

Lecture 6



PROGRAMMING LANGUAGE

BY: MOHAMMED ABDELFATTAH

Objectives

➤ Inheritance

➤ super Keyword

- ✓ Variables
- ✓ Methods
- ✓ Constructors

➤ Inheritance Types:

- ✓ Single Inheritance
- ✓ Multilevel Inheritance
- ✓ Hierarchical Inheritance
- ✓ Multiple Inheritance

➤ Polymorphism

➤ Polymorphism Types:

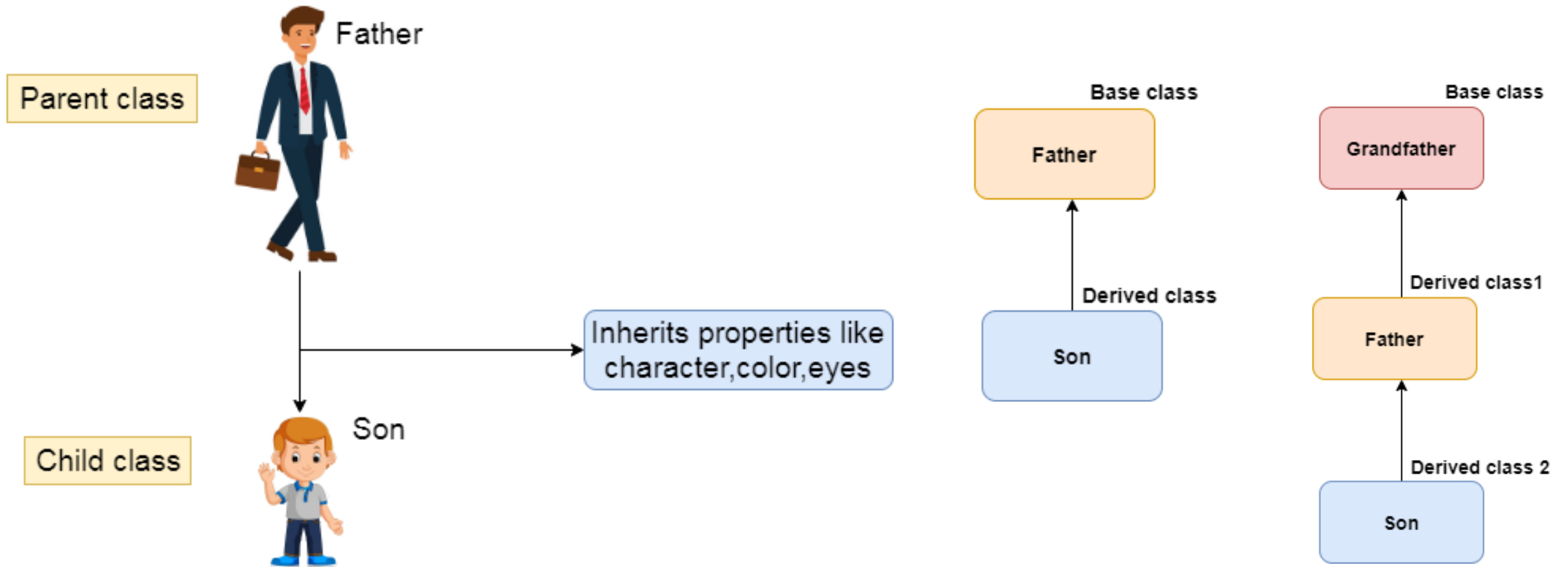
- ✓ Compile Time Polymorphism
- ✓ Run Time Polymorphism

➤ final Keyword:

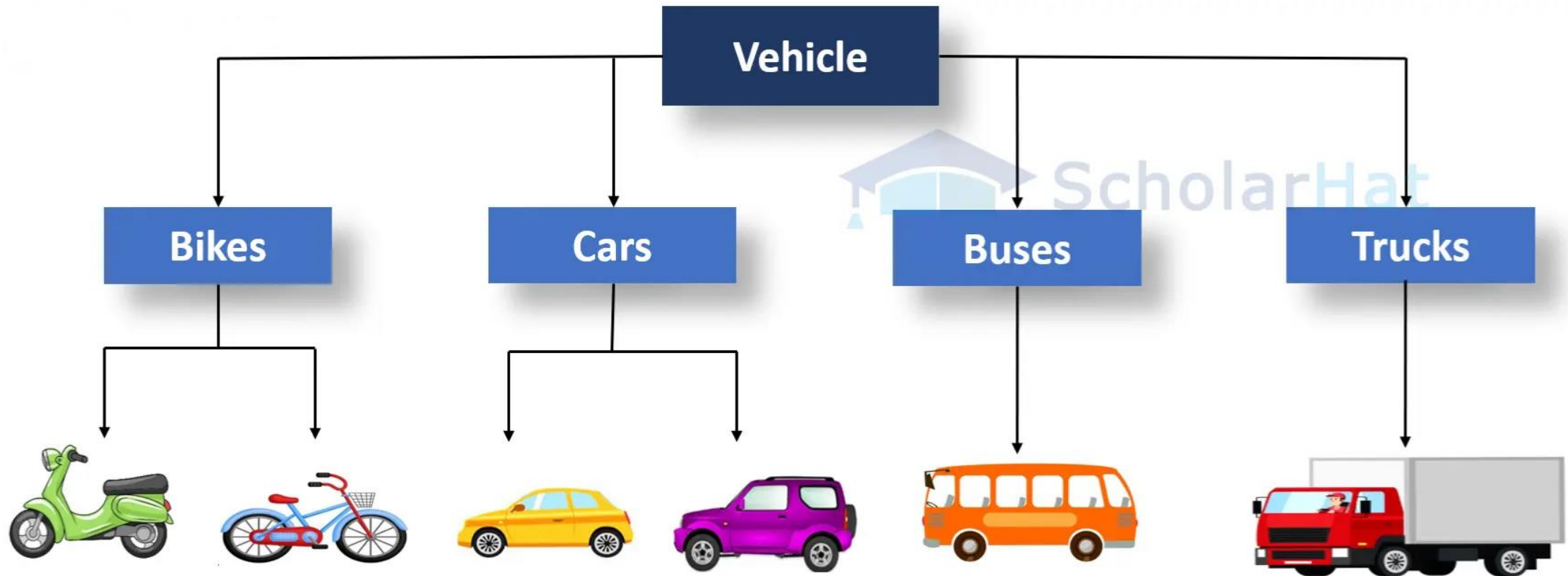
- ✓ Variables
- ✓ Methods
- ✓ Classes

➤ Method Overriding Rules

Inheritance



Inheritance

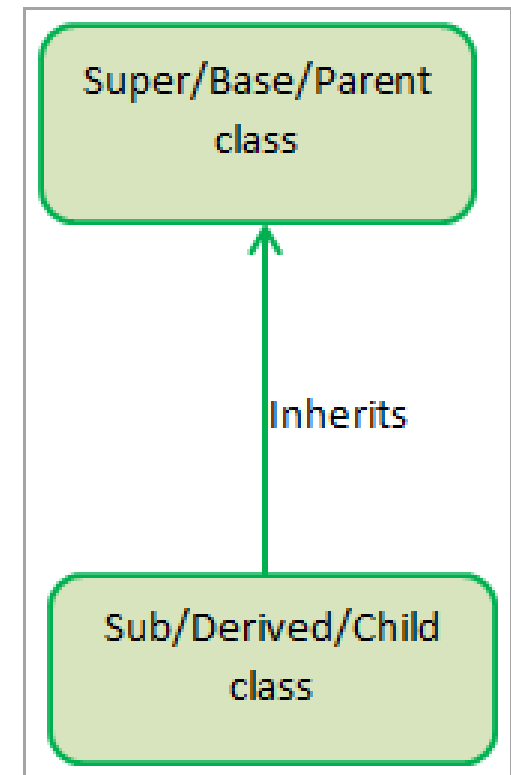


Inheritance

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 from.

To inherit from a class, use the **extends** keyword.



Example 1

In the example below, the Car class (subclass) inherits the attributes and methods from the Vehicle class (superclass).

```
class Vehicle {  
    // Vehicle attribute  
    protected String brand = "Hyundai";  
  
    // Vehicle method  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}
```

```
class Car extends Vehicle {  
    private String modelName = "Accent";    // Car attribute  
    public static void main(String[] args) {  
  
        Car myCar = new Car();    // Create a myCar object  
  
        // Call the honk() method (from the Vehicle class) on the myCar object  
        myCar.honk();  
  
        // Display the value of the brand attribute (from the Vehicle class) and  
        // the value of the modelName from the Car class  
        System.out.println(myCar.brand + " " + myCar.modelName);  
    }  
}
```

Did you notice the protected modifier in Vehicle?

We set the brand attribute in Vehicle to a **protected** access modifier. If it was set to **private**, the Car class would not be able to access it.

Why And When To Use "Inheritance"?

- It is useful for **code reusability**: reuse attributes and methods of an existing class when you create a new class.

// Output

Tuut, tuut!

Hyundai Accent

Example 2

```
class Calculation {
    int z;

    public void addition(int x, int y) {
        z = x + y;
        System.out.println("The sum of the given numbers:"+z);
    }

    public void Subtraction(int x, int y) {
        z = x - y;
        System.out.println("The difference between the given numbers:"+z);
    }
}
```

// Output

The sum of the given numbers:30

The difference between the given numbers:10

The product of the given numbers:200

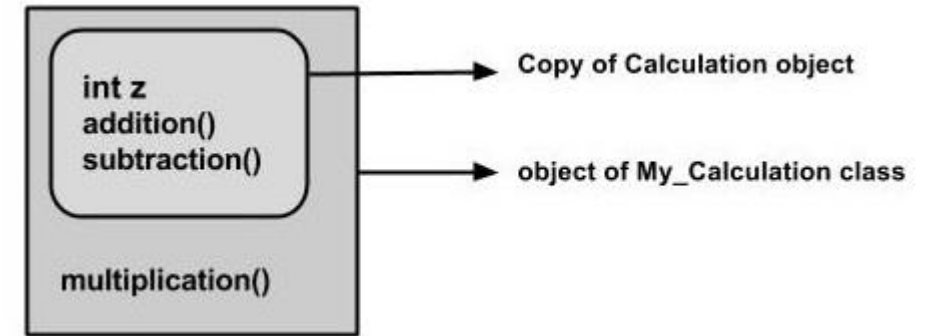
```
public class My_Calculation extends Calculation {
    public void multiplication(int x, int y) {
        z = x * y;
        System.out.println("The product of the given numbers:"+z);
    }

    public static void main(String args[]) {
        int a = 20, b = 10;
        My_Calculation demo = new My_Calculation();
        demo.addition(a, b);
        demo.Subtraction(a, b);
        demo.multiplication(a, b);
    }
}
```

Example 2

```
class Calculation {  
    int z;  
  
    public void addition(int x, int y) {  
        z = x + y;  
        System.out.println("The sum of the given numbers:"+z);  
    }  
  
    public void Subtraction(int x, int y) {  
        z = x - y;  
        System.out.println("The difference between the given numbers:"+z);  
    }  
}
```

```
Calculation demo = new My_Calculation();  
demo.addition(a, b);  
demo.Subtraction(a, b);
```



```
public class My_Calculation extends Calculation {  
    public void multiplication(int x, int y) {  
        z = x * y;  
        System.out.println("The product of the given numbers:"+z);  
    }  
  
    public static void main(String args[]) {  
        int a = 20, b = 10;  
        My_Calculation demo = new My_Calculation();  
        demo.addition(a, b);  
        demo.Subtraction(a, b);  
        demo.multiplication(a, b);  
    }  
}
```


Inheritance

If you don't want other classes to inherit from a class, use the **final** keyword.

If you try to access a **final** class, Java will generate an error:

```
final class Vehicle
{
    ...
}
```

```
class Car extends Vehicle
{
    ...
}
```

// Output

Main.java:9: error: cannot inherit from final Vehicle
class Main extends Vehicle {

^

1 error)

super Keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

- ❑ It is used to **differentiate the members of superclass** from the members of subclass, if they have same names.
- ❑ It is used to **invoke the superclass constructor** from subclass.

Example 1

```
class Super_class {  
    int num = 20;  
  
    // display method of superclass  
    public void display() {  
        System.out.println("superclass method ");  
    }  
}
```

// Output

Subclass method

superclass method

num for subclass method: 10

num for subclass method: 20

```
public class Sub_class extends Super_class {  
  
    int num = 10;  
    // display method of sub class  
    public void display() {  
        System.out.println("subclass method ");  
    }  
  
    public void my_method() {  
        Sub_class sub = new Sub_class(); // Instantiating subclass  
  
        // Invoking the display() method of sub class  
        sub.display();  
  
        // Invoking the display() method of superclass  
        super.display();  
  
        // printing the value of variable num of subclass  
        System.out.println("num for subclass method: " + sub.num);  
  
        // printing the value of variable num of superclass  
        System.out.println("num for super method: " + super.num);  
    }  
  
    public static void main(String args[]) {  
        Sub_class obj = new Sub_class();  
        obj.my_method();  
    }  
}
```

Example 2

Invoking Superclass Constructor.

```
class Superclass {
    int age;

    Superclass(int age) {
        this.age = age;
    }

    public void getAge() {
        System.out.println("Age value in super class is: " +age);
    }
}
```

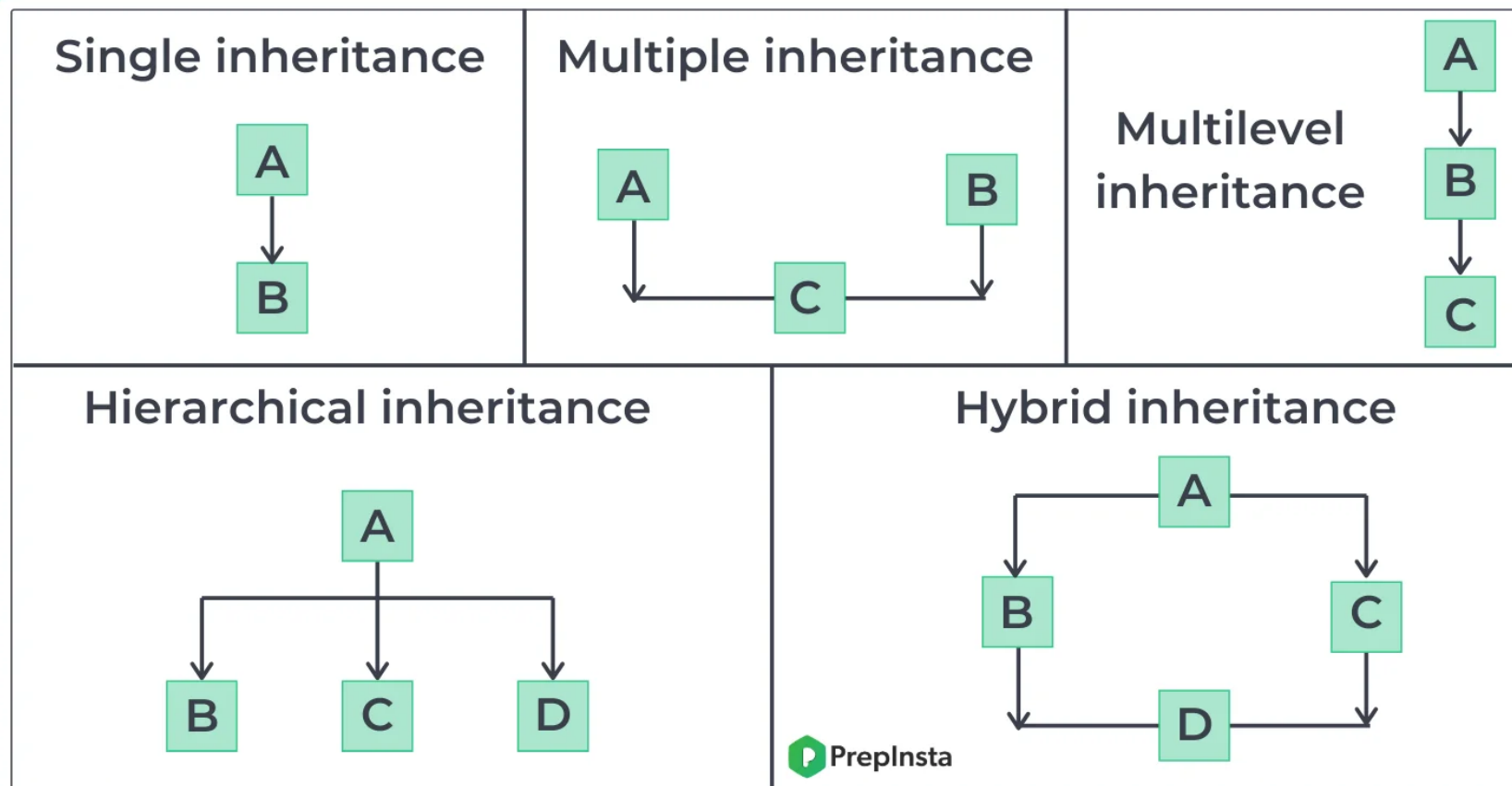
```
public class Subclass extends Superclass {
    Subclass(int age) {
        super(age);
    }

    public static void main(String args[]) {
        Subclass s = new Subclass(24);
        s.getAge();
    }
}
```

// Output
Age value in super class is: 24

Inheritance Types

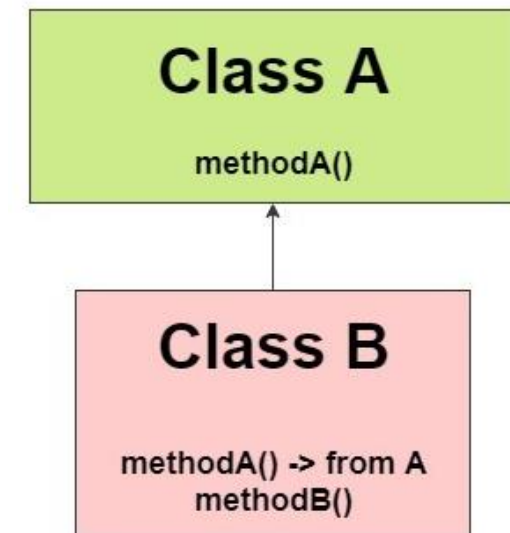
In Java, there are mainly three types of inheritances **Single**, **Multilevel**, and **Hierarchical**. Java does not support **Multiple** and **Hybrid** inheritance.



Single Inheritance

The inheritance in which there is only one base class and one derived class is known as single inheritance. The single (or, single-level) inheritance inherits data from only one base class to only one derived class.

```
class One {  
    public void printOne() {  
        System.out.println("printOne() method of One class.");  
    }  
}  
  
public class Main extends One {  
    public static void main(String args[]) {  
        // Creating object of the derived class (Main)  
        Main obj = new Main();  
  
        // Calling method  
        obj.printOne();  
    }  
}
```



// Output
printOne() method of One class.

Example

```
class Animal{
    void eat(){System.out.println("eating...");}
}

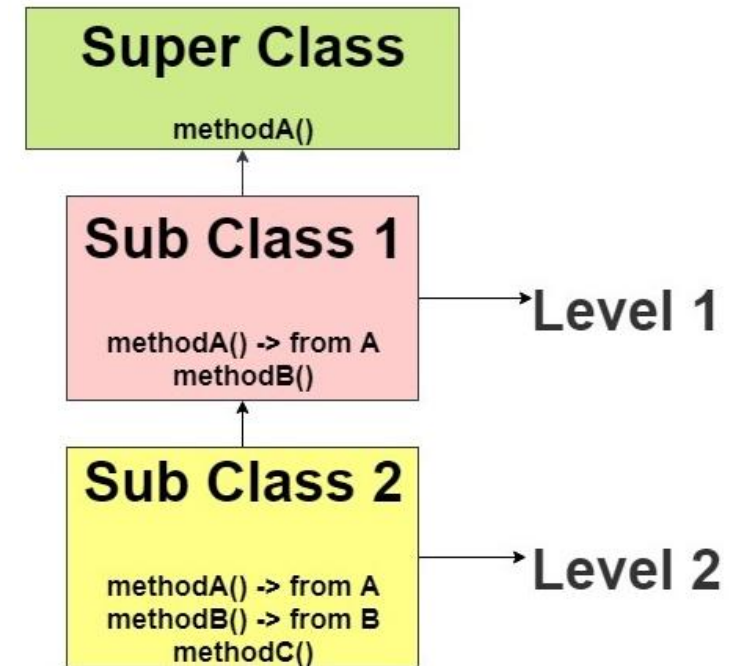
class Dog extends Animal{
    void bark(){System.out.println("barking...");}
}

class TestInheritance{
    public static void main(String args[]){
        Dog d=new Dog();
        d.bark();
        d.eat();
    }
}
```

Multilevel Inheritance

The inheritance in which a base class is inherited to a derived class and that derived class is further inherited to another derived class is known as multi-level inheritance. Multilevel inheritance involves multiple base classes.

```
class One {  
    public void printOne() {  
        System.out.println("printOne() method of One class.");  
    }  
}  
  
class Two extends One {  
    public void printTwo() {  
        System.out.println("printTwo() method of Two class.");  
    }  
}  
  
public class Main extends Two {  
    public static void main(String args[]) {  
        // Creating object of the derived class (Main)  
        Main obj = new Main();  
  
        // Calling methods  
        obj.printOne();  
        obj.printTwo();  
    }  
}
```



// Output
printOne() method of One class.
printTwo() method of Two class.

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();
    }
}
```

Hierarchical Inheritance

The inheritance in which only one base class and multiple derived classes is known as hierarchical inheritance.

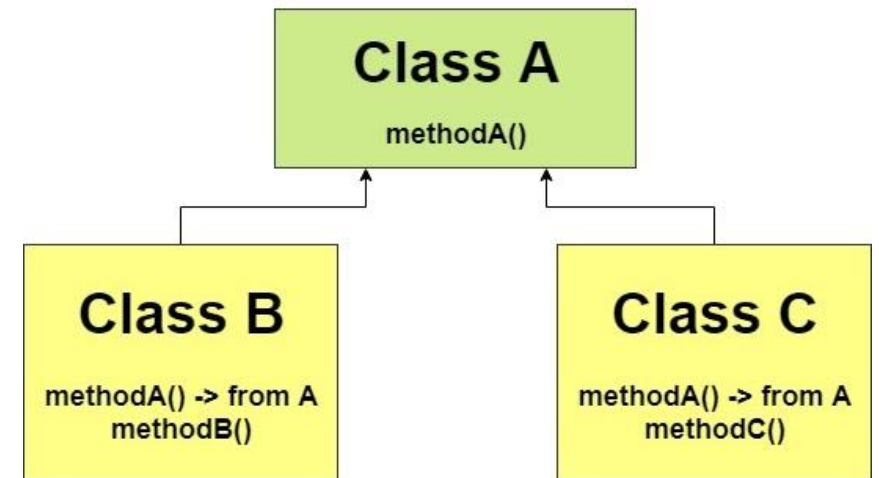
```
// Base class
class One {
    public void printOne() {
        System.out.println("printOne() Method of Class One");
    }
}

// Derived class 1
class Two extends One {
    public void printTwo() {
        System.out.println("Two() Method of Class Two");
    }
}

// Derived class 2
class Three extends One {
    public void printThree() {
        System.out.println("printThree() Method of Class Three");
    }
}
```

// Output

```
printOne() Method of Class One
printOne() Method of Class One
```



```
// Testing Class
public class Main {
    public static void main(String args[]) {
        Two obj1 = new Two();
        Three obj2 = new Three();

        //All classes can access the method of class One
        obj1.printOne();
        obj2.printOne();
    }
}
```

Example

```
class Animal{
    void eat(){System.out.println("eating...");}
}
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();    //Error
    }
}
```

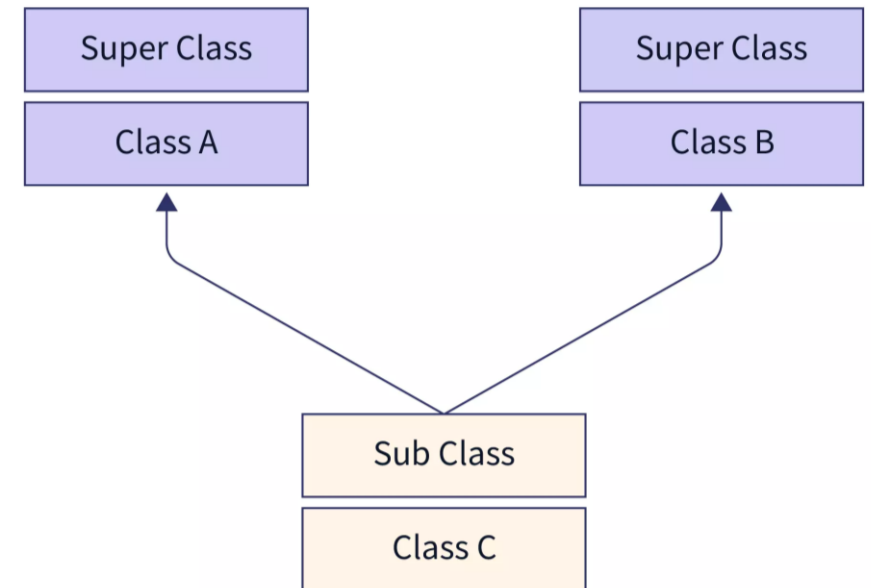
Why multiple inheritance is not supported?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

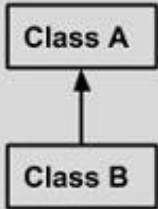
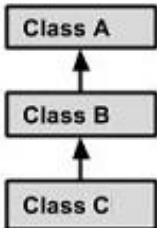
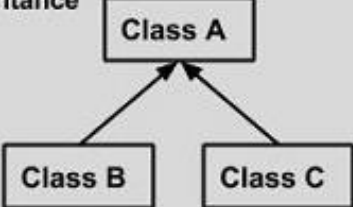
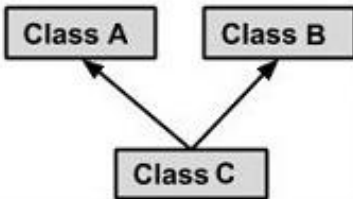
Consider a scenario where **A**, **B**, and **C** are three classes.

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.



Inheritance Types

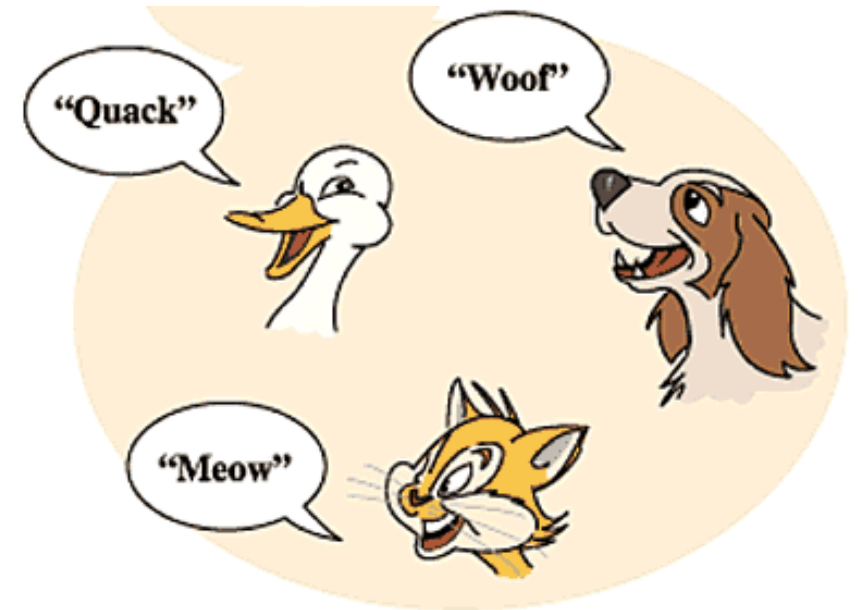
Single Inheritance	 <pre>graph BT; B[Class B] --> A[Class A]</pre>	<pre>public class A { } public class B extends A { }</pre>
Multi Level Inheritance	 <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre>	<pre>public class A {} public class B extends A {.....} public class C extends B {..... }</pre>
Hierarchical Inheritance	 <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre>	<pre>public class A {} public class B extends A {.....} public class C extends A {..... }</pre>
Multiple Inheritance	 <pre>graph BT; C[Class C] --> A[Class A]; C --> B[Class B]</pre>	<pre>public class A {} public class B {.....} public class C extends A,B { } // Java does not support mutiple Inheritance</pre>

Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Polymorphism uses those methods to perform different tasks.

This allows us to perform a single action in different ways.



Polymorphism

```
class Animal {  
    public void animalSound() {  
        System.out.println("The animal makes a sound");  
    }  
}
```

```
class Cat extends Animal {  
    public void animalSound() {  
        System.out.println("The cat says: Meow Meow");  
    }  
}
```

```
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("The dog says: Woof Woof");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Animal myAnimal = new Animal(); // Create a Animal object  
        Animal myCat = new Cat(); // Create a Cat object  
        Animal myDog = new Dog(); // Create a Dog object  
        myAnimal.animalSound();  
        myCat.animalSound();  
        myDog.animalSound();  
    }  
}
```

// Output

The animal makes a sound

The cat says: Meow Meow

The dog says: Woof Woof

Polymorphism Types

There are two types of polymorphism in Java:

❑ Compile Time Polymorphism

- Compile-time polymorphism is also known as **static polymorphism** and it is implemented by **method overloading**.

❑ Run Time Polymorphism

- Run time polymorphism is also known as **dynamic method** dispatch and it is implemented by the **method overriding**.

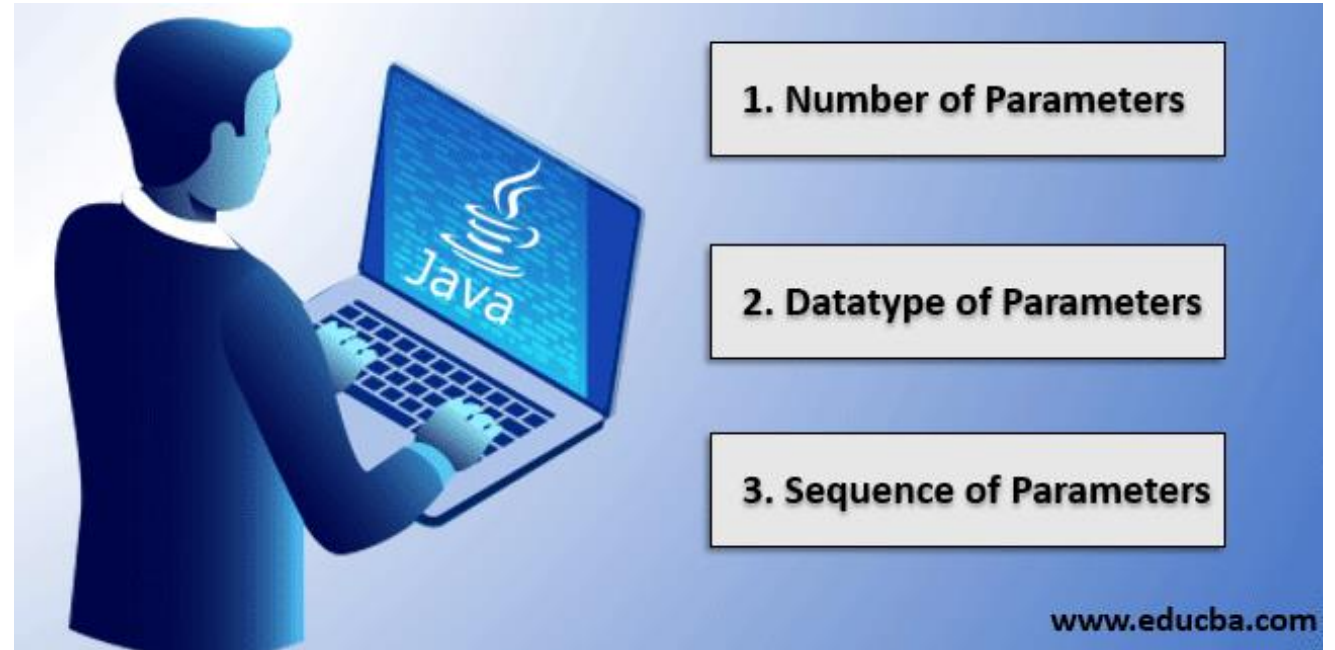
Method Overloading Types

```
int myMethod(int x)
```

```
float myMethod(float x)
```

```
double myMethod(int x, double y)
```

```
double myMethod(double x, double y)
```



Example 1: Compile Time Polymorphism

```
// Java Example: Compile Time Polymorphism
public class Main {
    // method to add two integers
    public int addition(int x, int y) {
        return x + y;
    }

    // method to add three integers
    public int addition(int x, int y, int z) {
        return x + y + z;
    }

    // method to add two doubles
    public double addition(double x, double y) {
        return x + y;
    }
}
```

```
// Main method
public static void main(String[] args) {
    // Creating an object of the Main method
    Main number = new Main();

    // calling the overloaded methods
    int res1 = number.addition(444, 555);
    System.out.println("Addition of two integers: " + res1);

    int res2 = number.addition(333, 444, 555);
    System.out.println("Addition of three integers: " + res2);

    double res3 = number.addition(10.15, 20.22);
    System.out.println("Addition of two doubles: " + res3);
}
```

// Output

Addition of two integers: 999

Addition of three integers: 1332

Addition of two doubles: 30.369999999999997

Example 2: Run Time Polymorphism

// Java Example: Run Time Polymorphism

```
class Vehicle {  
    public void displayInfo() {  
        System.out.println("Some vehicles are there.");  
    }  
}  
  
class Car extends Vehicle {  
    // Method overriding  
    public void displayInfo() {  
        System.out.println("I have a Car.");  
    }  
}  
  
class Bike extends Vehicle {  
    // Method overriding  
    public void displayInfo() {  
        System.out.println("I have a Bike.");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Vehicle v1 = new Car(); // Upcasting  
        Vehicle v2 = new Bike(); // Upcasting  
  
        // Calling the overridden displayInfo() method of Car class  
        v1.displayInfo();  
  
        // Calling the overridden displayInfo() method of Bike class  
        v2.displayInfo();  
    }  
}
```

// Output
I have a Car.
I have a Bike.

Example

```
class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}

class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
    public void bark() {
        System.out.println("Dogs can bark");
    }
}

public class TestDog {

    public static void main(String args[]) {
        Animal a = new Animal();    // Animal reference and object
        Animal b = new Dog();       // Animal reference but Dog object

        a.move();    // runs the method in Animal class
        b.move();    // runs the method in Dog class
        b.bark();
    }
}
```

// Output

TestDog.java:26: error: cannot find symbol

b.bark();

^

symbol: method bark()

location: variable b of type Animal

1 error

final Keyword

The final keyword in java is used to restrict the user. The java final keyword can be used in many context.

- ❑ Variable

- ❑ Method

- ❑ Class

Example 1 (variable)

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

```
class Bike9{
    final int speedlimit=90;//final variable
    void run(){
        speedlimit=400;
    }
    public static void main(String args[]){
        Bike9 obj=new Bike9();
        obj.run();
    }
} //end of class
```

// Output
Compile Time Error

Example 2 (Method)

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

```
class Bike{
    final void run(){System.out.println("running");}
}

class Honda extends Bike{
    void run(){System.out.println("running safely with 100kmph");}

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

Note Please: final method is inherited but you cannot override it.

// Output
Compile Time Error

Example 3 (Class)

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

```
final class Bike{}

class Honda1 extends Bike{
    void run(){System.out.println("running safely with 100kmph");}

    public static void main(String args[]){
        Honda1 honda= new Honda1();
        honda.run();
    }
}
```

// Output
Compile Time Error

Method Overriding Rules

- ❑ The argument list should be exactly the same as that of the overridden method.
- ❑ The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- ❑ The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overriding method in the sub class cannot be either private or protected.
- ❑ Instance methods can be overridden only if they are inherited by the subclass.
- ❑ A method declared final cannot be overridden.
- ❑ A method declared static cannot be overridden but can be re-declared.
- ❑ If a method cannot be inherited, then it cannot be overridden.
- ❑ A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- ❑ A subclass in a different package can only override the non-final methods declared public or protected.
- ❑ Constructors cannot be overridden.

Thanks

References:

<https://www.w3schools.com> & <https://www.tutorialspoint.com> & <https://www.javatpoint.com>