Constructors in Inheritance & Uses of super keyword in Java

1. Constructors in Inheritance

✅ **Key Points:**

- When a subclass (child) inherits from a superclass (parent), **constructors are *not* inherited**.
- However, **superclass constructors are called** when the subclass object is created.
- The **superclass constructor runs first**, then the subclass constructor.

Default Constructor Behaviour:

If no constructor is defined, Java automatically calls the **default constructor** of the superclass.

📌 **Example 1: Default Constructors**

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

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

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

🟢 **Output:**

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

Parameterized Constructor Behavior:

If the superclass has a **parameterized constructor**, you must explicitly call it using super().

📌 **Example 2: Parameterized Constructor**

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

class Child extends Parent {
    Child() {
        super("Ishan");  // Must call super with argument
        System.out.println("Child constructor called");
    }
}

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

Output:

```
Parent constructor called: Ishan
Child constructor called
```

2. Uses of super keyword

The super keyword in Java is used to refer to the **immediate parent class**. It has **three main uses**:

✅ **a. Call parent class constructor**

super(); // Calls parent default constructor

super(args); // Calls parent parameterized constructor

✅ **b. Access parent class method**

If a method is overridden, you can call the parent's version using super.methodName().

```
class Parent {
    void show() {
        System.out.println("Parent show");
    }
}

class Child extends Parent {
    void show() {
        super.show();  // Call parent method
        System.out.println("Child show");
    }
}
```

✅ **c. Access parent class variables**

When child class has a variable with the same name, super.variableName accesses the parent's version.

```
class Parent {
    int x = 10;
}
```

```
class Child extends Parent {
    int x = 20;

    void display() {
        System.out.println("Child x = " + x);
        System.out.println("Parent x = " + super.x);
    }
}
```

Summary Table

Use of super	Purpose
super();	Calls parent constructor
super.method();	Calls parent method
super.variable;	Accesses parent variable

📘 Access Modifiers in Java

🔹 What are Access Modifiers?

Access Modifiers in Java define the **visibility/scope** of classes, variables, constructors, and methods.
There are **four** main access levels:

Modifier	Same Class	Same Package	Subclass (other pkg)	Other Packages
public	✅	✅	✅	✅
protected	✅	✅	✅	❌
(default)	✅	✅	❌	❌
private	✅	❌	❌	❌

1. public

- Accessible **from anywhere** (any class, package).
- Common for APIs or methods meant to be used externally.

📌 **Example:**

```
public class A {
    public int x = 10;

    public void show() {
        System.out.println("Public show");
    }
}
class B {
    public static void main(String[] args) {
        A obj = new A();
        obj.show(); // ✅ Accessible
```

```
}}
```

2. private

- Accessible **only within the same class**.
- Good for encapsulation and hiding internal details.

📌 **Example:**

```
public class A {
    private int x = 100;

    private void display() {
        System.out.println("Private display");
    }

    public void accessPrivate() {
        System.out.println("x = " + x);
        display(); // ✅ Access within same class
    }
}
class B {
    public static void main(String[] args) {
        A obj = new A();
        // obj.display(); // ❌ Not allowed
        obj.accessPrivate(); // ✅ Allowed through public method
    }
}
```

3. protected

- Accessible in:
    - Same class
    - Same package
    - Subclasses (even in different packages)

📌 **Example:**

```
package pack1;


public class A {
   protected void show() {
      System.out.println("Protected show");
   }
}
package pack2;


import pack1.A;


class B extends A {
   public void display() {
      show(); // ✅ Allowed in subclass
   }
}
```

4. Default (No Modifier)

- Also called **package-private**.
- Accessible **only within the same package**.
- Not visible to subclasses in other packages.

📌 **Example:**

```
class A {
    void show() {
        System.out.println("Default show");
    }
}
class B {
    public static void main(String[] args) {
        A obj = new A();
        obj.show(); // ✅ Allowed (same package)
    }
}
```

🔚 **Summary Table**

Modifier	Visibility Scope
public	Everywhere
private	Only within the declared class
protected	Same package + subclasses
default	Same package only

💡 **Best Practices:**

- Use private for variables and helper methods.
- Use public only when necessary (e.g., API methods).
- Use protected for inheritance-friendly design.
- Default is okay for package-level utilities.