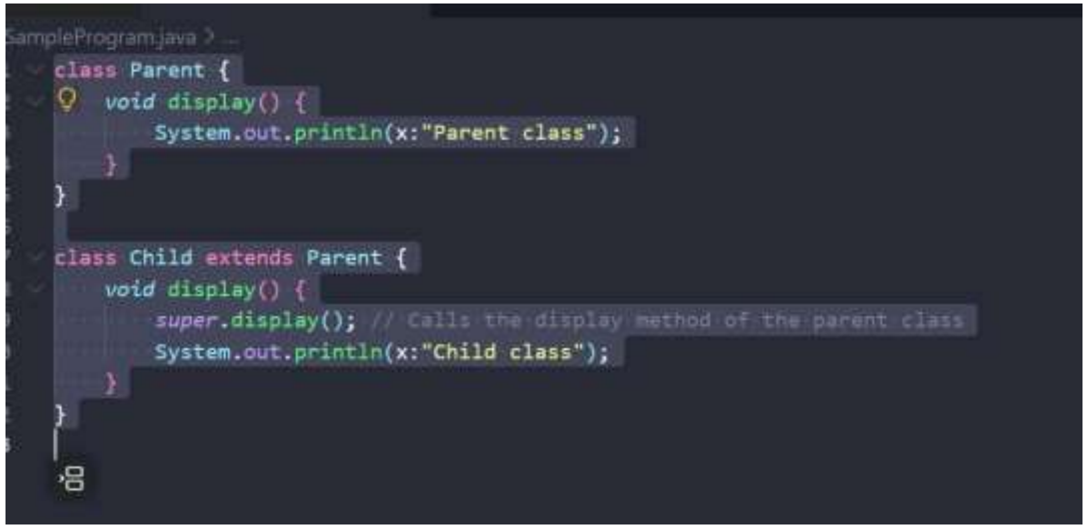
Keywords and constructor

The super Keyword:

In Java, the super keyword is used to access members (fields and methods) of the parent class within a subclass. It's particularly useful when the subclass overrides a method or hides a field of the parent class.

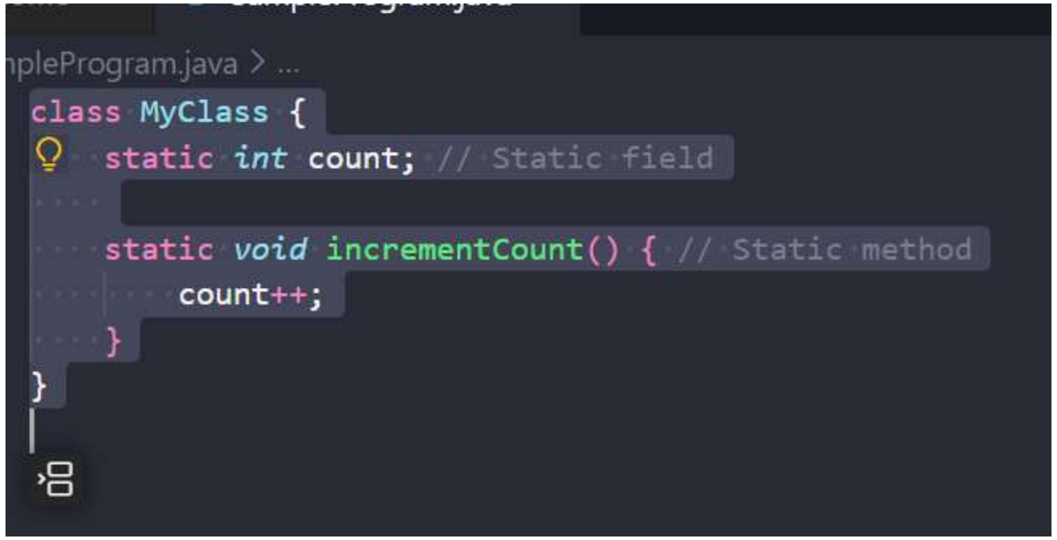

SampleProgram.java > ...

```
class Parent {
    void display() {
        System.out.println(x:"Parent class");
    }
}

class Child extends Parent {
    void display() {
        super.display(); // Calls the display method of the parent class
        System.out.println(x:"Child class");
    }
}
```

The static Keyword

In Java, the static keyword is used to define a class-level member that belongs to the class itself, rather than to instances of the class. Static methods and fields are accessible without creating an instance of the class.

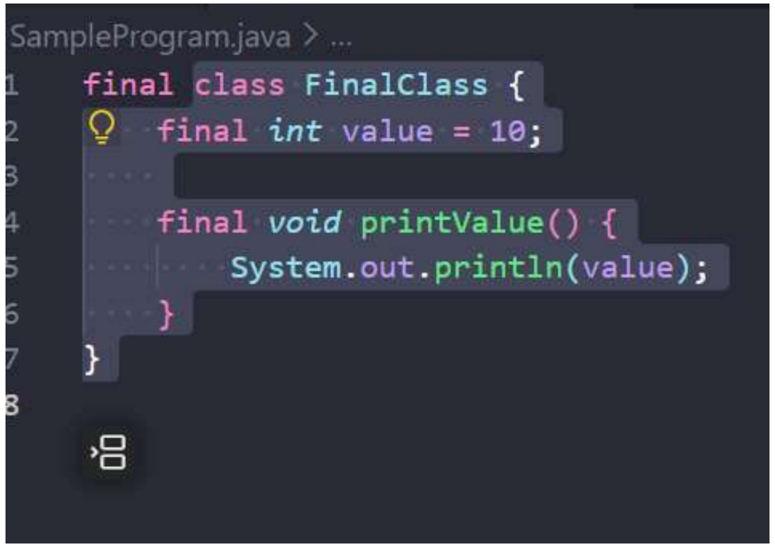


```
class MyClass {
    static int count; // Static field

    static void incrementCount() { // Static method
        count++;
    }
}
```

The final Keyword:

In Java, the final keyword can be applied to classes, methods, and fields. It indicates that a class cannot be extended, a method cannot be overridden, and a field cannot be changed after initialization.

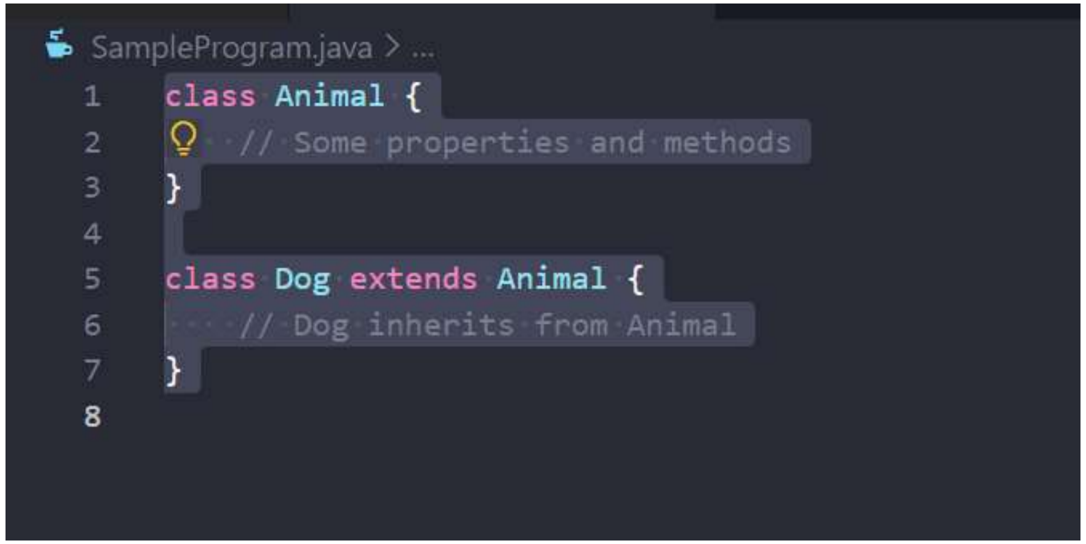


```java
final class FinalClass {
    final int value = 10;

    final void printValue() {
        System.out.println(value);
    }
}
```

The extends Keyword:

In Java, the extends keyword is used to indicate that a class is inheriting from another class. It establishes an "is-a" relationship between classes.



SampleProgram.java > ...

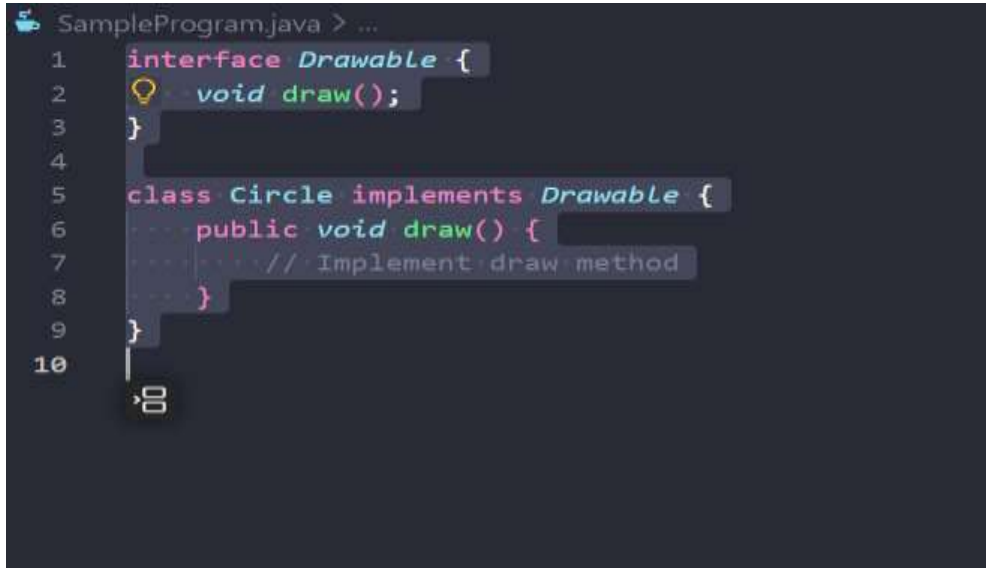
```java
class Animal {
    // Some properties and methods
}

class Dog extends Animal {
    // Dog inherits from Animal
}

```

The implements Keyword:

In Java, the implements keyword is used to indicate that a class is implementing one or more interfaces. It establishes a "can-do" relationship, allowing a class to fulfill the contract specified by the interface.



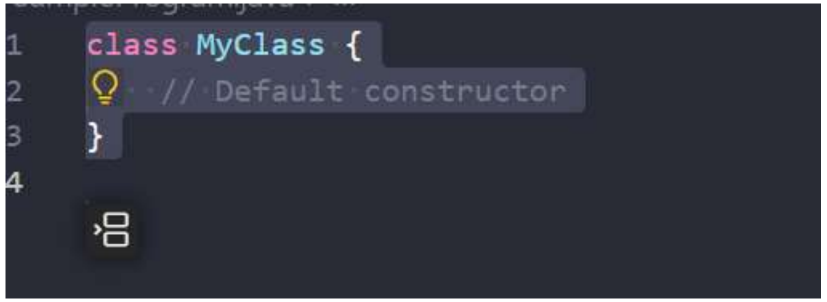
SampleProgram.java > ...

```java
interface Drawable {
    void draw();
}

class Circle implements Drawable {
    public void draw() {
        // Implement draw method
    }
}
```

Default Constructors:

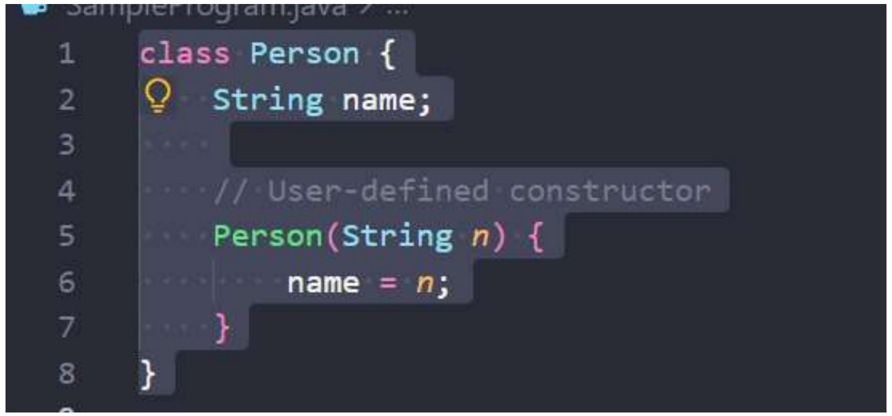
A default constructor is a constructor that is automatically generated by Java if no constructor is defined explicitly in a class. It initializes the object with default values or performs no action. It's especially useful when you want to create instances without specific initialization.



```
class MyClass {
    // Default constructor
}
```

User-Defined Constructors:

A user-defined constructor is a constructor that you define in a class to initialize its objects with specific values. It allows you to provide custom initialization logic.

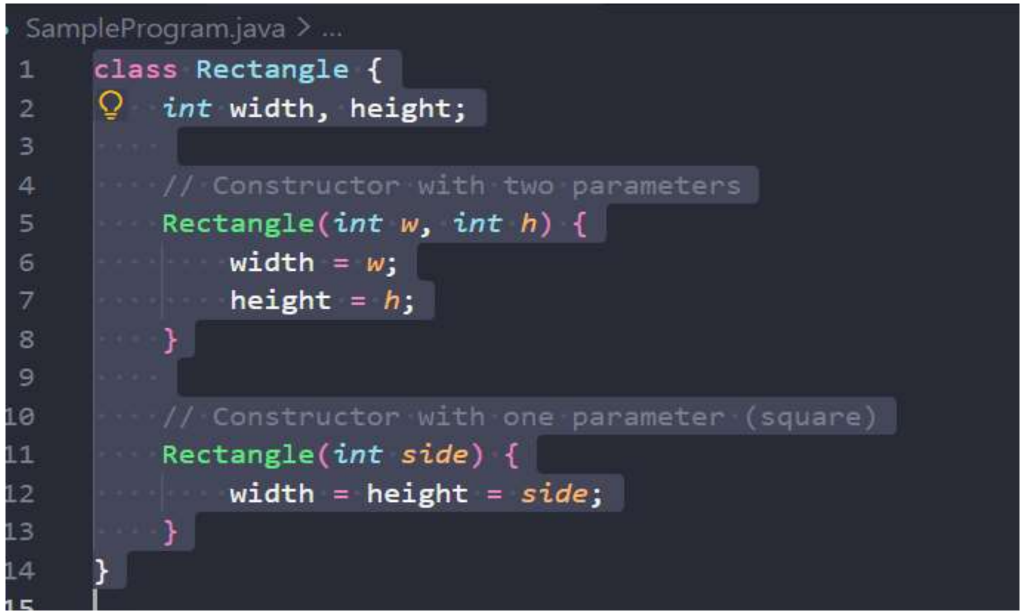


```
class Person {
    String name;

    // User-defined constructor
    Person(String n) {
        name = n;
    }
}
```

Constructor Overloading:

Constructor overloading involves defining multiple constructors in a class, each with a different parameter list. This allows objects to be initialized in various ways, providing flexibility to users.


SampleProgram.java > ...

```java
class Rectangle {
    int width, height;

    // Constructor with two parameters
    Rectangle(int w, int h) {
        width = w;
        height = h;
    }

    // Constructor with one parameter (square)
    Rectangle(int side) {
        width = height = side;
    }
}
```