the feature and only showing the functionality to the users. i.e. what it works (showing), how it works (hiding). Both abstract class and interface are used for abstraction.

Rules:

1. Abstract method must present in abstract class only.
2. Cannot be instantiated i.e not allowed to create object.
3. Method must be overridden.
4. It can have constructors & final and static methods

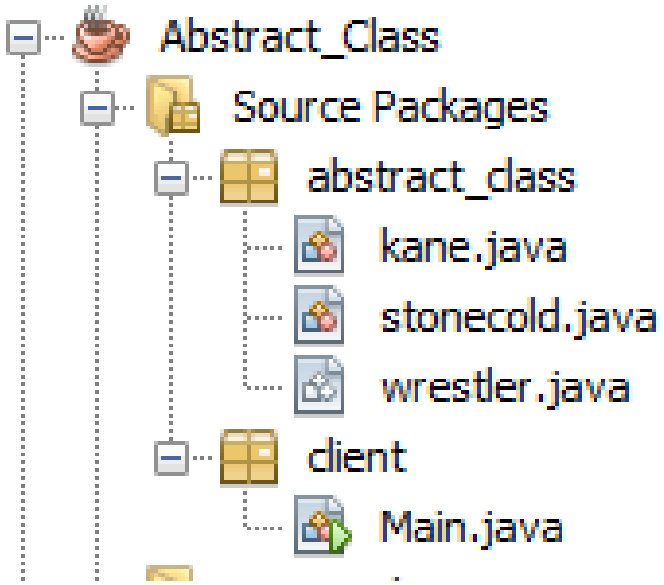
Abstract Class

Marwadi University

Example for abstract class and method:

```
abstract class Animal                    //abstract parent class
{
   public abstract void sound();//abstract method
// public abstract void disp();//if this method is not overridden class error occurs
}
public class Dog extends Animal{
            public void sound()
            {
                        System.out.println(“Bowww..");
            }
            public static void main(String args[])
            {
                        Animal obj = new Dog();
                        obj.sound();
            }}
```

Abstract Class – Case Study



Abstract Class – Case Study

Marwadi University

wrestler.java

```
1  package abstract_class;
2  public abstract class wrestler {
3      public void paymentForWork(int hours)
4      {
5          System.out.println("Amount" + hours*250);
6      }
7      public abstract void themeMusic();
8      public abstract void finisher();
9  }
```

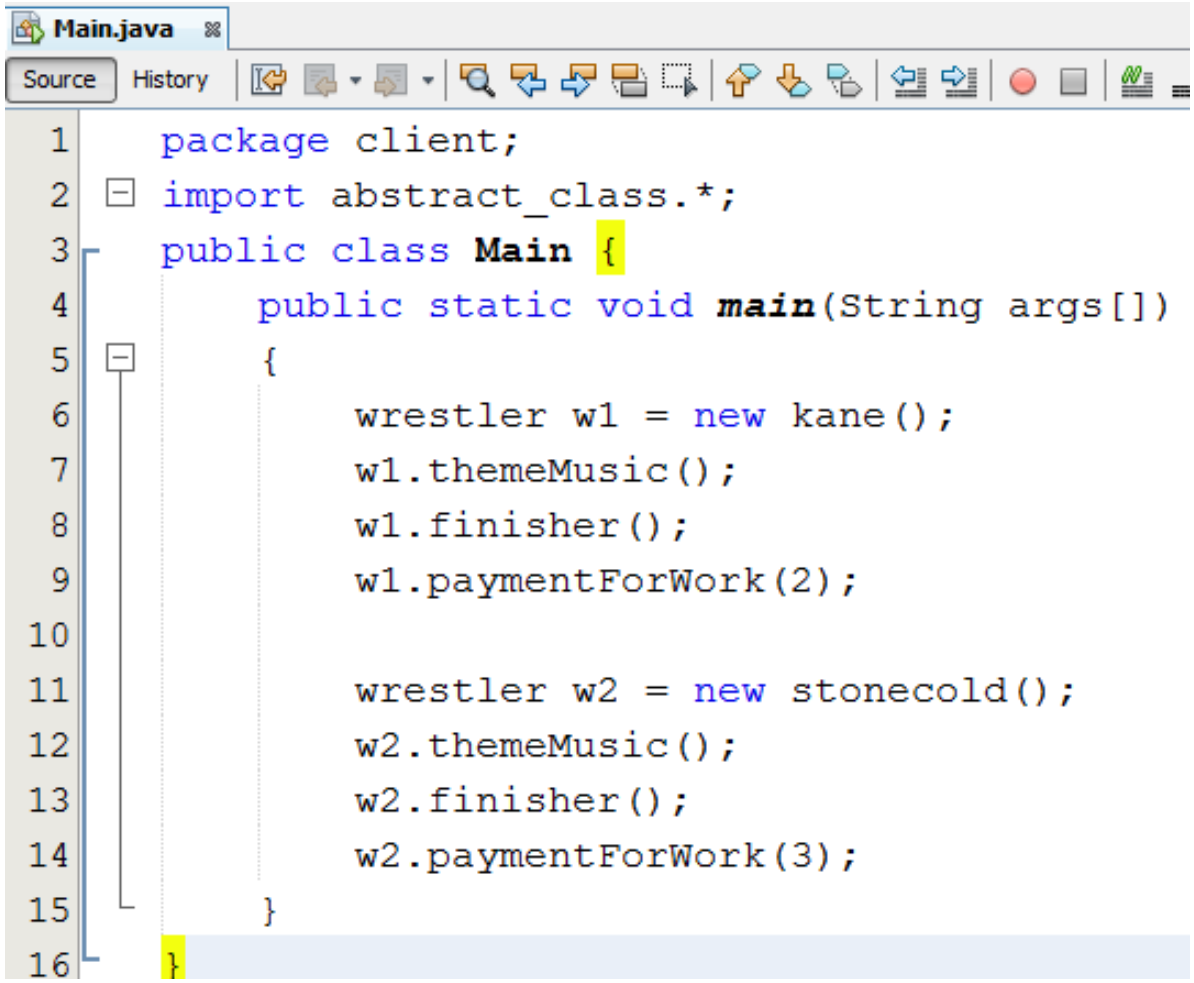
kane.java

```
1  package abstract_class;
2  public class kane extends wrestler {
3      public void themeMusic(){
4        System.out.println("Kane intro")  ;
5      }
6      public void finisher(){
7        System.out.println("Tombstone")  ;
8      }
9  }
```

stonecold.java

```
1  package abstract_class;
2  public class stonecold extends wrestler{
3      public void themeMusic(){
4        System.out.println("Stonecold intro")  ;
5      }
6      public void finisher(){
7        System.out.println("stonecold stunner")  ;
8      }
9  }
```

Abstract Class – Case Study



Interface vs Abstract Class

Marwadi University

Parameters	Interface	Abstract class
Speed	Slow	Fast
Multiple Inheritances	Implement several Interfaces	Only one abstract class
Structure	Abstract methods	Abstract & concrete methods
When to use	Future enhancement	To avoid independence
Inheritance/ Implementation	A Class can implement multiple interfaces	The class can inherit only one Abstract Class
Data fields	the interface cannot contain data fields.	the class can have data fields.
Abstract keyword	“interface” keyword used	“abstract” keyword used
Hiding	100% abstraction	0 to 100% abstraction
Instantiation	Object cannot be created	Object cannot be created
Methods	Methods in interface are abstract by default & compulsory to override.	Both abstract & non abstract methods can be used.

Encapsulation

Marwadi University

- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

- In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

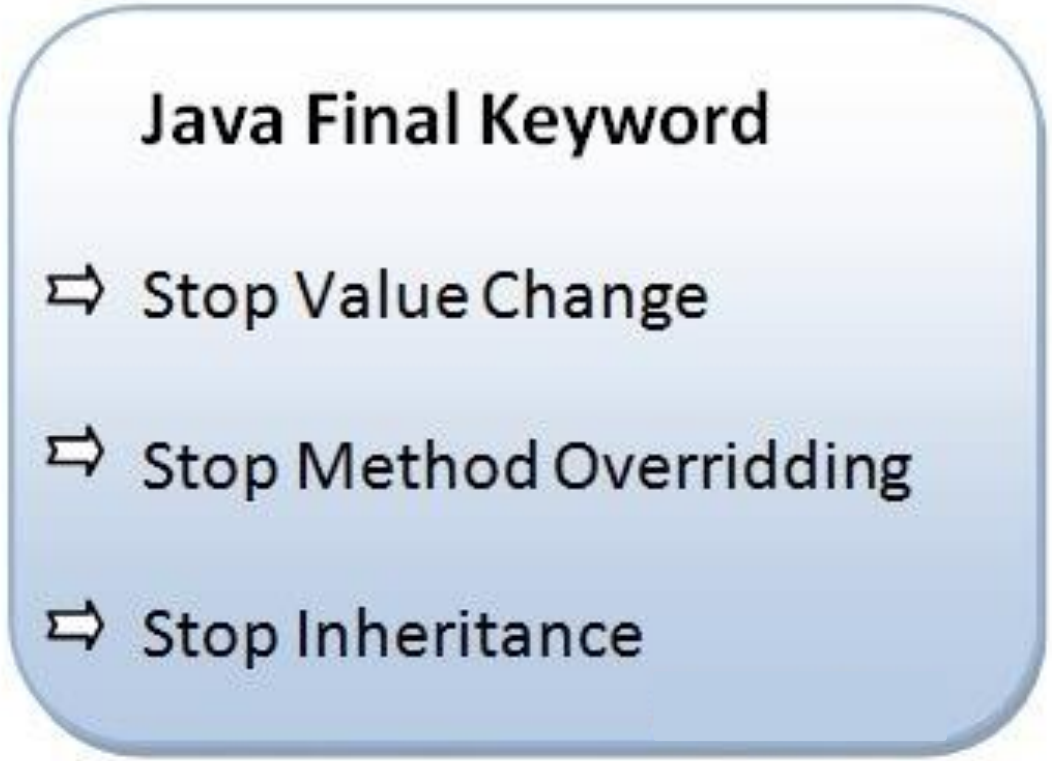
Encapsulation

Marwadi University

```
class Encapsulate {
private String Name;
public String getName()
{ return Name; }
public void setName(String newName)
   {
         Name = newName;
   }
public class Main{
   public static void main(String[] args)  {
      Encapsulate obj = new Encapsulate();
         obj.setName("Harsh");
         System.out.println("name: " + obj.getName());
         //System.out.println("name: " + obj.Name);
}}
```

Using Final with Inheritance

- The final keyword in java is used to restrict the user.
- The java final keyword can be used in many context. Final can be:
- **Variable**
- **Method**
- **Class**



Using Final with Inheritance

Marwadi University

- Java final variable
- If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class A{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Aobj=new  A();
 obj.run();
 }
```

Output: Compile Time Error

Using Final with Inheritance

Marwadi University

- Java final method
- If you make any method as final, you cannot override it.

```
class A{
  final void run(){System.out.println("From Class A");}
}

class B extends A{
   void run(){System.out.println("From Class B");}

   public static void main(String args[]){
   B b1= new B();
   b1.run();
   }
}
```

Output: Compile Time Error

Using Final with Inheritance

Marwadi University

- Java final class
- If you make any class as final, you cannot extend it.

```
final class B{}

class H extends B{
  void run(){System.out.println("From class H");}

  public static void main(String args[]){
  H h1= new H();
  h1.run();
  }
}
```

Output: Compile Time Error

Using Final with Inheritance

Marwadi University

- Is final method inherited?
- **Yes, final method is inherited but you cannot override it.**

```
class B{
  final void run(){System.out.println("B class");}
}
class H extends B{
   public static void main(String args[]){
    H h1 = new H()    ////new H().run();
    h1.run();
     }
}
```

Output: Compile Time Error

Object Class

Marwadi University

- The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- Object class is present in **java.lang** package.
- Object class methods:
- toString()
- hashCode()
- Equals()
- getClass()
- Finalize()
- Clone()

Object Class

Marwadi University

```
class Object { }
public class Test {
    public static void main(String[] args)
    {
        Object obj = new String(“Hi");
        Class c = obj.getClass();
        System.out.println("Class of Object obj is: "+ c.getName());
    }
}
```

Object Class


```
class Main{
 int rollno;
 String name;

 Main(int rollno, String name){
 this.rollno=rollno;
 this.name=name;
 }

 public String toString(){//overriding the toString() method
  return rollno+" "+name;
 }
 public static void main(String args[]){
   Main s1=new Main(101,"Raj");
   Main s2=new Main(102,"Vijay");

   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()  }  }
```

Packages: Defining and Importing

Marwadi University

- Packages is a mechanism in which variety of classes and interfaces can be grouped together.

- Advantages:

i. Code reused – from other package
ii. Same name – two classes from two different packages
iii. Possible to hide the classes
iv. Name of the directory becomes the package name.

- Two types:

i. **Built-in packages**
- Ex: java.lang, java.util, java.io, java.awt, java.applet

ii. **User defined packages**

Packages: Defining and Importing

Marwadi University

- Syntax to create package:
- **package** package_name;
- **package** abc;

- Syntax to Import package:
- **import** abc.*;

Packages: Defining and Importing

Marwadi University

STEP 1: Create a folder p1 and follow the below code,

```
package p1;
public class testpackage
{
        public void display()
        {
        System.out.println("Hi");
        }
}
```

Save the file in p1 folder as “testpackage” and compile it.

Packages: Defining and Importing

Marwadi University

STEP 2: Come out from the folder p1 and save the below code as “test”.

```
import p2.*;
public class test
{
        public static void main(String args[])
        {
                testpackage tp = new testpackage();
                tp.display();
        }
}
```

Now compile and run the “test” file

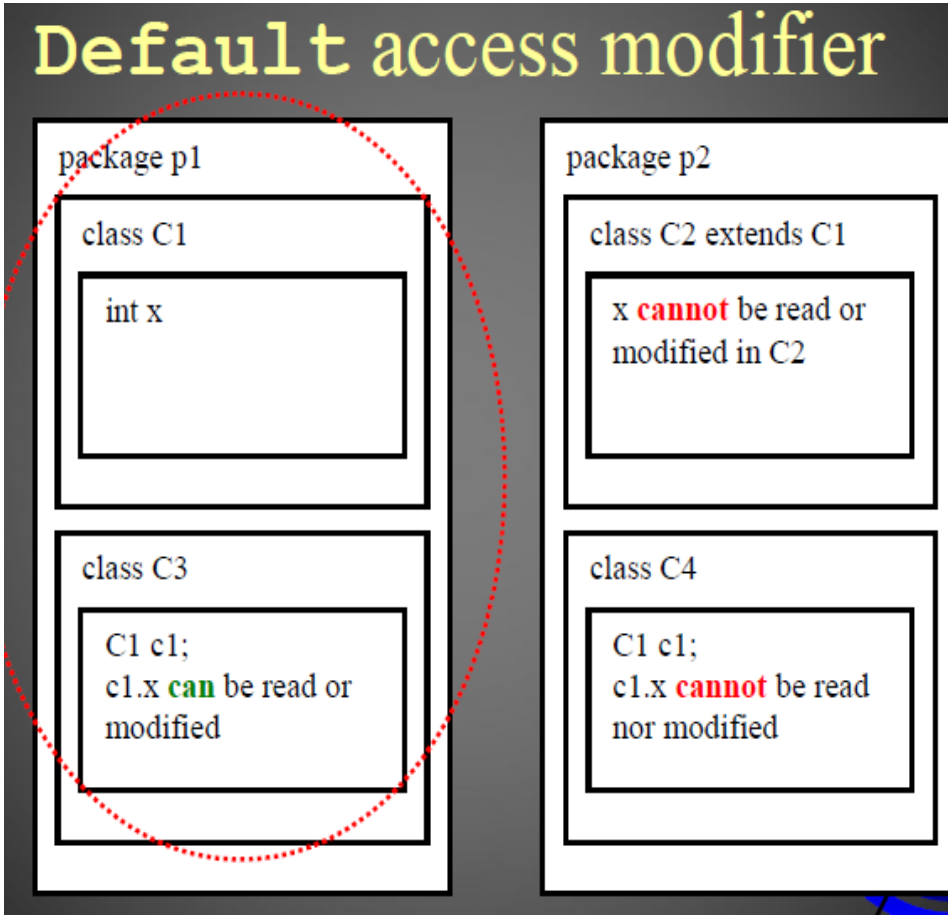
Access Protection

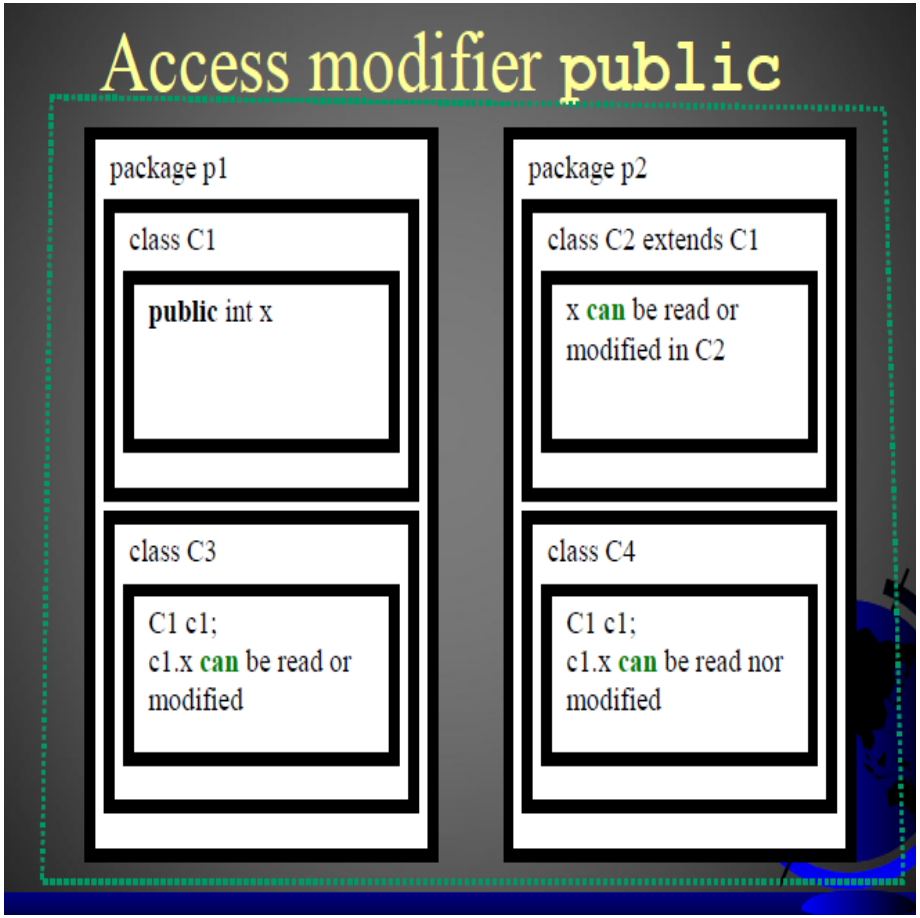
Marwadi University

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.
- Four types:
- Public
- Protected
- Default
- Private

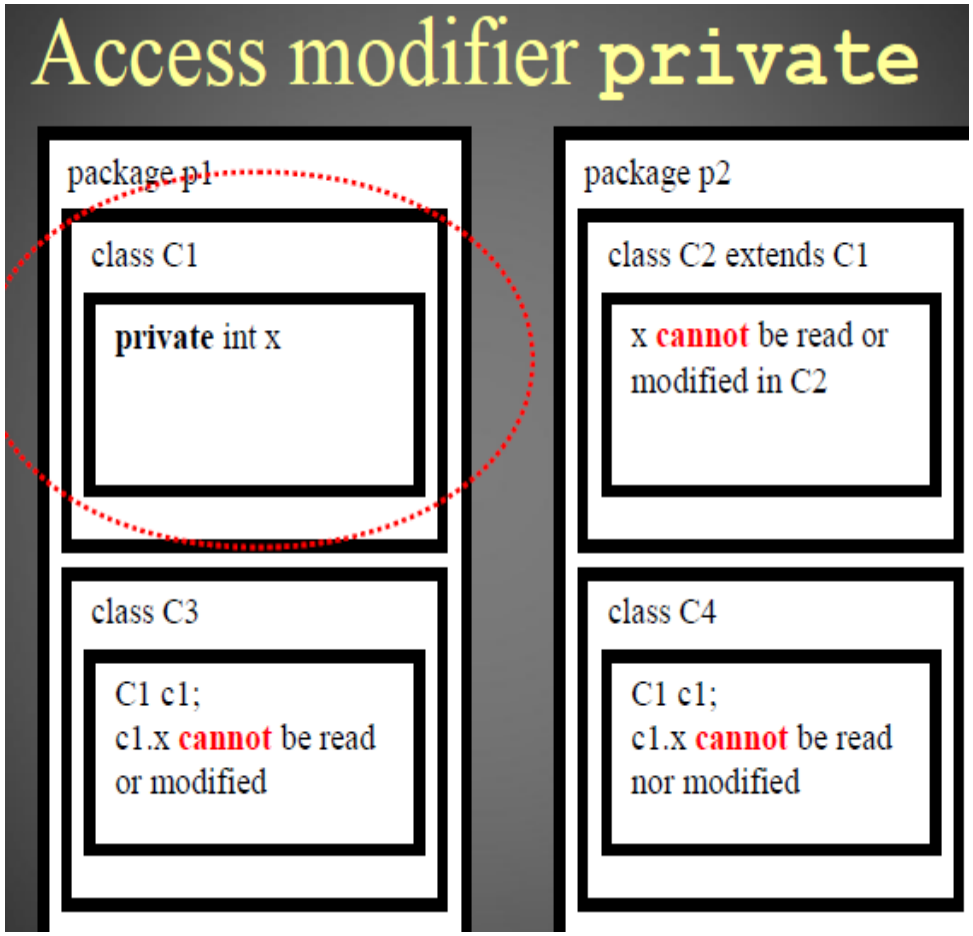
Access Location	Access Modifier			
	Public	Protected	Default	Private
Same class	**Yes**	**Yes**	**Yes**	**Yes**
Sub class in same package	**Yes**	**Yes**	**Yes**	No
Other classes in same package	**Yes**	**Yes**	**Yes**	No
Subclass in other packages	**Yes**	**Yes**	No	No
Non-subclass in other package	**Yes**	No	No	No

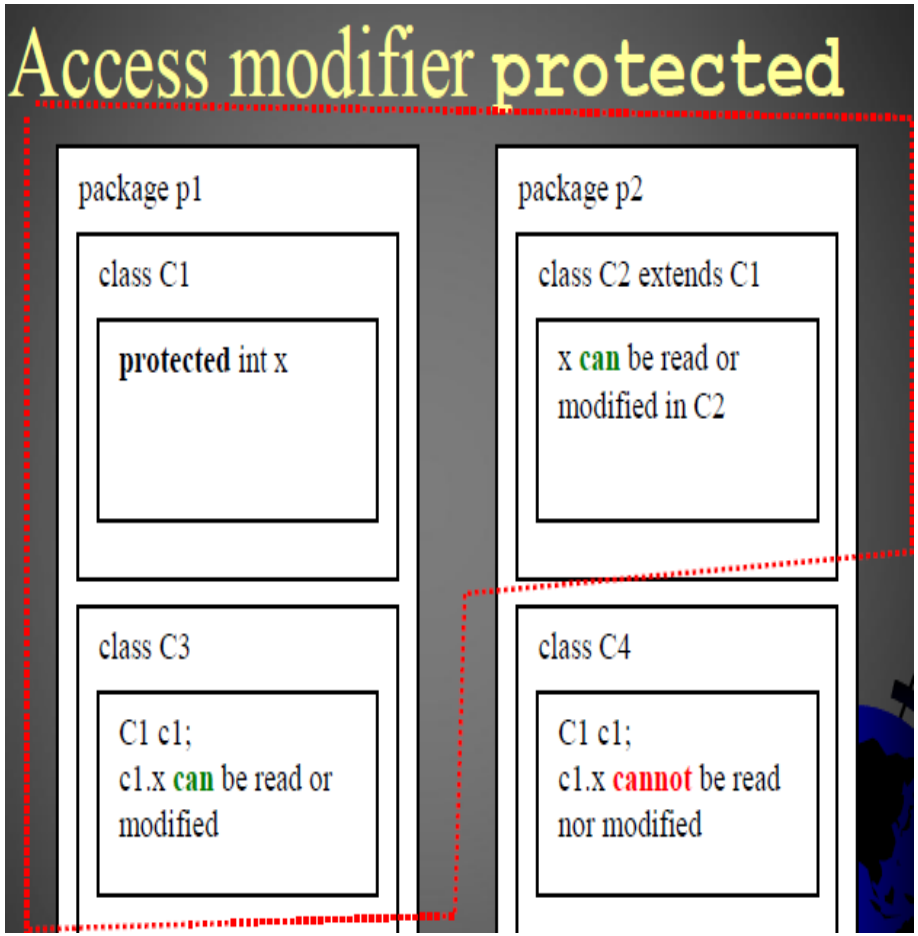
Access Protection





Access Protection





Summary

Marwadi University

- Inheritance basics
- Super keyword
- Multilevel hierarchy
- Overriding methods
- Dynamic method dispatch
- Abstract class
- Using final with inheritance
- Object class
- Interfaces
- Packages: defining and importing
- Access protection

Next

Marwadi University

- Exception handling overview
- Types of exception
- Using try
- Catch and finally clauses
- Multiple catch clauses
- Throw and throws keyword
- Custom exception class

END OF UNIT - 4