# This, Super and Constructor Chaining

## this Keyword in Java

'this' keyword in Java is a reference variable that refers to the **current object**. It is primarily used to avoid naming conflicts and to invoke other constructors or methods of the current class.

**Uses of this:**

1. To refer to current class instance variables.
2. To invoke current class methods.
3. To invoke current class constructors (this()).
4. To pass the current object as an argument.

### Referring to Instance Variables

```
class Student {
   int id;
   String name;

   Student(int id, String name) {
      this.id = id; // refers to instance variable
      this.name = name;
   }

   void display() {
      System.out.println(id + " " + name);
   }

   public static void main(String[] args) {
      Student s = new Student(1, "John");
      s.display();
   }
}
```

**Output:**

```
1 John
```

## 2. Invoking Current Class Method

```
class Example {
    void show() {
        System.out.println("Show method");
    }

    void display() {
        this.show();
    }

    public static void main(String[] args) {
        Example e = new Example();
        e.display();
    }
}
```

**Output:**

```
Show method
```

## 3. Invoking Constructor with this()

```
class Person {
    Person() {
        this("Default Name");
        System.out.println("No-arg constructor");
    }

    Person(String name) {
        System.out.println("Parameterized constructor: " + name);
    }

    public static void main(String[] args) {
        new Person();
    }
}
```

### Output:

```
Parameterized constructor: Default Name
No-arg constructor
```

## super Keyword in Java

The super keyword refers to the **immediate parent class** object.

**Uses of super:**

1. Access parent class variables.
2. Invoke parent class methods.
3. Invoke parent class constructors.

### Accessing Parent Class Variables

```
class Animal {
   String category = "Animal";
}

class Dog extends Animal {
   String category = "Dog";

   public void printCategory() {
      System.out.println("Child Category: " + category);
      System.out.println("Parent Category: " + super.category);
   }

   public static void main(String[] args) {
      Dog d = new Dog();
      d.printCategory();
   }
}
```

### Output:

```
Child Category: Dog
Parent Category: Animal
```

## Accessing Parent Class Methods

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

class Dog extends Animal {
   void sound() {
      super.sound();
      System.out.println("Dog barks");
   }

   public static void main(String[] args) {
      Dog d = new Dog();
      d.sound();
   }
}
```

## Output:

```
Animal makes a sound
Dog barks
```

## Calling Parent Class Constructor

```
class Animal {
   Animal() {
      System.out.println("Animal constructor called");
   }
}

class Dog extends Animal {
   Dog() {
      super();
      System.out.println("Dog constructor called");
   }
```

```
 public static void main(String[] args) {
       Dog d = new Dog();
   }
}
```

**Output:**

```
Animal constructor called
Dog constructor called
```

## Constructor Chaining in Java

Constructor chaining is the process of calling one constructor from another constructor using this() (same class) or super() (parent class).

🔑 **Key Points:**

- this() calls another constructor in the same class.
- super() calls a constructor from the superclass.
- Must be the **first statement** in the constructor.
- Cannot use both this() and super() in the same constructor.

**1. Using this()**

```
class Person {
   Person() {
      this("Unknown");
      System.out.println("Default constructor called");
   }

   Person(String name) {
      System.out.println("Parameterized constructor called with name: " +
name);
   }

   public static void main(String[] args) {
      Person p = new Person();
   }
}
```

## Output:

```
Parameterized constructor called with name: Unknown
Default constructor called
```

### 2. Using super()

```
class Parent {
    Parent() {
        System.out.println("Parent constructor called");
    }
}

class Child extends Parent {
    Child() {
        super();
        System.out.println("Child constructor called");
    }

    public static void main(String[] args) {
        Child c = new Child();
    }
}
```

## Output:

```
Parent constructor called
Child constructor called
```

### 3. Combined Example

```
        class Vehicle {
    Vehicle(String name) {
        System.out.println("Vehicle constructor: " + name);
    }
}

class Car extends Vehicle {
    Car() {
        this("Honda");
```

```
        System.out.println("Default Car constructor");
    }

    Car(String brand) {
        super("Transport");
        System.out.println("Car constructor: " + brand);
    }

    public static void main(String[] args) {
        Car c = new Car();
    }
}
```

## Output:

```
Vehicle constructor: Transport
Car constructor: Honda
Default Car constructor
```