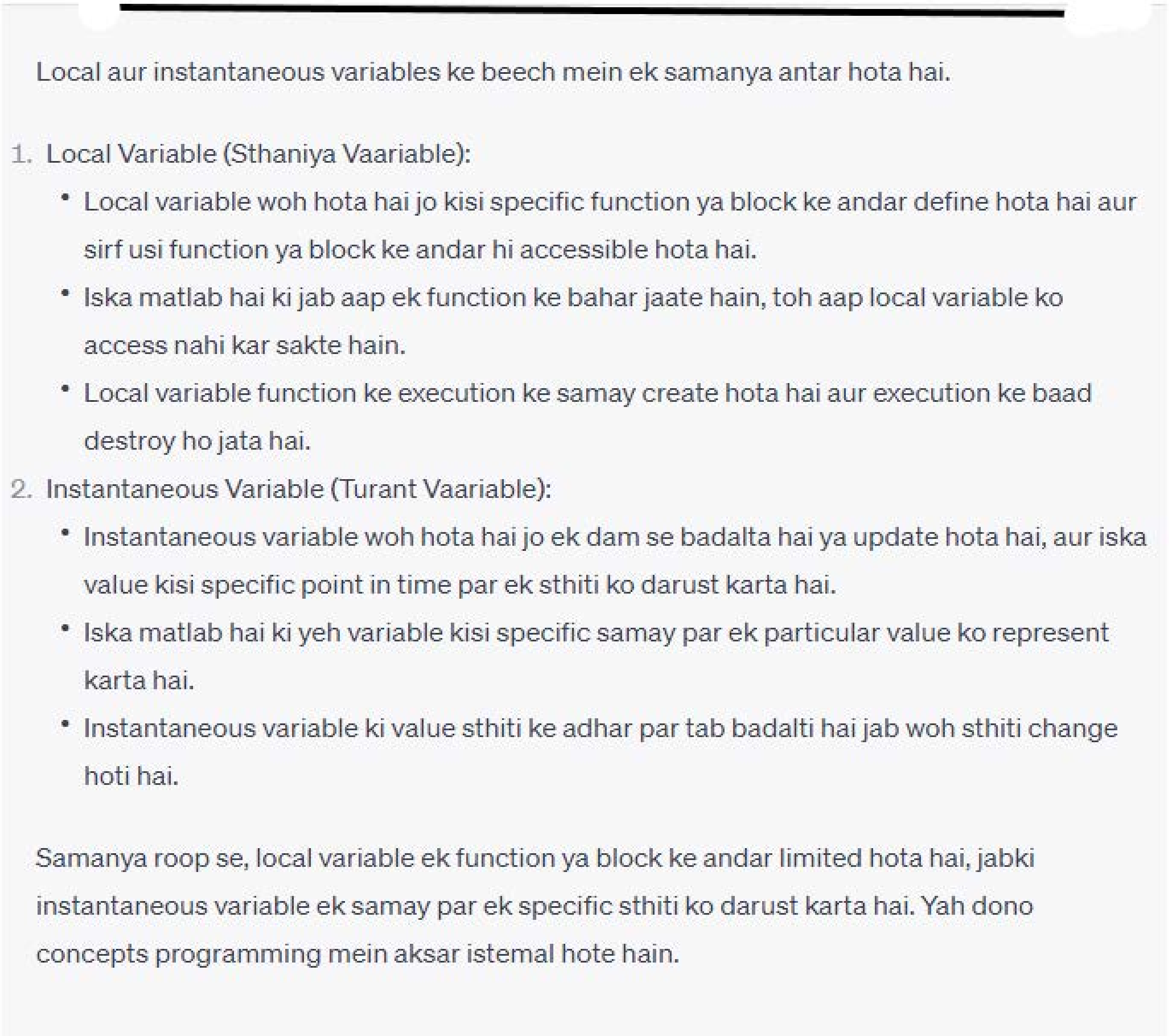
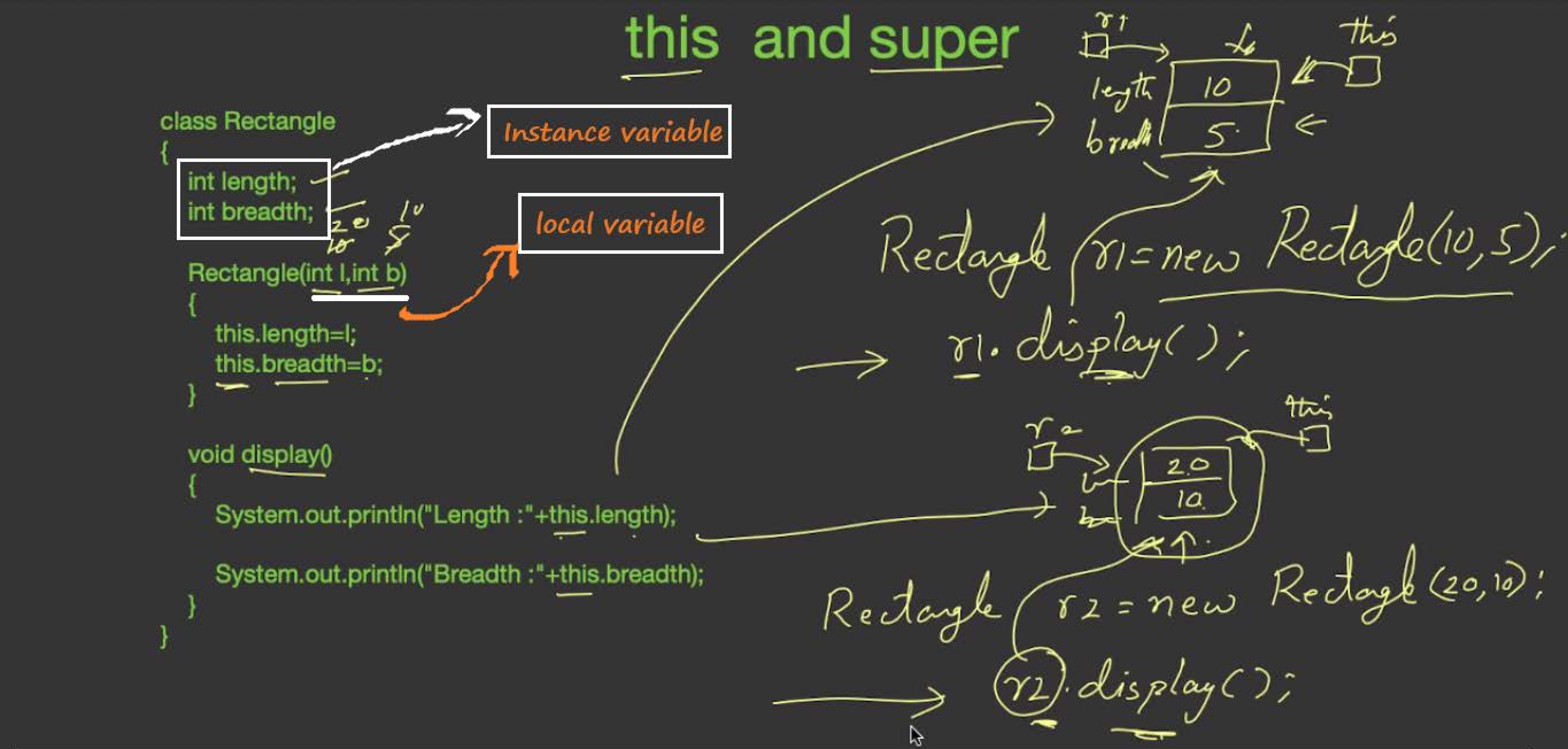
Local & Instantaneous Variable



Local aur instantaneous variables ke beech mein ek samanya antar hota hai.

1. Local Variable (Sthaniya Vaariable):
   - Local variable woh hota hai jo kisi specific function ya block ke andar define hota hai aur sirf usi function ya block ke andar hi accessible hota hai.
   - Iska matlab hai ki jab aap ek function ke bahar jaate hain, toh aap local variable ko access nahi kar sakte hain.
   - Local variable function ke execution ke samay create hota hai aur execution ke baad destroy ho jata hai.
2. Instantaneous Variable (Turant Vaariable):
   - Instantaneous variable woh hota hai jo ek dam se badalta hai ya update hota hai, aur iska value kisi specific point in time par ek sthiti ko darust karta hai.
   - Iska matlab hai ki yeh variable kisi specific samay par ek particular value ko represent karta hai.
   - Instantaneous variable ki value sthiti ke adhar par tab badalti hai jab woh sthiti change hoti hai.

Samanya roop se, local variable ek function ya block ke andar limited hota hai, jabki instantaneous variable ek samay par ek specific sthiti ko darust karta hai. Yah dono concepts programming mein aksar istemal hote hain.



this and super

```java
class Rectangle
{
  int length;
  int breadth;

  Rectangle(int l,int b)
  {
    this.length=l;
    this.breadth=b;
  }

  void display()
  {
    System.out.println("Length :"+this.length);

    System.out.println("Breadth :"+this.breadth);
  }
}
```

```java
Rectangle r1=new Rectangle(10,5);
→ r1.display();

Rectangle r2 = new Rectangle(20,10);
→ r2.display();
```

--> " This " keyword ka use tab kiya jata hai jab ham within same class same name ka variable ko create karta hai and haam instance variable ko refer karna chahta hai.

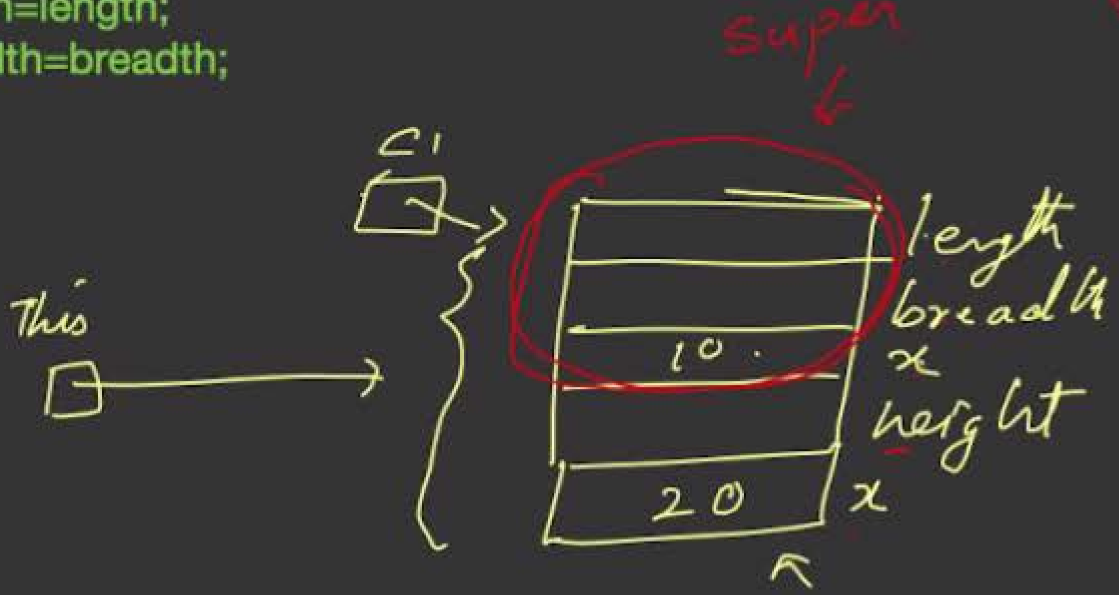
--> Instance class variable ki preference high hoti hai over local variable, iseliya haam isa instance class ka variable koi hi mostly show krana ka liya use karta hai.

--> Define: In Java, " this " keyword is used to refer to the current object inside a method or a constructor.

this and super

```
class Rectangle
{
    int length;
    int breadth;
    int x=10;

    Rectangle(int length,int breadth)
    {
        this.length=length;
        this.breadth=breadth;
    }
}
```

```
class Cuboid extends Rectangle
{
    int height;
    int x=20;

    Cuboid(int l,int b,int h)
    {
        super(l,b);
        height=h;
    }

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



"Super" aur "this" keywords Java mein dono hi important hain aur inka use class ke instance variables aur methods mein confusion ko avoid karne ke liye hota hai. Yahan, main "super" aur "this" keywords ke beech ke mukhya antar ko explain karunga:

1. **Super Keyword:**
   - "super" keyword ka use parent class (superclass) ke members (variables aur methods) ko refer karne ke liye hota hai.
   - Agar child class (subclass) mein ek method ya variable parent class ke method ya variable se same naam se declare hota hai, to "super" keyword ka use us parent class ke member ko access karne ke liye hota hai.
   - Isse aap parent class ke methods ko call kar sakte hain aur parent class ke variables ko access kar sakte hain.

   Example:

   java Copy code

```java
class Parent {
    int num = 10;
    void display() {
        System.out.println("This is the parent class");
    }
}

class Child extends Parent {
    int num = 20;
    void display() {
        System.out.println("This is the child class");
        System.out.println("Value of num in child class: " + num);
```

2. **This Keyword:**
   - "this" keyword current class ke members (variables aur methods) ko refer karne ke liye hota hai.
   - Agar ek method mein local variable aur class ke instance variable ka naam same hota hai, to "this" keyword ka use instance variable ko indicate karne ke liye hota hai.
   - Isse aap current class ke methods ko call kar sakte hain aur current class ke variables ko access kar sakte hain.

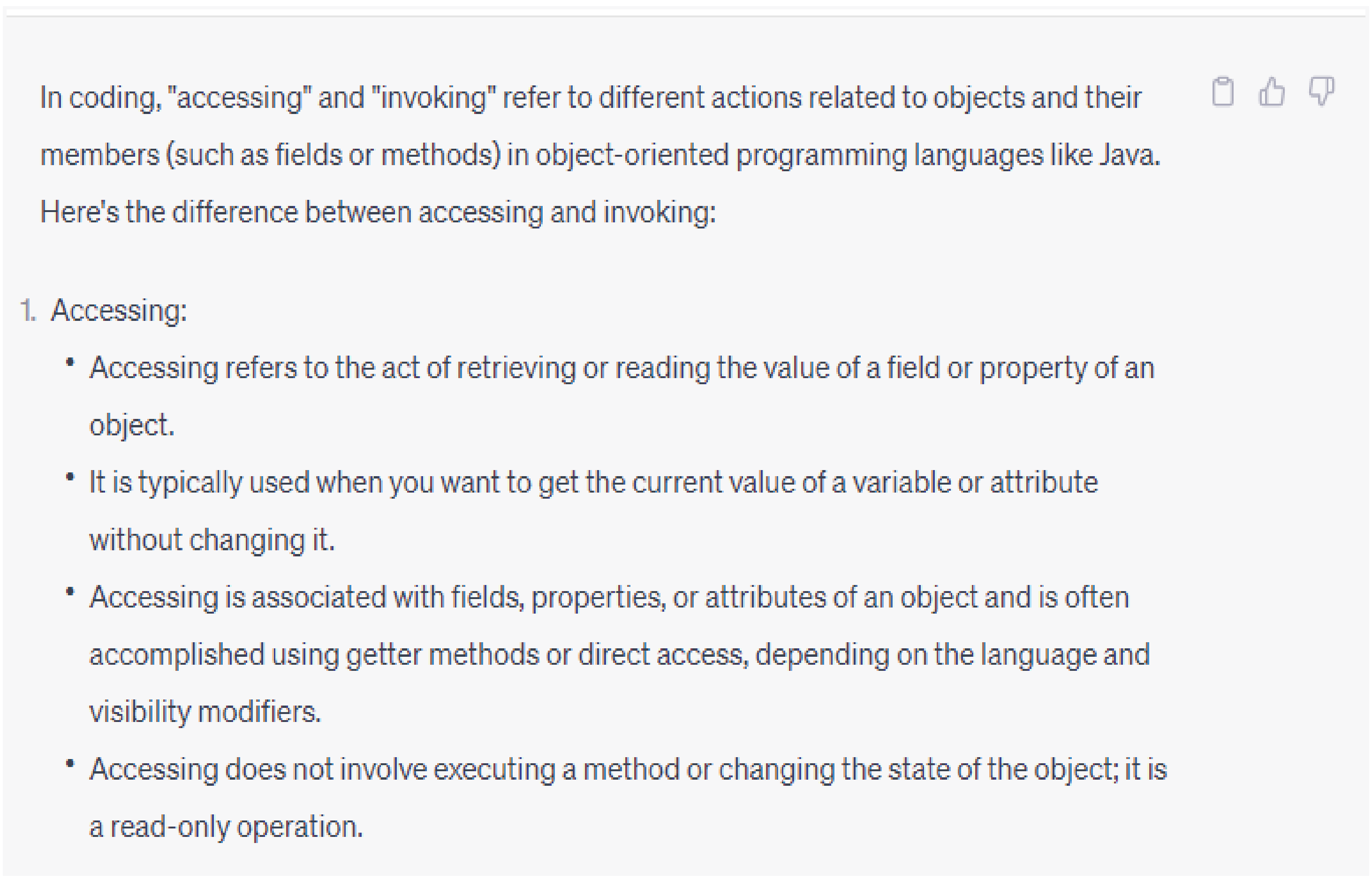
Example:

java Copy code

```java
class MyClass {
    int num = 10;
    void display(int num) {
        System.out.println("Value of num in method parameter: " + num);
        System.out.println("Value of num in instance variable using this ke
    }
}
```

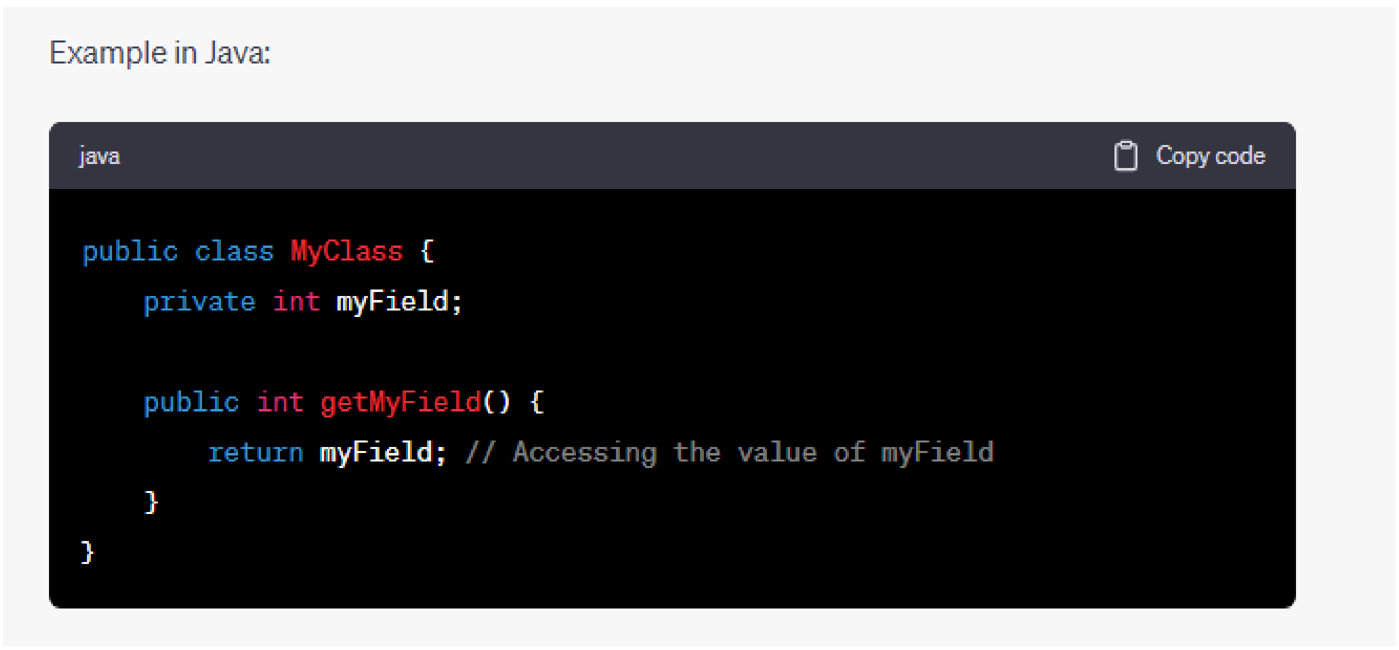
Samanya taur par, "super" parent class ke aur "this" current class ke members ke liye use hota hai taki aap confusion ko avoid kar sakein jab same naam ke variables aur methods hote hain.

→ ***Accessing v/s Invoking***



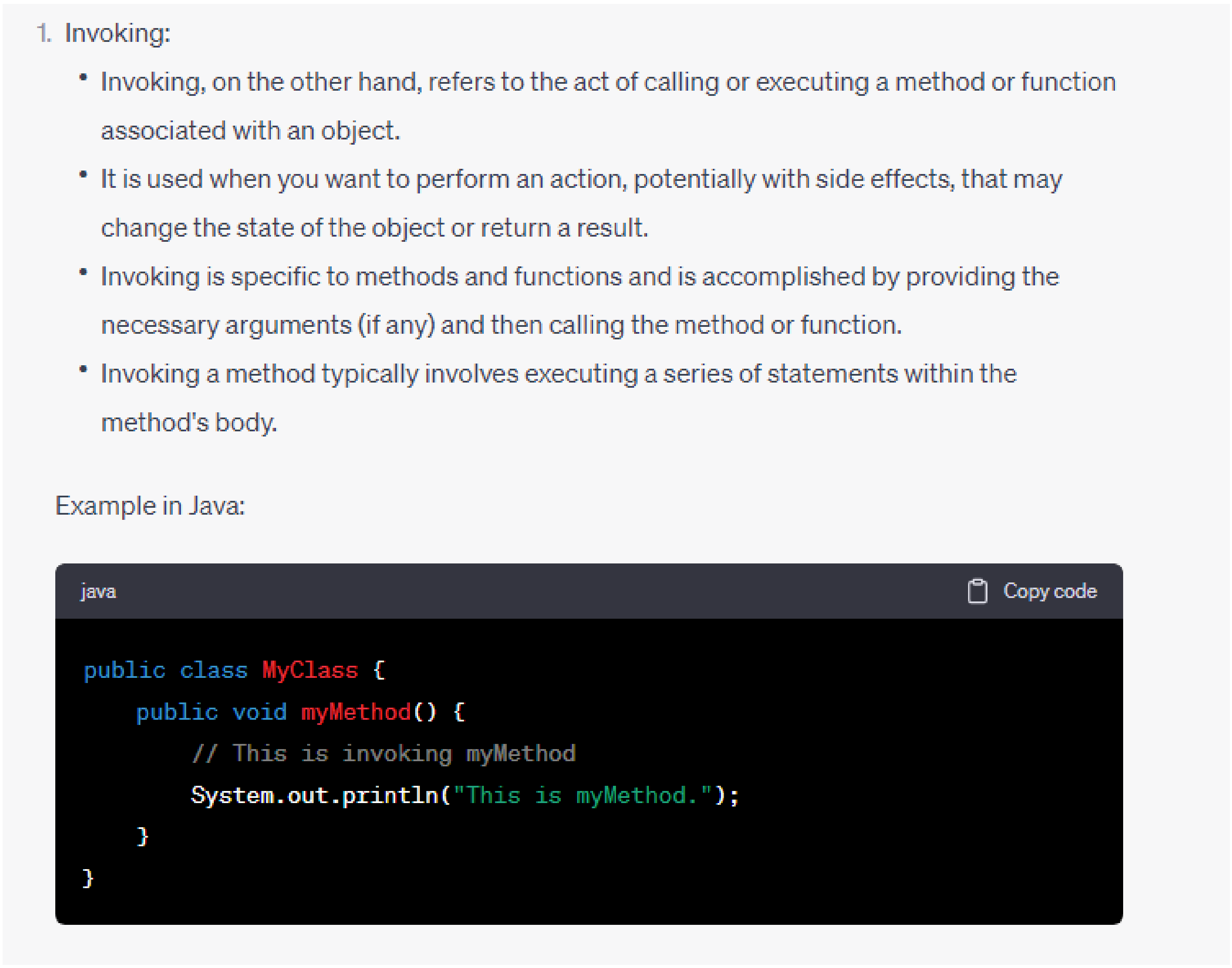
In coding, "accessing" and "invoking" refer to different actions related to objects and their members (such as fields or methods) in object-oriented programming languages like Java. Here's the difference between accessing and invoking:

1. Accessing:
   - Accessing refers to the act of retrieving or reading the value of a field or property of an object.
   - It is typically used when you want to get the current value of a variable or attribute without changing it.
   - Accessing is associated with fields, properties, or attributes of an object and is often accomplished using getter methods or direct access, depending on the language and visibility modifiers.
   - Accessing does not involve executing a method or changing the state of the object; it is a read-only operation.



Example in Java:

```java
public class MyClass {
    private int myField;

    public int getMyField() {
        return myField; // Accessing the value of myField
    }
}
```



1. Invoking:
   - Invoking, on the other hand, refers to the act of calling or executing a method or function associated with an object.
   - It is used when you want to perform an action, potentially with side effects, that may change the state of the object or return a result.
   - Invoking is specific to methods and functions and is accomplished by providing the necessary arguments (if any) and then calling the method or function.
   - Invoking a method typically involves executing a series of statements within the method's body.

   Example in Java:

java Copy code

```java
public class MyClass {
    public void myMethod() {
        // This is invoking myMethod
        System.out.println("This is myMethod.");
    }
}
```

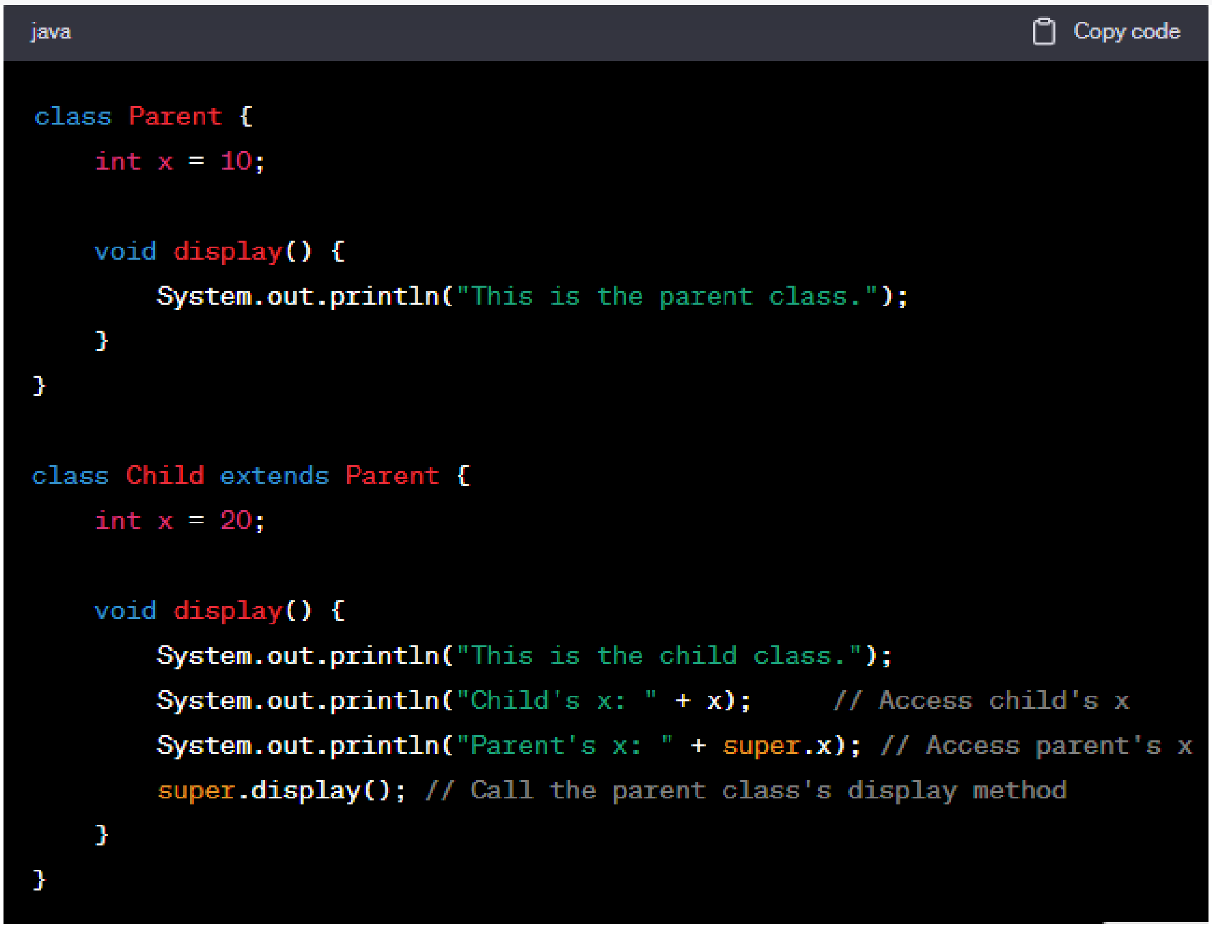
In summary, accessing is about reading values (like fields or properties), while invoking is about executing methods or functions. Both are fundamental concepts in object-oriented programming and understanding the difference between them is essential for writing effective and maintainable code.

→ Super v/s this keyword

*In Java, the "**super**" keyword is used to access members (fields, methods, or constructors) of a superclass (parent class). It is often used within a sub-class (child class) to refer to the members of its immediate superclass. The "**super**" keyword can be used in the following ways:*

1. *Accessing Superclass Members:*

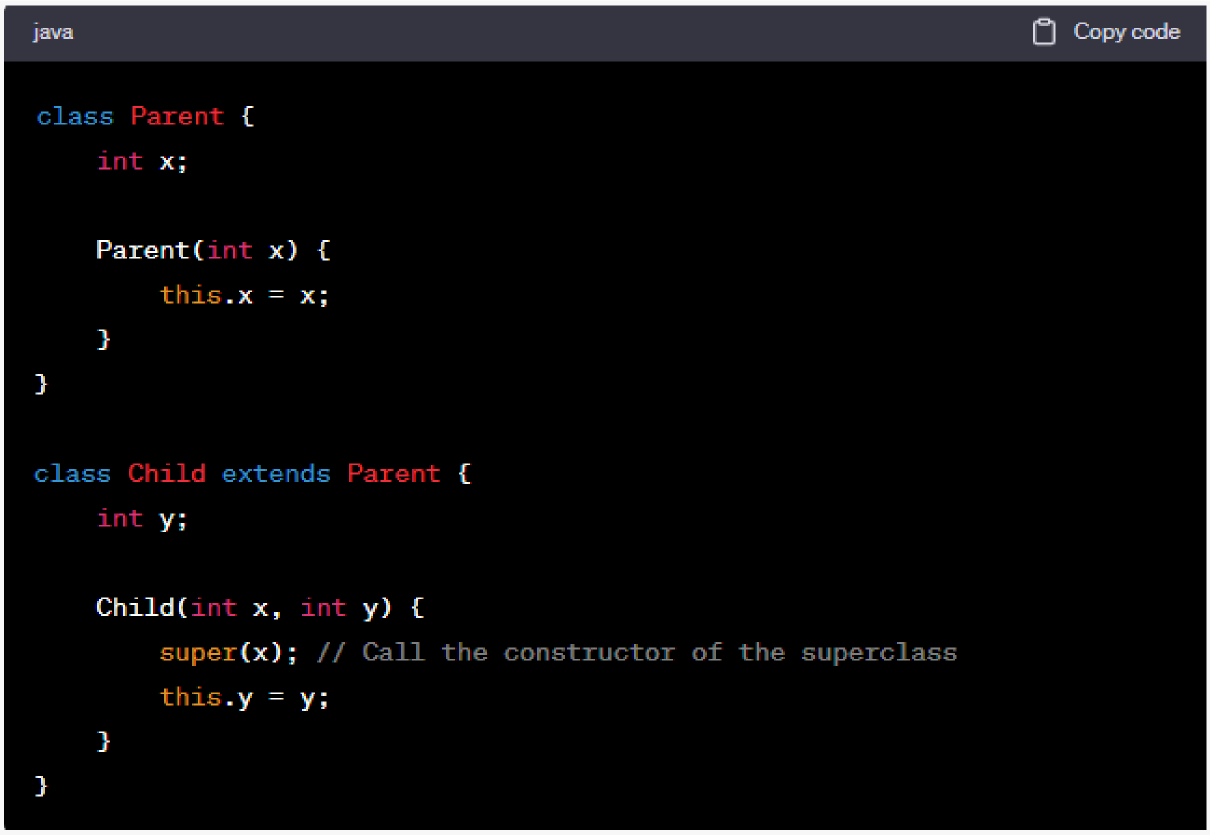
*** You can use "**super**" to access fields and methods of the superclass. This is particularly useful when the subclass has its own members with the same name as those in the superclass, and you want to distinguish between them.*


java Copy code

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

    void display() {
        System.out.println("This is the parent class.");
    }
}

class Child extends Parent {
    int x = 20;

    void display() {
        System.out.println("This is the child class.");
        System.out.println("Child's x: " + x);      // Access child's x
        System.out.println("Parent's x: " + super.x); // Access parent's x
        super.display(); // Call the parent class's display method
    }
}
```

2. *Invoking Superclass Constructors:*

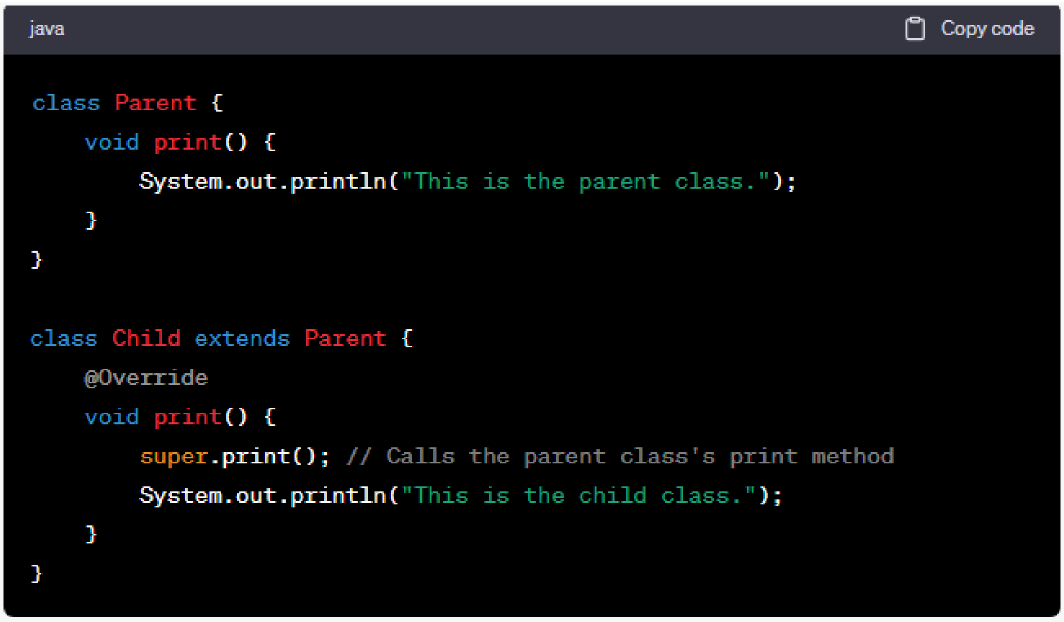
*** You can use super to invoke constructors of the superclass. This is typically done in the constructor of a subclass to initialise the inherited members.*



```java
class Parent {
    int x;

    Parent(int x) {
        this.x = x;
    }
}

class Child extends Parent {
    int y;

    Child(int x, int y) {
        super(x); // Call the constructor of the superclass
        this.y = y;
    }
}
```

3. ***Invoking Superclass Method:***

***You can also use super to call a method from the superclass, even if the subclass overrides that method.*



```java
class Parent {
    void print() {
        System.out.println("This is the parent class.");
    }
}

class Child extends Parent {
    @Override
    void print() {
        super.print(); // Calls the parent class's print method
        System.out.println("This is the child class.");
    }
}
```

*The "**super**" keyword is essential for maintaining a clear distinction between the members of the superclass and those of the subclass, especially when they have the same names. It ensures that you can access and manipulate the superclass's members when needed.*