Nadar Saraswathi College of Engineering & Technology

NSCET

E-LEARNING

PRESENTATION

LISTEN … LEARN… LEAD…

COMPUTER SCIENCE AND ENGINEERING

II YEAR / III SEMESTER

CS8392 – OBJECT ORIENTED PROGRAMMING

Mr.C.PRATHAP M.Tech.,(Phd).,
Assistant Professor
Nadar Saraswathi College of Engineering & Technology,
Vadapudupatti, Annanji (po), Theni – 625531.

UNIT 02

INHERITANCE AND INTERFACE

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

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

Moreover, you can add new methods and fields in your current class also

INHERITANCE

Inheritance in Java

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

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

Moreover, you can add new methods and fields in your current class also

Terms used in Inheritance

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

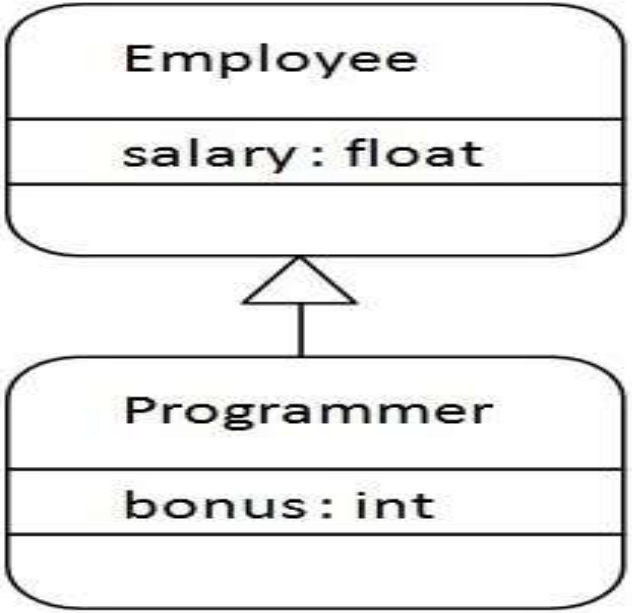
The syntax of Java Inheritance

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. The meaning of "extends" is to increase the functionality.

Java Inheritance Example



```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary
);
   System.out.println("Bonus of Programmer is:"+p.bo
nus
}
}
```

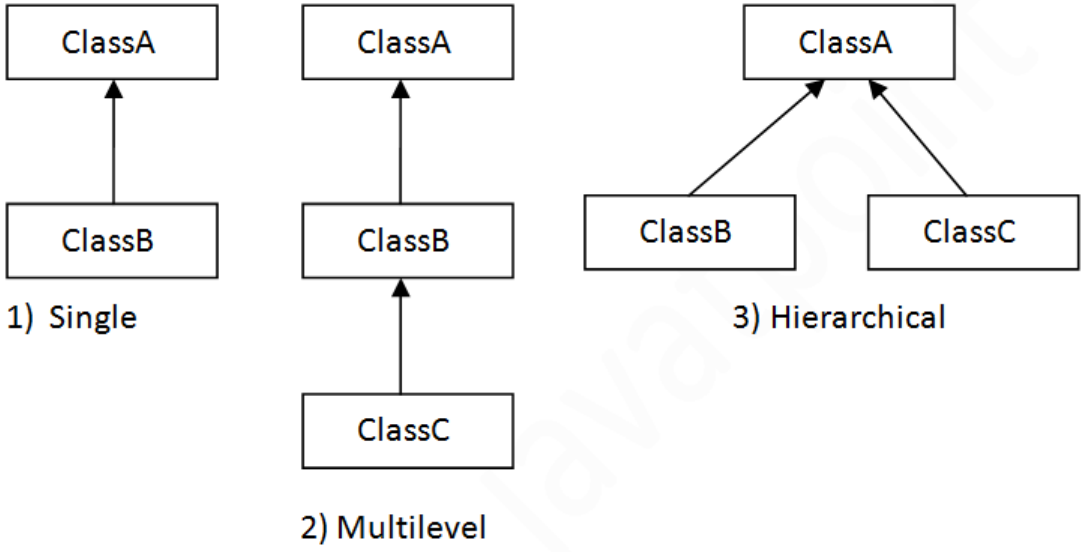
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

Types of inheritance in java

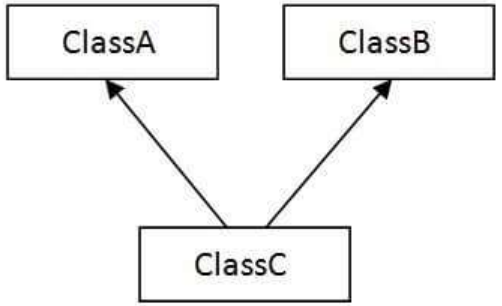
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

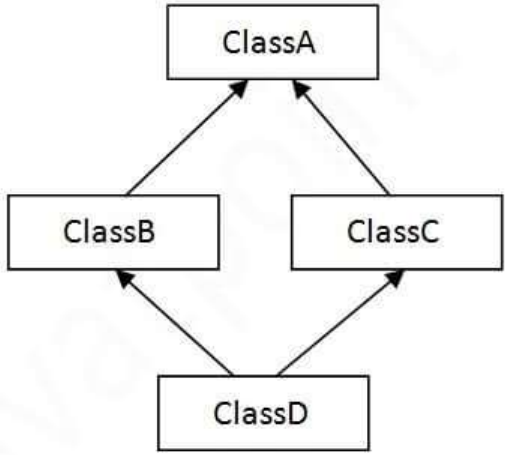
TYPES INHERITANCE



ACCESS THROUGH INTERFACE



4) Multiple



5) Hybrid

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 args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

OUTPUT:

barking...
eating..

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();
}}
```

OUTPUT:

weeping...
barking...
eating...

Hierarchical Inheritance Example

```
 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();
//c.bark();//C.T.Error
}}
```

OUTPUT:
meowing...
eating...

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 the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B
{
 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked
}
}
```

“super” KEYWORD Usage of super keyword

1. super() invokes the constructor of the parent class.

2. super.variable_name refers to the variable in the parent class.

3. super.method_name refers to the method of the parent class.

The Object Class

There is one special class, Object, defined by Java. All other classes are subclasses of Object. That is, 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. Also, since arrays are implemented as classes, a variable of type Object can also refer to any array. Object defines the following methods, which means that they are available in every object.

The Object Class

The methods getClass( ), notify( ), notifyAll( ), and wait( ) are declared as final. These methods are described elsewhere in this book. However, notice two methods now: equals( ) and toString( ).

The equals( ) method compares two objects. It returns true if the objects are equal, and false otherwise.

The precise definition of equality can vary, depending on the type of objects being compared.

The toString( ) method returns a string that contains a description of the object on which it is called. Many classes override this method.

Doing so allows them to tailor a description specifically for the types of objects that they create.

ABSTRACT CLASS

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

It needs to be extended and its method implemented. It cannot be instantiated.

Example1:

File: TestAbstraction1.java

```
abstract class Shape{ abstract void draw(); }
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");} }
 class Circle1 extends Shape{
 void draw(){System.out.println("drawing circle");} }
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1
s.draw(); } }
```

An interface in java is a blueprint of a class. It has static constants and abstract methods.
The interface in java is a mechanism to achieve abstraction and multiple inheritance.

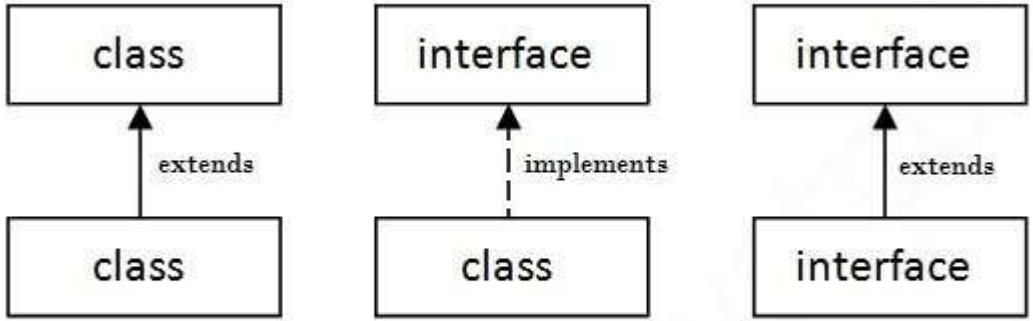
INTERFACE

INTERFACE IN JAVA

Syntax:

```
interface {
// declare constant fields
// declare methods that abstract
// by default. }
```

Relationship between classes and Interfaces

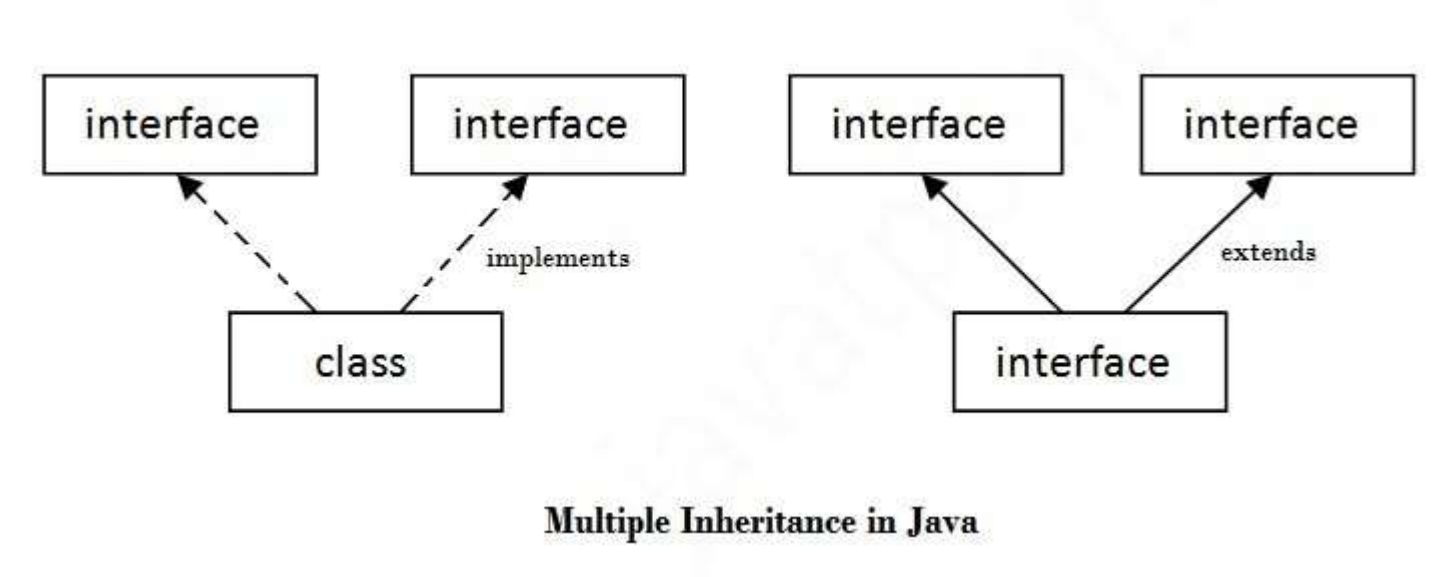


Example:

```
interface printable{
void print(); }
class A6 implements printable{
public void print(){
System.out.println("Hello");}
public static void main(String args[]){
A6 obj = new A6(); obj.print(); } }
```

Output: Hello

Multiple inheritance in Java by Interface



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

A class implements interface but one interface extends another interface .

```
interface Printable{
void print(); }
interface Showable extends Printable{
void show(); }
class TestInterface4 implements Showable{
public void print(){
System.out.println("Hello");}
public void show(){
System.out.println("Welcome");}
public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print(); obj.show(); }}
```

Output: Hello Welcome

Nested Interface in Java

An interface can have another interface i.e. known as nested interface. interface printable{
void print();
interface MessagePrintable{ void msg();
}
}

Advantages of interface in java:

Without bothering about the implementation part, we can achieve the security of implementation

In java, multiple inheritance is not allowed, however you can use interface to make use of it as you can implement more than one interface.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

FINAL KEYWORD

Final keyword can be used along with variables, methods and classes.

1) final variable
2) final method
3) final class

Java final variable A final variable is a variable whose value cannot be changed at anytime once assigned, it remains as a constant forever.

Example:

```
public class Travel {
final int SPEED=60;
void increaseSpeed(){
 SPEED=70; }
 public static void main(String args[]) {
Travel t=new Travel();
t.increaseSpeed();
} }
```

Java final method

When you declare a method as final, then it is called as final method. A final method cannot be overridden.

```
package com.javainterviewpoint;
class Parent {
 public final void disp() {
System.out.println("disp() method of parent class"); } }
 public class Child extends Parent {
public void disp() {
System.out.println("disp() method of child class"); }
public static void main(String args[]) {
Child c = new Child();
c.disp(); } }
```

Java final class

A final class cannot be extended(cannot be subclassed), lets take a look into the below example

```
package com.javainterviewpoint;
final class Parent { }
 public class Child extends Parent
public static void main(String args[]) {
Child c = new Child();
 } }
```

OBJECT CLONING

The object cloning is a way to create exact copy of an object.

The clone() method of Object class is used to clone an object.

The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create.

If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

The clone() method is defined in the Object class.

Syntax of the clone() method:

protected Object clone() throws CloneNotSupportedException

The clone() method saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning

Advantage of Object cloning

You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.

It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project.

Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done. Clone() is the fastest way to copy array.

Disadvantage of Object cloning

To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc. We have to implement cloneable interface while it doesn?t have any methods in it.

We just have to use it to tell the JVM that we can perform clone() on our object. Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.

Object.clone() doesn?t invoke any constructor so we don?t have any control over object construction.

Example of clone() method (Object cloning)

```
class Student implements Cloneable{
int rollno;
String name;
Student(int rollno,String name){ this.rollno=rollno; this.name=name; }
public Object clone()throws CloneNotSupportedException{
return super.clone(); }
 public static void main(String args[]){
try{ Student s1=new Student(101,"amit");
 Student s2=(Student)s1.clone();
System.out.println(s1.rollno+" "+s1.name);
System.out.println(s2.rollno+" "+s2.name); }
catch(CloneNotSupportedException c){} } }
```

INNER CLASSES

Inner class means one class which is a member of another class.
There are basically four types of inner classes in java.

1) Nested Inner class
2) Method Local inner classes
3) Anonymous inner classes
4) Static nested classes

Nested Inner class

Nested Inner class can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and default modifier. Like class, interface can also be nested and can have access specifiers.

```
class Outer { {
public void show() {
System.out.println("In a nested class method"); } } }
class Main {
public static void main(String[] args) {
Outer.Inner in = new Outer().new Inner();
in.show(); } }
```

Method Local inner classes

Inner class can be declared within a method of an outer class. In the following example, Inner is an inner class in outerMethod().

Example:

```
class Outer {
 void outerMethod() {
System.out.println("inside outerMethod"); // Inner class is local to outerMethod()
class Inner {
void innerMethod() {
 System.out.println("inside innerMethod"); } }
 Inner y = new Inner();
y.innerMethod(); } }
class MethodDemo {
public static void main(String[] args) {
 Outer x = new Outer();
x.outerMethod(); } }
```

Static nested classes

Static nested classes are not technically an inner class. They are like a static member of outer class

```
class Outer {
 private static void outerMethod() {
System.out.println("inside outerMethod"); } // A static inner class
static class Inner {
 public static void main(String[] args) {
System.out.println("inside inner class Method");
outerMethod(); } } }
```

Anonymous inner classes

Anonymous inner classes are declared without any name at all. They are created in two ways.

As subclass of specified type

```
class Demo {
void show() {
System.out.println("i am in show method of super class"); } }
 class Flavor1Demo { // An anonymous class with Demo as base
class static Demo d = new Demo() {
void show() { super.show();
System.out.println("i am in Flavor1Demo class"); } };
 public static void main(String[] args){
d.show(); } }
```

STRINGS IN JAVA

In java, string is basically an object that represents sequence of char values. Java String provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc. In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable. String s="javatpoint";

There are two ways to create String object:

1. By string literal 2. By new keyword

1) String Literal Java String literal is created by using double quotes. For Example: String s="welcome";

2) By new keyword String s=new String("Welcome");

Example:

```
public classstringmethod { public static void main(String[] args) {
String string1 = new String("hello");
String string2 = new String("hello");
if (string1 == string2) {
System.out.println("string1= "+string1+" string2= "+string2+" are
equal"); }
else {
System.out.println("string1= "+string1+" string2= "+string2+" are
Unequal"); }
System.out.println("string1 and string2 is=
"+string1.equals(string2));
String a= "information";
```

```
System.out.println("Uppercase of String a is= "+a.toUpperCase()); String
b="technology";
System.out.println("Concatenation of object a and b is= "+a.concat(b));
System.out.println("After concatenation Object a is= "+a.toString());
System.out.println("\"Joseph\'s\" is the greatest\\ college in chennai");
System.out.println("Length of Object a is= "+a.length());
System.out.println("The third character of Object a is= "+a.charAt(2));
StringBuffer n=new StringBuffer("Technology"); StringBuffer m=new
StringBuffer("Information"); System.out.println("Reverse of Object n is=
"+n.reverse()); n= new StringBuffer("Technology");
System.out.println("Concatenation of Object m and n is= "+m.append(n));
System.out.println("After concatenation of Object m is= "+m);
```

Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are: Java ArrayList class can contain duplicate elements.

Java ArrayList class maintains insertion order.

Java ArrayList class is non synchronized. Java ArrayList allows random access because array works at the index basis.

In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

ArrayList class declaration Syntax:

public class ArrayList extends AbstractList implements List, RandomAccess, Clone able, Serializable

Java ArrayList Example

```
import java.util.*; class Book {
 int id; String name,author,publisher;
int quantity; public Book(int id, String name, String author, String publisher, int quantity)
{ this.id = id; this.name = name; this.author = author; this.publisher = publisher;
this.quantity = quantity; } }
 public class ArrayListExample {
public static void main(String[] args) { //Creating list of Books List list=new ArrayList();
//Creating Books Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw
Hill",4);
Book b3=new Book(103,"Operating System","Galvin","Wiley",6); //Adding Books to list
list.add(b1); list.add(b2);
list.add(b3); //Traversing list for(Book b:list){
System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity); } } }
```