INTRODUCTION TO PROGRAMMING LANGUAGE II(JAVA)

Fariha Zahin
Lecturer
CSE, Southeast University

super keyword:

It is used to refer to the immediate **parent class** object.

•Helps access:

- •Parent class constructor
- •Parent class methods
- •Parent class data members

Using super to Access Parent Class Variables

In this code, the Child class has a variable x that hides the x variable from its Parent class. Inside the display method of Child, super.x is used to access the x from the parent class (value 10), rather than the one in the child class (value 20). This shows how super can resolve variable name conflicts in inheritance.

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

class Child extends Parent {
    int x = 20;

    void display() {
        System.out.println(super.x); // Outputs 10
    }
}
```

Using super to Call Parent Class Method

The Child class overrides the greet method of the Parent class. Inside the child's version of greet, super.greet is used to call the original method from the parent before printing its own message. This demonstrates how to invoke a superclass method when it is overridden in the subclass.

```
class Parent {
    void greet() {
        System.out.println("Hello from Parent");
    }
}

class Child extends Parent {
    @Override
    void greet() {
        super.greet();
        System.out.println("Hello from Child");
    }
}
```

Using super to Call Parent Constructor

super() must be the first statement in a constructor if used.

Here, the Child class constructor explicitly calls the no-argument constructor of the Parent class using super. This ensures the parent constructor runs first before executing the child constructor's code. It illustrates how constructor chaining works between parent and child classes in Java.

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

class Child extends Parent {
    Child() {
        super(); // Optional if no-arg constructor
        System.out.println("Child constructor called");
    }
}
```

In this example, the Student class extends Person, and both have constructors that accept parameters. The Student constructor uses super to call the Person constructor and initialize the inherited name field. The main method creates a Student object, showing how both constructors are executed and how values are initialized through inheritance.

```
class Person {
    String name;
    Person(String name) {
        this.name = name;
        System.out.println("Person constructor called");
    }
}

class Student extends Person {
    int id;
    Student(String name, int id) {
        super(name);
        this.id = id;
        System.out.println("Student constructor called");
    }
```

super vs this

super → refers to the immediate parent class object

this → refers to the current object

They can be used together when necessary.

```
20
20
10
```

Main.java

```java
class Parent {  1 usage  1 inheritor
    int x = 10;  1 usage
}

class Child extends Parent {  2 usages
    int x = 20;  2 usages

    void show() {  1 usage
        System.out.println(x);        // 20  (Child's x)
        System.out.println(this.x);   // 20  (explicitly current object's x)
        System.out.println(super.x);  // 10  (Parent's x)
    }
}

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

Constructor Chaining

When constructors call other constructors (in the same class or in the parent class) to avoid code duplication and ensure proper initialization.

Two ways:
1. this(...) → calls another constructor **in the same class**
2. super(...) → calls a constructor **in the immediate parent class**

Main.java

```java
class Person {  1 usage  1 inheritor
    String name;  1 usage
    int age;  1 usage

    // Constructor 1 (no-arg)
    Person() {  1 usage
        this( name: "Unknown",  age: 0);                    // chaining inside sa
        System.out.println("Person() called");
    }

    // Constructor 2 (one-arg)
    Person(String name) {  no usages
        this(name,  age: 0);                              // chaining inside same cla
        System.out.println("Person(String) called");
    }

    // Constructor 3 (two-arg) - the "main" one
    Person(String name, int age) {  3 usages
        this.name = name;                                 // actual initialization here
        this.age = age;
        System.out.println("Person(String, int) called");
    }
}
```

Constructor Chaining

Rules:

this(...) or super(...) must be the **first statement** in a constructor.

Only **one** of them can be used (not both).

If you write nothing → compiler automatically inserts super() (no-arg parent constructor).

```
Person(String, int) called
Person() called
Student() called

Person(String, int) called
Student(full) called
```

```
25  class Student extends Person {  4 usages
26      int roll;  1 usage
27
28      Student() {  1 usage
29          super();                                // chaining to parent (calls Pe
30          System.out.println("Student() called");
31      }
32
33      Student(String name, int age, int roll) {  1 usage
34          super(name, age);                       // chaining to parent's 2-arg c
35          this.roll = roll;
36          System.out.println("Student(full) called");
37      }
38  }
39  public class Main {
40      public static void main(String[] args)
41      {
42          Student s1 = new Student();
43          System.out.println();
44          Student s2 = new Student( name: "Alice",  age: 20,  roll: 101);
45      }
46  }
```

Calling Overloaded Parent Constructors

Main.java

```java
class Person {  1 usage  1 inheritor
    String name;  1 usage
    int age;  1 usage

    Person()  no usages
    { System.out.println("Person()"); }
    Person(String name)  1 usage
    { this.name = name; System.out.println("Person(String)"); }
    Person(String name, int age) {  1 usage
        this(name);
        this.age = age;
        System.out.println("Person(String,int)");
    }
}
class Student extends Person {  2 usages
    Student() {  1 usage
        super( name: "Anonymous", age: 18);   // explicitly choosing the 2-ar
        System.out.println("Student()");
    }
}

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

```
Person(String)
Person(String,int)
Student()
```

super() is inserted automatically (only for no-arg)

Main.java

```java
class A {  1 usage  1 inheritor
    A()  { System.out.println("A()"); }  1 usage
    A(int x) { System.out.println("A(int)"); }  no usages
}
class B extends A {  2 usages
    B() {  1 usage
        // compiler inserts super(); here → calls A()
        System.out.println("B()");
    }
}

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

```
C:\Users\u1904\.jdks\openjdk-25\bin\java
A()
B()
```

super() is inserted automatically (only for no-arg)

```
300
200
Parent
```

Main.java

```java
class GrandParent {  1 usage  2 inheritors
    int x = 100;  no usages
    void show() { System.out.println("Grand"); }  1 usage  1 override
}
class Parent extends GrandParent {  1 usage  1 inheritor
    int x = 200;  1 usage
    void show() { System.out.println("Parent"); }  1 usage
}
class Child extends Parent {  2 usages
    int x = 300;  1 usage
    void display() {  1 usage
        System.out.println(x);        // 300
        System.out.println(super.x);  // 200 ← Parent's x, not GrandParent's
        super.show();                 // "Parent"
    }
}
public class Main {
    public static void main(String[] args)
    {
        Child c =new Child();
        c.display();
    }
}
```