K J's EDUCATIONAL INSTITUTES
TRINITY COLLEGE OF ENGINEERING & RESEARCH
Approved by AICTE, Government of Maharashtra & Affiliated to Savitribai Phule Pune University

Accredited NAAC Grade 'A+'

Department of Computer Engineering

Course Name :Principles of Programming Languages

Course Code: 210255

Unit : 4 - Inheritance, Packages and Exception Handling using Java

Course Coordinator: Mrs. Shaikh J. N.

A.Y : 2024-25

Semester: II

Course Objectives:

To learn Object Oriented Programming (OOP) principles using Java Programming Language.

Course Outcomes:

On completion of the course, learner will be able to
CO3: Develop application using inheritance, encapsulation, and polymorphism.

Contents :

Inheritances: member access and inheritance, super class references, Using super, multilevel hierarchy, constructor call sequence, method overriding, dynamic method dispatch, abstract classes, Object class.

Packages and Interfaces: defining a package, finding packages and CLASSPATH, access protection, importing packages
Interfaces (defining, implementation, nesting, applying), variables in interfaces, extending interfaces, instance of operator.
fundamental, **exception** types, uncaught exceptions, try, catch, throw, throws, finally, multiple catch clauses, nested try statements, built-in exceptions, custom exceptions (creating your own exception sub classes).
Managing I/O: Streams, Byte Streams and Character Streams, Predefined Streams, Reading console Input, Writing Console Output, Print Writer class.

SPPU Questions

Inheritance

- Inheritance is the process of **acquiring the properties by the sub class (or derived class or child class) from the super class (or base class or parent class).**
- When a **child** class(newly defined abstraction) inherits(extends) its **parent** class (being inherited abstraction), all the properties and methods of parent class becomes the member of child class.
- In addition, child class can add new data fields(properties) and beh
- It can override methods that are inherited from its parent class.
- The keyword **extends** is used to define inheritance in Java.
- **Syntax:-**

class subclass-name extends superclass-name

{

```
//base class:
class A
{  //members of A
}
//Derived class syntax:
class  B extends A
{
}
```

```
// Create a superclass.
class A {
    int i, j;
    void showij() {
    System.out.println("i and j: " + i + " " + j);
    }
}
// Create a subclass by extending class A.
class B extends A {
    int k;
    void showk() {
            System.out.println("k: " + k);
    }
    void sum() {
            System.out.println("i+j+k: " + (i+j+k));
    }
}
class SimpleInheritance {
    public static void main(String args[]) {
            A superOb = new A();
            B subOb = new B();
            // The superclass may be used by itself.
            superOb.i = 10;
            superOb.j = 20;
            System.out.println("Contents of superOb:");
            superOb.showij();
/* The subclass has access to all public members
of its superclass. */
                    subOb.i = 7;
                    subOb.j = 8;
                    subOb.k = 9;

System.out.println("Contents of  subOb: ");

                    subOb.showij();
                    subOb.showk();
                    System.out.println();

System.out.println("Sum of i, j and k in subOb:");

                    subOb.sum();
                    }
}
```

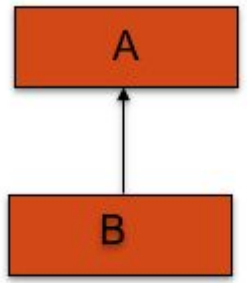
```
Contents of superOb:
i and j: 10 20

Contents of subOb:
i and j: 7 8
k: 9

Sum of i, j and k in subOb:
i+j+k: 24
```

The Benefits of Inheritance

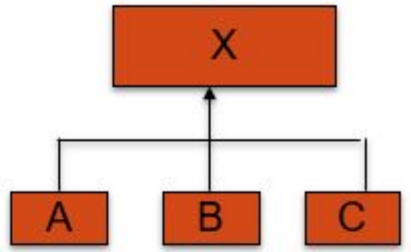
- **Software Reusability** (among projects)
  - Code (class/package) can be reused among the projects.
  - Ex., code to insert a new element into a table can be written once and reused.
- **Code Sharing** (within a project)
  - It occurs when two or more classes inherit from a single parent class.
  - This code needs to be written only once and will contribute only once to the size of the resulting program.
- **Increased Reliability** (resulting from reuse and sharing of code)
  - When the same components are used in two or more applications, the bugs can be discovered more quickly.
- **Information Hiding**
  - The programmer who reuses a software component needs only to understand the nature of the component and its interface.
  - It is not necessary for the programmer to have detailed information such as the techniques used to implement the component.

Types of Inheritance

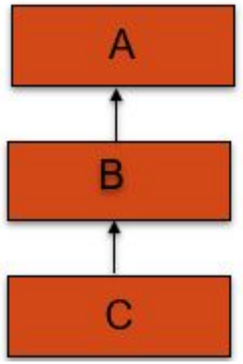
Single Inheritance



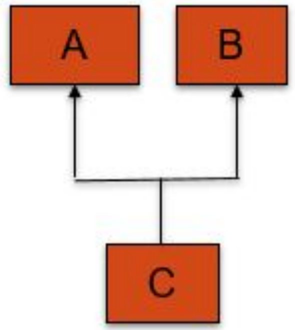
Hierarchical Inheritance



Multilevel Inheritance



Multiple Inheritance / Not Actually exists in Java



```
//Single Inheritance

class A{
}
class B extends A{
}
```

```
//Multilevel Inheritance
class A{
}
class B extends A{
}
class C extends B{
}
```

```
//Hierarchical Inheritance
class A{
}
class B extends A{
}
class C extends A{
}
```

```
//Multiple Inheritance

interface one{
}
interface two{
}
class A implements one, two{
}
```

Multiple Inheritance can be implemented by implementing multiple interfaces not by extending multiple classes.

Example :

```
class B extends A implements C , D{

}
```

OK

```
class C extends A extends B{

}
```

WRONG

```
class C extends A ,B{   }
```

WRONG

A Superclass Variable Can Reference a Subclass Object

When a reference to a subclass object is assigned to a superclass variable, **you will have access only to those parts of the object defined by the superclass.**

```
class Base
{  public void display()
    {
      System.out.println("Base class display is called");
     }
}
class Derv1 extends Base
  { int i;
       public void display()
        {
         System.out.println("Derv1 class display is called");
        } }
class Derv2 extends Base
  {
     public void display()
     {
      System.out.println("Derv2 class display is called");
     }}
```

```
public class polymorhism {
      public static void main(String[] args)
   {
     Base ptr; //Base class reference variable
     Derv1 dl = new Derv1();
     Derv2 d2 = new Derv2();
     ptr = dl; // Base class variable is assigned reference
of super class
     ptr.i=10;  // Error
 System.out.println(ptr.i);// ptr contain reference of Derv1
object .>Error
     ptr.display();
     ptr = d2; // ptr contain reference of derv2 object
     ptr.display();
   }}
```

Output:

```
Derv1 class display is called
Derv2 class display is called
```

Super Keyword

- Subclass refers to its immediate superclass by using **super** keyword.
- **super** has two general forms.
  - First it calls the superclass constructor.
  - Second is used to access a member of the superclass that has been hidden by a member of a subclass.
- **Using super to call superclass constructors**
  - super (parameter-list);
  - parameter-list specifies any parameters neede
  - **super( )** must always be the first statement ex

```
class Box
{
 int i;
Box()
{
 i =10;
 System.out.println("Box() in super class "+i);
}
Box(int a)
{
 System.out.println("Box(int a) in super class");
}}
```

```
class BoxWeight extends Box
{
int i;
BoxWeight()
{
 i = 20;
 super. i = 30;
 System.out.println("BoxWeight() in sub class " +i);
}}
class DemoBoxWeight
{
public static void main(String args[])
{
BoxWeight mybox1 = new BoxWeight();
}}
```

Output:

```
Box() in super class
BoxWeight() in sub class
```

```
//Using super to call superclass //constructors
class Box
{
Box()
{
        System.out.println("Box() in super class");
}
Box(int a)
{
         System.out.println("Box(int a) in super class");
}
}
class BoxWeight extends Box
{
BoxWeight(){
 super(10);
  System.out.println("BoxWeight() in sub class");
}
}
class DemoBoxWeight
{
public static void main(String args[])
{
BoxWeight mybox1 = new BoxWeight();
}
}
```

Output:

```
Box(int a) in super class
BoxWeight() in sub class
```

- The second form of **super** acts somewhat like **this**, except that it always refers to the superclass of the subclass in which it is used.
- Syntax: super.member
- Here, member can be either a method or an instance variable.
- This second form of **super** is most applica hide members by the same name in the superclass.

```
// Using super to overcome name hiding.
class A
{
      int i;
}
// Create a subclass by extending class A.
class B extends A
{
      int i; // this i hides the i in A
      B(int a, int b)
{
      super.i = a; // i in A
      i = b; // i in B
}
```

```
void show()
{
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}
class UseSuper
{
  public static void main(String args[])
{
    B subOb = new B(1, 2);
    subOb.show();
}}
```

OUTPUT

```
i in superclass: 1
i in subclass: 2
```

Method overriding

- When a method in a subclass has the same name, signature and return type as a method in its superclass, then the method in the subclass is said to be overridden the method in the superclass.
- By method overriding, subclass can implem

```
//Overriding example
class A{
      int i,j;
      A(int a,int b){
         i=a;
         i=b;
      }
      void show(){
        System.out.println("i and j :"+i+" "+j);
      }
}
```

```
class B extends A{
       int k;
      B(int a, int b, int c){
         super(a,b);
         k=c;
      }
void show(){
         System.out.println("k=:"+k);
      }}
class Override{
         public static void main(String args[]){
                  B subob=new B(3,4,5);
                  subob.show();
         }
}
```

Output:
K: 5

Dynamic Method Dispatch

- Dynamic method dispatch is the mechanism by which a call to an **overridden** method is resolved at run time, rather than compile time.
- When an **overridden** method is called through a superclass reference, the method to execute will be based upon the type of the object being referred to at the time the call occurs. Not the type of the reference variable.

```
class A{
      void callme(){
      System.out.println("Inside A's callme method");
      }}
class B extends A{
      void callme(){
      System.out.println("Inside B's callme method");
      }}
class C extends A{
      void callme(){
      System.out.println("Inside C's callme method");
      }}
```

```
class Dispatch
{
public static void main(String args[])
{
          A a=new A();
          B b=new B();
          C c=new C();
          A r;
           r=a;
          r.callme();
           r=b;
          r.callme();
           r=c;
          r.callme();
          }
}
```

Output:

```
Inside A's callme method
Inside B's callme method
Inside C's callme method
```

Abstract Classes

- A method that has been declared but not defined is an abstract method.
- Any class that contains one or more abstract methods must also be declared abstract.
- You must declare the abstract method with the keyword abstract:
  - abstract type name (parameter-list);
- You must declare the class with the keyword abstract:

```
abstract class MyClass
{
        ......
}
```

- An abstract class is incomplete, It has "missing" method bodies.
- You cannot instantiate (create a new instance of) an abstract class but you can create reference to an abstract class.
- Also, you cannot declare abstract constructors, or abstract static methods.

- You can extend (subclass) an abstract class.
- If the subclass defines all the inherited abstract methods, it is “complete” and can be instantiated.
- If the subclass does not define all the inherited abstract methods, it is also an abstract class

```
abstract class A
{
    abstract void callme();
    // concrete methods are still allowed in abstract classes
    void callmetoo() {
        System.out.println("This is a concrete method.");
    }
}
class B extends A
{
    void callme() {
    System.out.println("B's implementation of callme.");
    }
}
class AbstractDemo
{
    public static void main(String args[]) {
        B b = new B();
        b.callme();
        b.callmetoo();
    }
}
```

Output:

```
    B's implementation of callme.
    This is a concrete method.
```

The Object Class

- **Object** is a special class, defined by Java.
- **Object** is a superclass of all other classes.
- This means that a reference variable of type **Object** can refer to an object of any other class.
- **Object** defines the following methods:

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class getClass( )	Obtains the class of an object at run time
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.

Method	Purpose
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

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

Output:

```
Class of Object obj is : java.lang.String
366712642
```

Any questions
?

Thank you