PROGRAMMING

Lecture 6

Sushil Paudel

PREVIOUS TOPIC

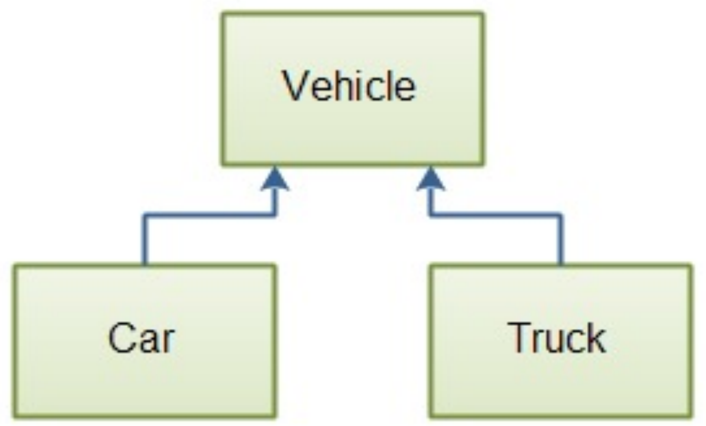
- Scanner – User Input
- Constructor
- Overloading
- Variable

TODAY’S TOPIC

- Inheritance
- Type of inheritance
- Method overriding

INHERITANCE

- Inheritance is an important pillar of OOP(Object Oriented Programming).
- It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.



IMPORTANT TERMINOLOGY

- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

SYNTAX

The keywords used in inheritance is **extends** and **super**.

```
public class Child extends Parent{


}
```

Extend Keyword:

- The extends keyword extends a class (indicates that a class is inherited from another class).

Super Keyword:

- The **super** keyword in java is a reference variable that is used to refer parent class objects.

INHERITANCE

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.

PARENT CLASS

```
public class Vehicle {

    protected String brand = "Ford";

    public void horn() {

        System.out.println("Hornnnnnnn……");
    }
}
```

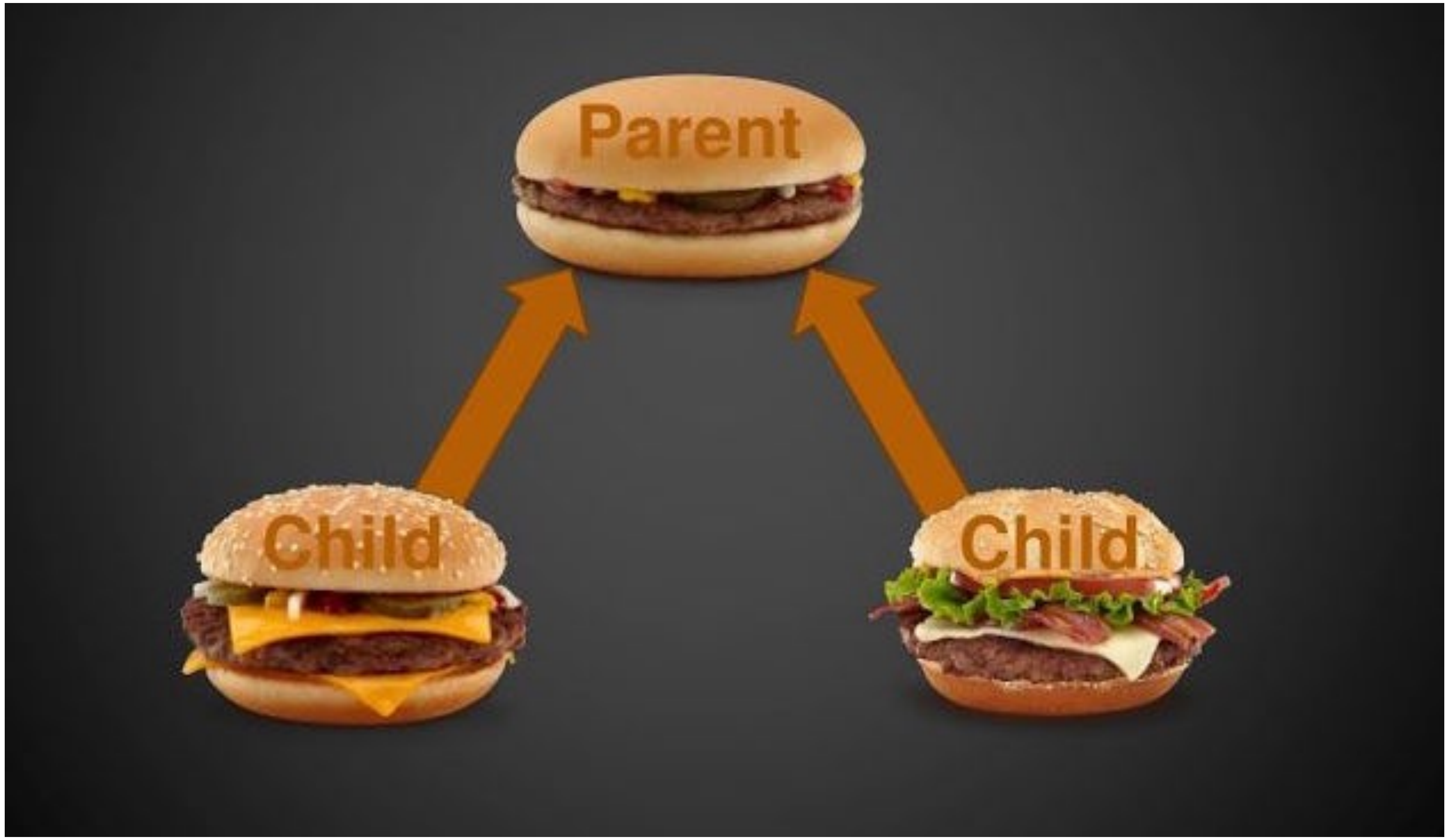
CHILD PARENT

```
public class Car extends Vehicle {

    private String modelName = ”Model S";    // Car attribute

    public static void main(String[] args) {
        // Create a myCar object
        Car myCar = new Car();

        // Call the honk() method (from the Vehicle class) on the myCar object
        myCar.horn();

        // 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);
    }
}
```

OUTPUT

```
Hornnnnnnn……

Tesla Model S
```

PARENT - CHILD

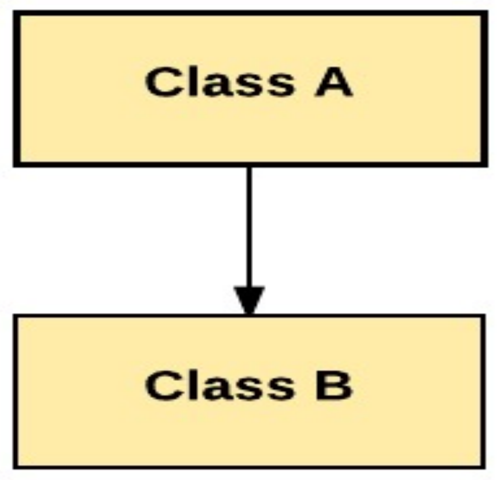


TYPES OF INHERITANCE

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

SINGLE INHERITANCE

In Single Inheritance one class extends another class (one class only).



In above diagram, Class B extends only Class A. Class A is a super class and Class B is a Sub-class.

SINGLE INHERITANCE

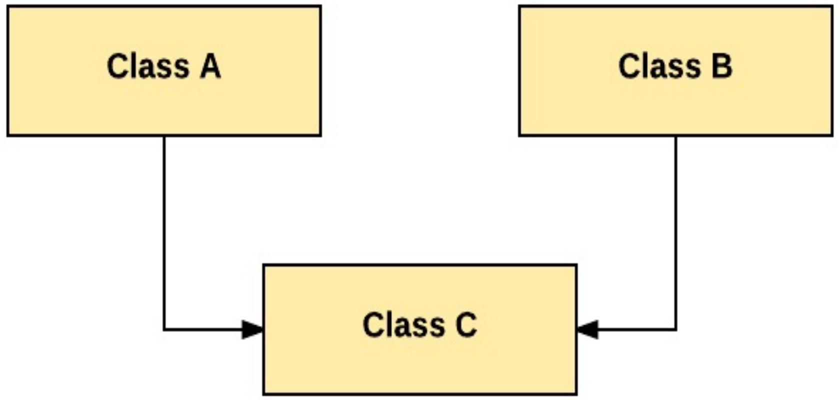
```
class Animal {
    public void eat() {
        System.out.println("eating...");
    }
}

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

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

OUTPUT

```
barking...

eating...
```

MULTIPLE INHERITANCE

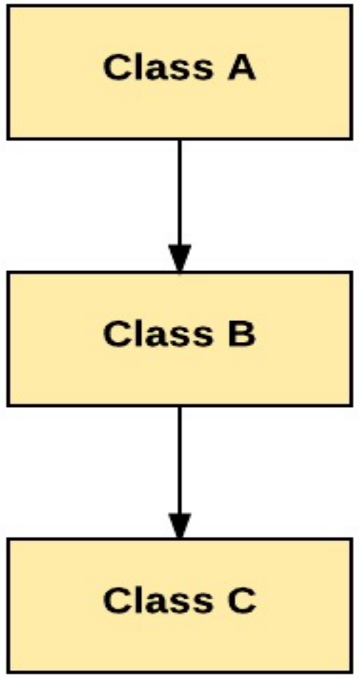
In Multiple Inheritance, one class extending more than one class. Java does not support multiple inheritance. However, it can be achieved through interface.



As per above diagram, Class C extends Class A and Class B both.

MULTILEVEL INHERITANCE

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.



As per shown in diagram Class C is subclass of B and B is a of subclass Class A.

MULTILEVEL INHERITANCE

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

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

class BabyDog extends Dog {
    public void sleep() {
        System.out.println("sleeping...");
    }
}
```

MULTILEVEL INHERITANCE

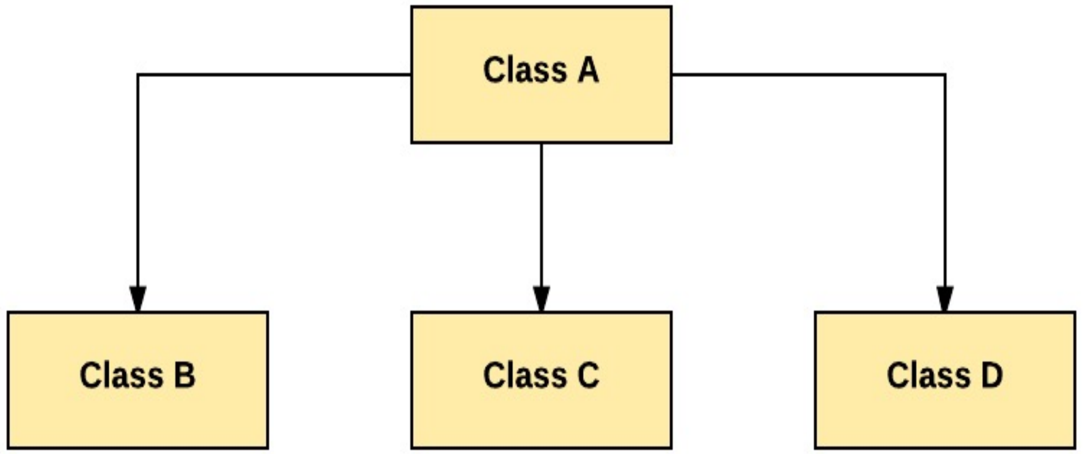
```
class TestInheritance2 {
    public static void main(String args[]) {
        BabyDog babyDog = new BabyDog();
        babyDog.sleep();
        babyDog.bark();
        babyDog.eat();
    }
}
```

OUTPUT

```
sleeping...
barking...
eating...
```

HIERARCHICAL INHERITANCE

In Hierarchical Inheritance, one class is inherited by many sub classes.



As per above example, Class B, C, and D inherit the same class A.

HIERARCHICAL INHERITANCE

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

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

class Cat extends Animal {
    public void meow() {
        System.out.println("meowing...");
    }
}
```

HIERARCHICAL INHERITANCE

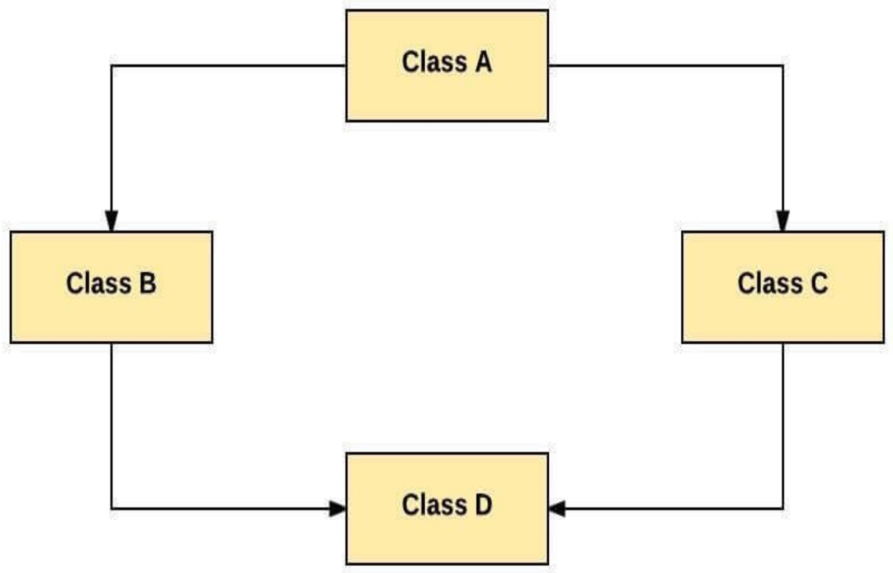
```
class TestInheritance3 {
    public static void main(String args[]) {
        Cat cat = new Cat();
        cat.meow();
        cat.eat();
        //c.bark();
    }
}
```

OUTPUT

```
meowing...
eating...
```

HYBRID INHERITANCE

Hybrid inheritance is a combination of Single and Multiple inheritance. Again Hybrid inheritance is also not directly supported in Java only through interface we can achieve this.

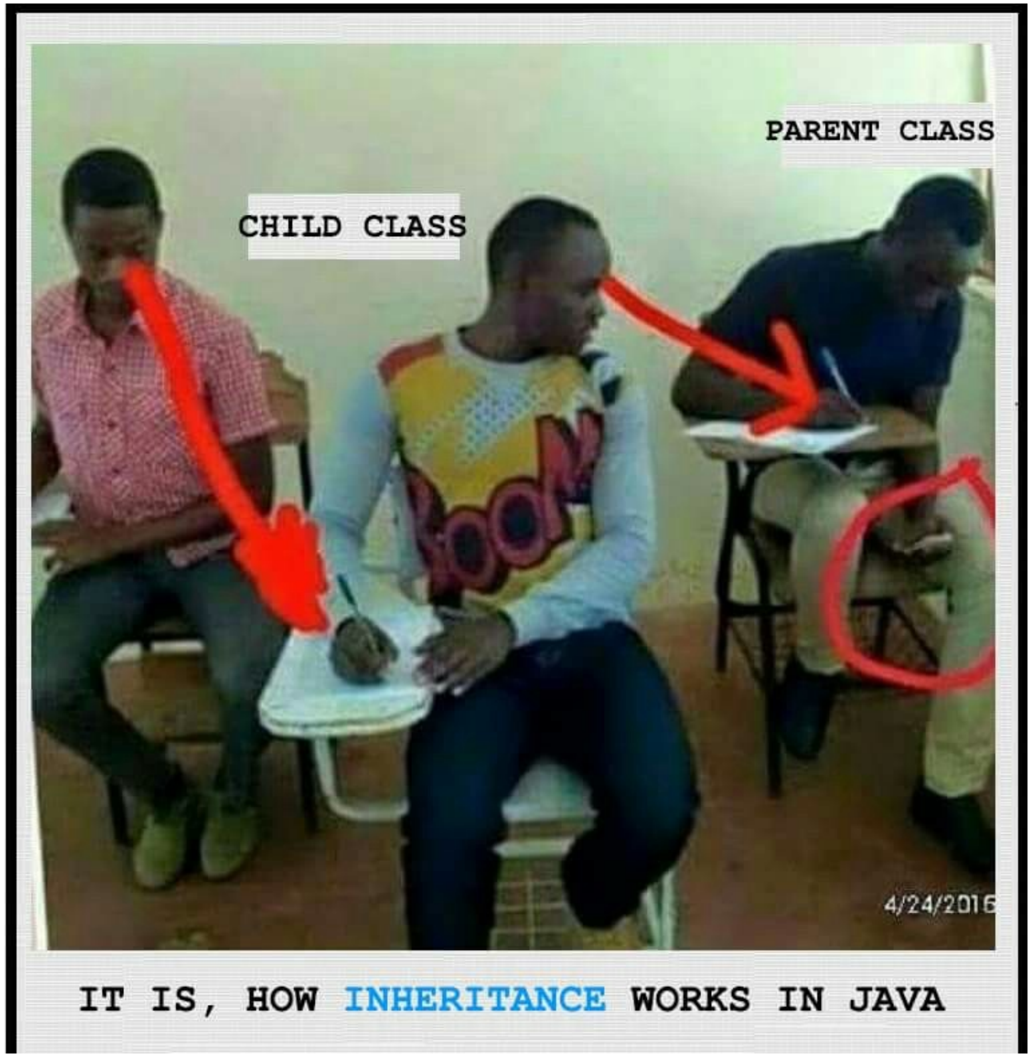


As per above example, all the public and protected members of Class A are inherited into Class D, first via Class B and secondly via Class C.

IMPORTANT POINTS

- In Java, when an "Is-A" relationship exists between two classes we use Inheritance
- The parent class is termed super class and the inherited class is the sub class
- The keyword "extend" is used by the sub class to inherit the features of super class
- Inheritance is important since it leads to reusability of code

INHERITANCE



METHOD OVERRIDING

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

METHOD OVERRIDING

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

EXAMPLE (WITHOUT OVERRIDING)

```
public class Vehicle{
    void run(){
        System.out.println("Vehicle is running");
    }
}
```

```
public class Bike extends Vehicle {
    public static void main(String args[]) {
        // creating an instance of child class
        Bike obj = new Bike();
        // calling the method with child class instance
        obj.run();
    }
}
```

OUTPUT

```
Vehicle is running
```

EXAMPLE (WITH OVERRIDING)

```java
public class Vehicle{
    void run(){
        System.out.println("Vehicle is running");
    }
}
```

```java
public class Bike extends Vehicle {
    // defining the same method as in the parent class
    void run() {
        System.out.println("Bike is running");
    }

    public static void main(String args[]) {
        Bike obj = new Bike();// creating object
        obj.run();// calling method
    }
}
```

OUTPUT

```
Bike is running
```

CALLING PARENT CLASS METHOD

```
class ParentClass{
    //Parent class constructor
    public ParentClass(){
        System.out.println("Constructor of Parent");
    }
    public void display(){
        System.out.println("Parent Method");
    }
}
```

CALLING PARENT CLASS METHOD

```
class JavaExample extends ParentClass{
    public JavaExample(){
        System.out.println("Constructor of Child");
    }
    public void dispay(){
        System.out.println("Child Method");
        //Calling the disp() method of parent class
        super.dispay();
    }
    public static void main(String args[]){
        //Creating the object of child class
        JavaExample obj = new JavaExample();
        obj.display();
    }
}
```

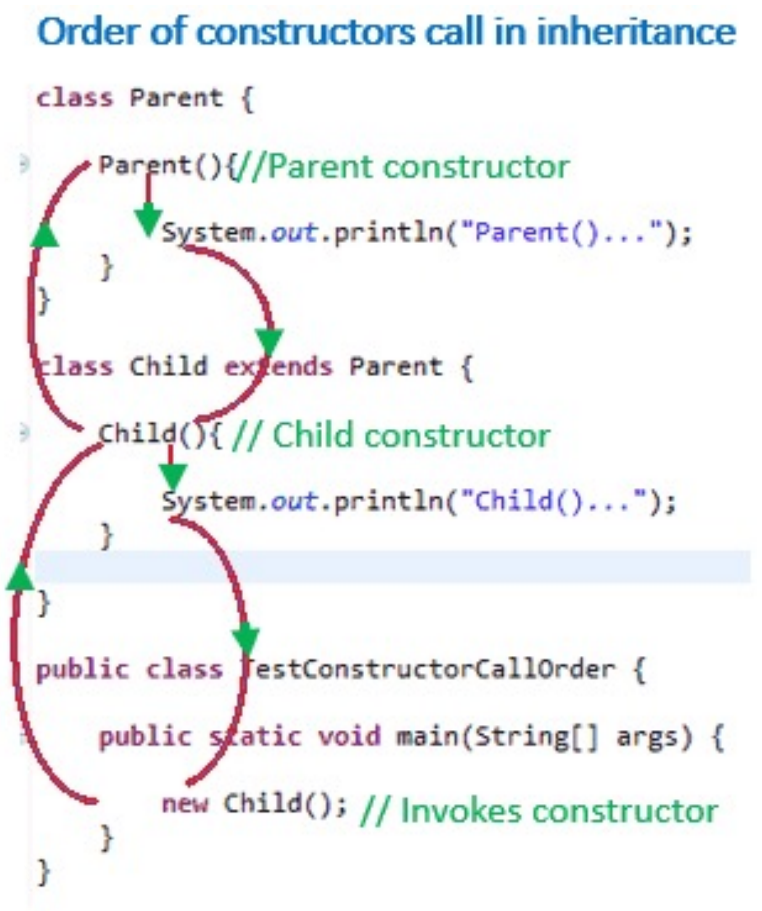
OUTPUT

```
Constructor of Parent

Constructor of Child

Child Method

Parent Method
```

CONSTRUCTOR CALL ORDER

- You create an object of child class in main() as “new Child()”
- Control goes to child constructor, but, its body is not getting executed.
- Control goes to parent constructor, body of it get executed.
- Then, control comes back to child constructor and body get executed.
- Then, the controls come back to “new Child()” statement and exit.

CONSTRUCTOR CALL ORDER



Order of constructors call in inheritance

```
class Parent {
    Parent(){//Parent constructor
        System.out.println("Parent()...");
    }
}
class Child extends Parent {
    Child(){ // Child constructor
        System.out.println("Child()...");
    }

}
public class TestConstructorCallOrder {
    public static void main(String[] args) {
        new Child(); // Invokes constructor
    }
}
```

CALLING PARENT CLASS METHOD

```
class Parent {


    //Parent class constructor
    public Parent(String message) {
        System.out.println("Constructor of Parent: " + message);
    }

    public void display() {
        System.out.println("Parent Method");
    }
}
```

CALLING PARENT CONSTUCTOR

```java
public class Child extends Parent {
    public Child() {
        super(”Hi Parent!");
        System.out.println("Constructor of Child");
    }

    public void display() {
        System.out.println("Child Method");
        //Calling the display() method of parent class
        super.display();
    }

    public static void main(String args[]) {
        //Creating the object of child class

        Child obj = new Child();
        obj.display();
    }
}
```

OUTPUT

```
Constructor of Parent: Hi parent!

Constructor of Child

Child Method

Parent Method
```

THANK YOU!

Any questions?