



# JAVA BASICS

4KCJ001 - LECTURE 4

# INDEX

- ❖ Inheritance
- ❖ Super Keyword
- ❖ Instanceof Operator
- ❖ Abstract Classes
- ❖ Overriding
- ❖ Rules of Overriding



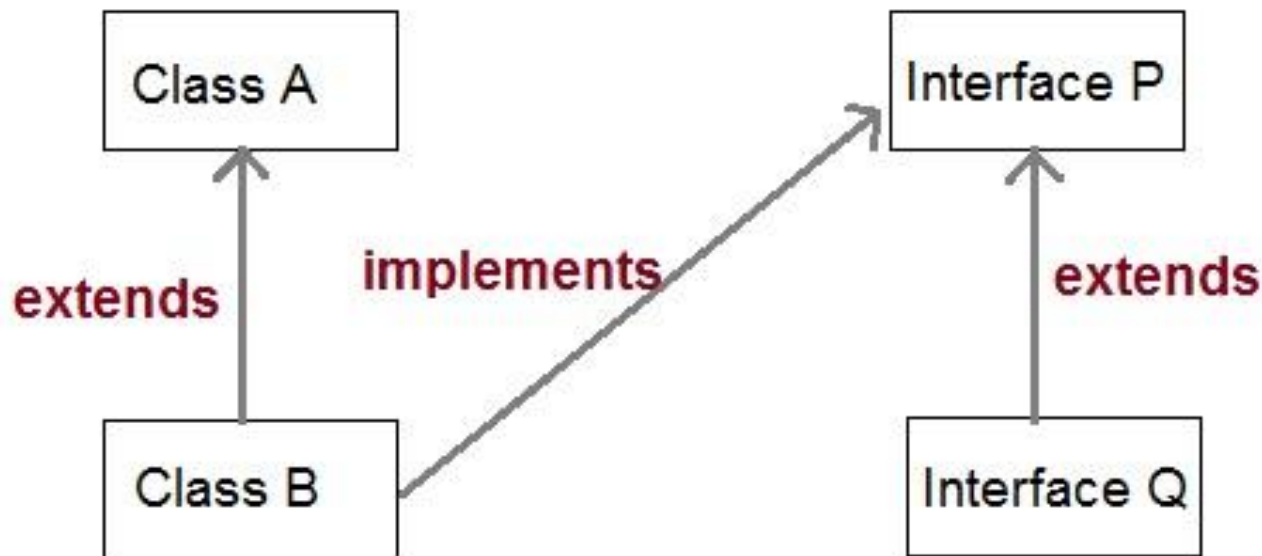
# INHERITANCE (IS-A)

- ❖ **Inheritance** is one of the key features of Object Oriented Programming.
- ❖ Inheritance provided mechanism that allowed a class to inherit property of another class.
- ❖ When a Class extends another class it inherits all non-private members including fields and methods.
- ❖ Inheritance in Java can be best understood in terms of Parent and Child relationship, also known as **Super class(Parent)** and **Sub class(child)** in Java language.

# INHERITANCE (IS-A)

❖ Inheritance defines is-a relationship between a Super class and its Sub class.

❖ **extends** and **implements** keywords are used to describe inheritance in Java.



# INHERITANCE

```
class Vehicle.  
{  
    .....  
}  
class Car extends Vehicle  
{  
    .....    //extends the property of vehicle class.  
}
```

- ❖ Now based on above example. In OOPs term we can say that,
- ❖ Vehicle is super class of Car.
- ❖ Car is sub class of Vehicle.
- ❖ Car **IS-A** Vehicle.

# PURPOSE OF INHERITANCE

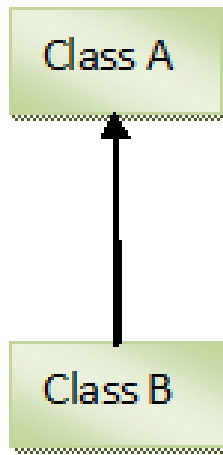
- ❖ It promotes the code **reusability** i.e. the same methods and variables which are defined in a parent/super/base class can be used in the child/sub/derived class.
- ❖ It promotes polymorphism by allowing method overriding.
- ❖ Main **disadvantage** of using inheritance is that the two classes (parent and child class) **gets tightly coupled**.
- ❖ This means that if we change code of parent class, it will affect to all the child classes which is inheriting/deriving the parent class, and hence, it **cannot be independent** of each other.



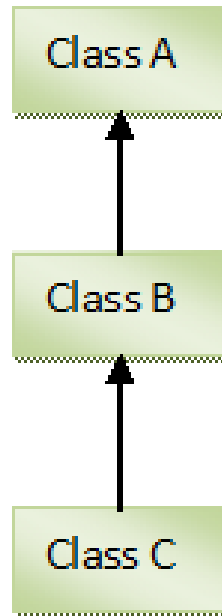
# INHERITANCE - EXAMPLE

```
class Parent
{
    public void p1()
    {
        System.out.println("Parent method");
    }
}
public class Child extends Parent {
    public void c1()
    {
        System.out.println("Child method");
    }
    public static void main(String[] args)
    {
        Child cobj = new Child();
        cobj.c1();    //method of Child class
        cobj.p1();    //method of Parent class
    }
}
```

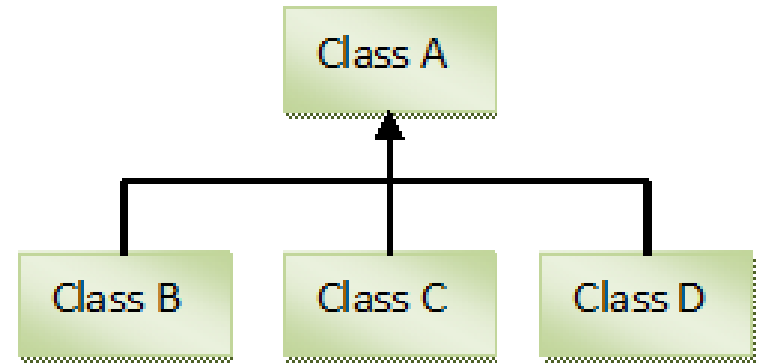
# TYPES OF INHERITANCE IN JAVA



Single Inheritance



Multilevel Inheritance

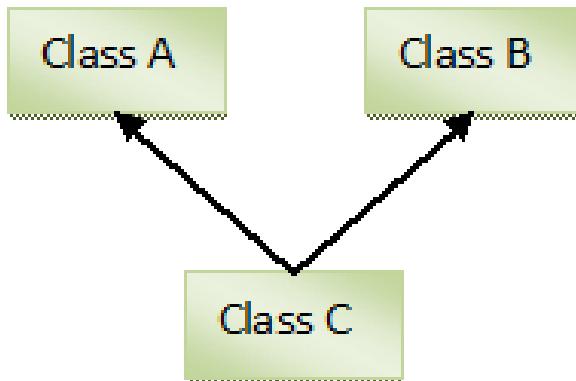


Hierarchical Inheritance

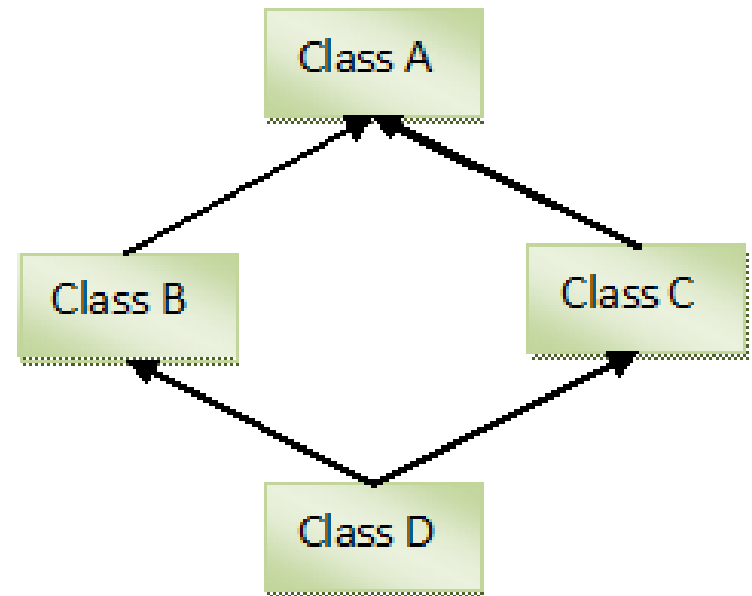


# INHERITANCE NOT SUPPORTED IN JAVA

Multiple inheritance is not supported in java.



Multiple Inheritance



Hybrid Inheritance

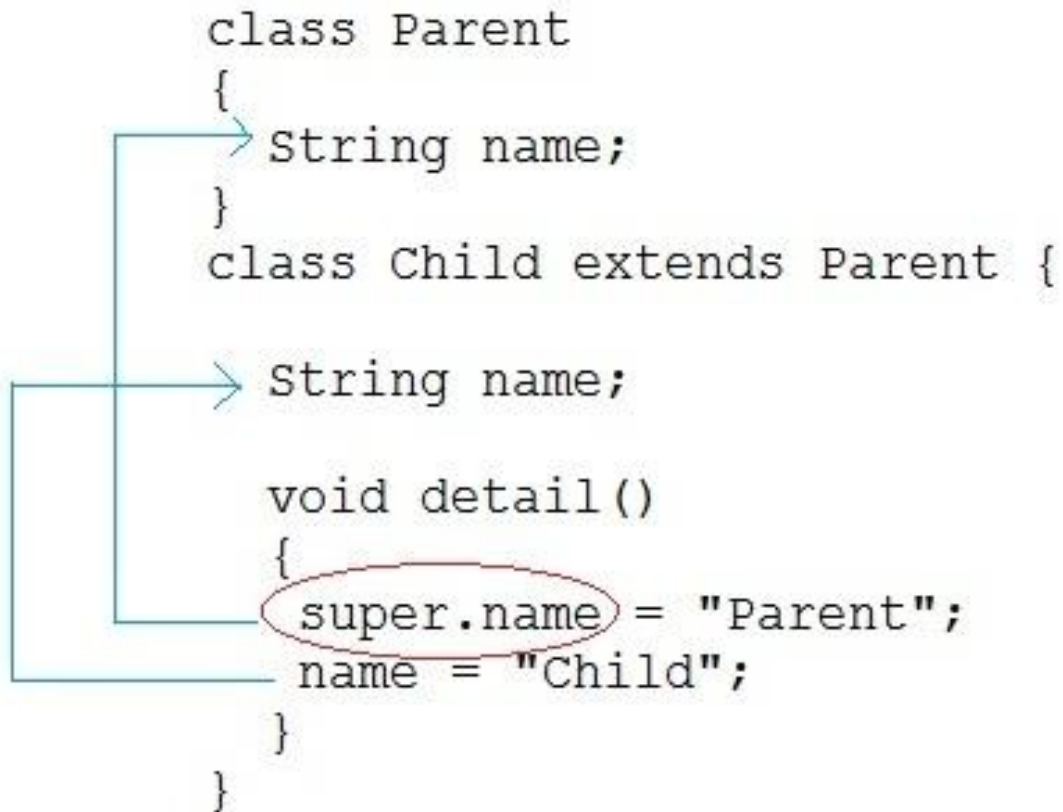
# SUPER KEYWORD

- ❖ In Java, **super** keyword is used to refer to immediate parent class of a child class.
- ❖ In other words super keyword is used by a subclass whenever it need to refer to its immediate super class.
- ❖ Usage of java super Keyword
  - ❖ super can be used to refer immediate parent class instance variable.
  - ❖ super can be used to invoke immediate parent class method.
  - ❖ super() can be used to invoke immediate parent class constructor.

# SUPER KEYWORD

```
class Parent
{
    String name;
}
class Child extends Parent {
    String name;

    void detail()
    {
        super.name = "Parent";
        name = "Child";
    }
}
```

A diagram illustrating the use of the 'super' keyword in Java. It shows two classes: 'Parent' and 'Child'. 'Parent' has a 'String name' attribute. 'Child' extends 'Parent' and also has a 'String name' attribute. The 'Child' class has a 'detail()' method. Inside 'detail()', the line 'super.name = "Parent";' is circled in red. Blue arrows indicate the following: one arrow points from the 'super.name' in the circled line to the 'name' attribute of the 'Parent' class; another arrow points from the 'name' attribute of the 'Child' class to the 'name' attribute of the 'Child' class; and a third arrow points from the 'detail()' method of the 'Child' class to the 'detail()' method of the 'Child' class.

# METHOD OVERRIDING

- ❖ When a class extends its super class, all or some members of super class are inherited to sub class.
- ❖ When a inherited super class method is modified in the sub class, then we call it as **method overriding** .
- ❖ Through method overriding, we can modify super class method according to requirements of sub class.

# RULES OF METHOD OVERRIDING

❖ **Name of the overridden method** must be same as in the super class. You can't change name of the method in subclass.

## ❖ **Return Type Of Overridden Method :**

❖ The return type of the overridden method must be compatible with super class method.

❖ If super class method has primitive data type as its return type, then overridden method must have same return type in sub class also.

❖ If super class method has derived or user defined data type as its return type, then return type of sub class method must be of same type or its sub class.

# RULES OF METHOD OVERRIDING

## ❖ **Visibility Of Overridden method :**

- ❖ You can keep same visibility or increase the visibility of overridden method but you can't reduce the visibility of overridden methods in the subclass.
- ❖ For example, default method can be overridden as default or protected or public method but not as private

## ❖ **Arguments Of Overridden Methods :**

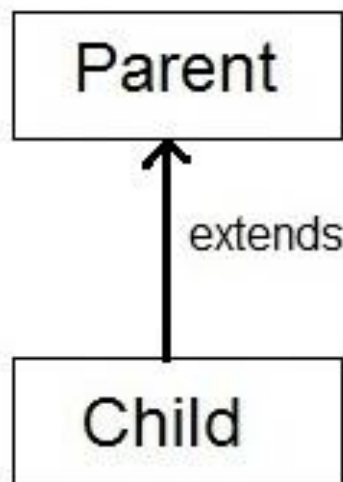
- ❖ For method to be properly overridden , You must not change arguments of method in subclass.
- ❖ If you change the number of arguments or types of arguments of overridden method in the subclass, then method will be overloaded not overridden .

# INSTANCEOF OPERATOR

- ❖ In Java, **instanceof** operator is used to check the type of an object at runtime.
- ❖ It is the means by which your program can obtain run-time type information about an object.
- ❖ **instanceof** operator is also important in case of casting object at runtime.
- ❖ **instanceof** operator return **boolean** value, if an object reference is of specified type then it return **true** otherwise **false**.



# INSTANCE OF OPERATOR



```
Parent p = new Child( );
```

Upcasting

```
Child c = new Parent( );
```

Compile time error

```
Child c = ( Child ) new Parent( );
```

Downcasting but throws  
ClassCastException at runtime.

# ABSTRACT CLASS

- ❖ If a class contain any abstract method then the class is declared as abstract class.
- ❖ An abstract class is never instantiated. It is used to provide abstraction.
- ❖ Although it does not provide 100% abstraction because it can also have concrete method.

```
abstract class class_name { }
```

# ABSTRACT CLASSES

- ❖ Abstract classes are not Interfaces.
- ❖ An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
- ❖ Abstract classes can have Constructors, Member variables and Normal methods.
- ❖ Abstract classes are never instantiated.
- ❖ When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

# ABSTRACT METHOD

- ❖ Method that are declared without any body within an abstract class are called abstract method.
- ❖ The method body will be defined by its subclass.
- ❖ Abstract method can never be final and static.
- ❖ Any class that extends an abstract class must implement all the abstract methods declared by the super class.

```
abstract return_type function_name ();    // No  
definition
```

# ABSTRACTION USING ABSTRACT CLASS

- ❖ Abstraction is an important feature of OOPS.
- ❖ It means hiding complexity.
- ❖ Abstract class is used to provide abstraction.
- ❖ Although it does not provide 100% abstraction because it can also have concrete method.

# INTERFACE

- ❖ Interface is a **pure abstract** class.
- ❖ They are syntactically similar to classes, but you cannot create instance of an Interface and their methods are declared without any body.
- ❖ Interface is used to achieve complete abstraction in Java.
- ❖ When you create an interface it defines what a class can do without saying anything about how the class will do it.
- ❖ Abstract classes may contain both abstract methods as well as concrete methods.
- ❖ But interfaces must contain **only abstract methods**. Concrete methods are not allowed in interfaces.

# INTERFACE

❖ Interfaces are declared with keyword '**interface**' and interfaces are implemented by the class using '**implements**' keyword.

```
interface InterfaceClass
{
    //Some Abstract methods
}

class AnyClass implements InterfaceClass
{
    //Use 'implements' while implementing Interfaces
    //Don't use 'extends'
}
```



# INTERFACE

- ❖ By default, Every field of an interface is public, static and final. You can't use any other modifiers other than these three for a field of an interface.
- ❖ You can't change the value of a field once they are initialized. Because they are static and final. Therefore, sometimes fields are called as **Constants**.
- ❖ By default, All **methods** of an interface are **public** and **abstract**.
- ❖ Like classes, for every interface **.class** file will be generated after compilation.
- ❖ While implementing any interface methods inside a class, that method must be declared as **public**.

# INTERFACE

❖ SIB – Static Initialization Block and IIB – Instance Initialization Block are not allowed in interfaces.

```
interface InterfaceClassOne
{
    static
    {
        //compile time error
        //SIB's are not allowed in interfaces
    }

    {
        //Here also compile time error.
        //IIB's are not allowed in interfaces
    }

    void methodOne(); //abstract method
}
```

# INTERFACE

❖ As we all know that, any class in java can not extend more than one class. But class can implement more than one interfaces. This is how multiple inheritance is implemented in java.

```
class AnyClass implements InterfaceClassOne, InterfaceClassTwo
{
    public void methodOne()
    {
        //method of first interface is implemented
    }

    //method of Second interface must also be implemented.
    //Otherwise, you have to declare this class as abstract.

    public void methodTwo()
    {
        //Now, method of Second interface is also implemented.
        //No need to declare this class as abstract
    }
}
```

# INTERFACE

interface Moveable

```
int AVERAGE-SPEED=40;  
void move();
```

what you declare

interface Moveable

```
public static final int AVERAGE-SPEED=40;  
public abstract void move();
```

what the compiler  
sees

# IMPLEMENTING INTERFACES

- ❖ A class can choose to implement only some of the methods of its interfaces. The class then be declared as **abstract**.
- ❖ The interface methods cannot be declared *static* because they comprise the *contract* fulfilled by the objects of the class implementing the interface.
- ❖ Interface methods are always implemented as instance methods.

# EXTENDING INTERFACES

- ❖ An interface can extend other interfaces using extend clause. Unlike extending classes, an interface can extend several interfaces.
- ❖ The interfaces extended by an interface (directly or indirectly) are called *superinterfaces*.
- ❖ A **subinterface** inherits all methods from its **superinterfaces**, as their method declarations are all implicitly public.

# DOES AN INTERFACE EXTEND OBJECT CLASS IN JAVA.?

❖ If an interface does not extend Object class, then why we can call methods of Object class on interface variable like below.

```
interface A
{
}

class InterfaceAndObjectClass
{
    public static void main(String[] args)
    {
        A a = null;

        a.equals(null);

        a.hashCode();

        a.toString();
    }
}
```



# DOES AN INTERFACE EXTEND OBJECT CLASS IN JAVA.?

❖ If an interface does not extend Object class, then why the methods of Object class are visible in interface.?

```
interface A
{
    @Override
    public boolean equals(Object obj);

    @Override
    public int hashCode();

    @Override
    public String toString();
}
```

# DOES AN INTERFACE EXTEND OBJECT CLASS IN JAVA.?

- ❖ This is because, for every public method in Object class, there is an implicit abstract and public method declared in every interface which does not have direct super interfaces.
- ❖ This is the standard Java Language Specification

# INTERFACES

❖ Interfaces allow you to use classes in different hierarchies, polymorphically. For example, say you have the following interface:

```
public interface Movable {  
    void move();  
}
```

❖ Any number of classes, across class hierarchies could implement Movable in their own specific way, yet still be used by some caller in a uniform way. So if you have the following two classes:

```
public class Car extends Vehicle implements Movable {  
    public void move() {  
        //implement move, vroom, vroom!  
    }  
}
```

# INTERFACES

```
public class Horse extends Animal implements Movable
{
    public void move() {
        //implement move, neigh!
    }
}
```

❖ From the perspective of the caller, it's just a Movable

```
❖ Movable movable = ...;
❖ movable.move();
```

# MARKER INTERFACES

❖ Marker Interface in java is an interface with no fields or methods within it. It is used to convey to the JVM that the class implementing an interface of this category will have some special behavior.

❖ Hence, an empty interface in java is called a marker interface. Marker interface is also called tag interface. In java we have the following major marker interfaces as under:

- ❖ Serializable interface
- ❖ Cloneable interface
- ❖ Remote interface
- ❖ ThreadSafe interface

## LAB4

- ❖ Q1. Create an abstract class 'Parent' with a method 'message'.
- ❖ It has two subclasses each having a method with the same name 'message' that prints "This is first subclass" and "This is second subclass" respectively.
- ❖ Call the methods 'message' by creating an object for each subclass.

## LAB4

- ❖ Q2. We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B.
- ❖ Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students.
- ❖ The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for each of the two classes and print the percentage of marks for both the students.



## LAB4

❖Q3. Create an abstract class 'Animals' with two abstract methods 'cats' and 'dogs'.

❖Now create a class 'Cats' with a method 'cats' which prints "Cats meow" and a class 'Dogs' with a method 'dogs' which prints "Dogs bark", both inheriting the class 'Animals'.

❖Now create an object for each of the subclasses and call their respective methods.