

Programación I

Programación Orientada a Objetos (POO)

POO

Es un paradigma de programación que utiliza "objetos" para diseñar aplicaciones y programas. Un objeto es una instancia de una clase, que es una plantilla o blueprint que define las propiedades y comportamientos (métodos) comunes.

Programación Estructurada VS POO

Programación Estructurada:

- Se basa en una secuencia de procedimientos o funciones.
- El enfoque está en las funciones y la lógica.
- Los datos y las funciones están separados.
- Ejemplo: C

Programación Orientada a Objetos:

- Se basa en objetos que contienen datos y métodos.
- El enfoque está en los objetos y la interacción entre ellos.
- Los datos y los métodos que operan sobre esos datos están encapsulados juntos.
- Ejemplo: Java

Beneficios de la Aplicación de la POO



Reusabilidad: Se pueden reutilizar clases y objetos en diferentes programas.

Mantenibilidad: El código es más fácil de mantener y actualizar.

Modularidad: El código está organizado en módulos (clases) independientes.

Escalabilidad: Facilita la ampliación y modificación de aplicaciones grandes.

Programación I

Principios de la Programación Orientada a Objetos

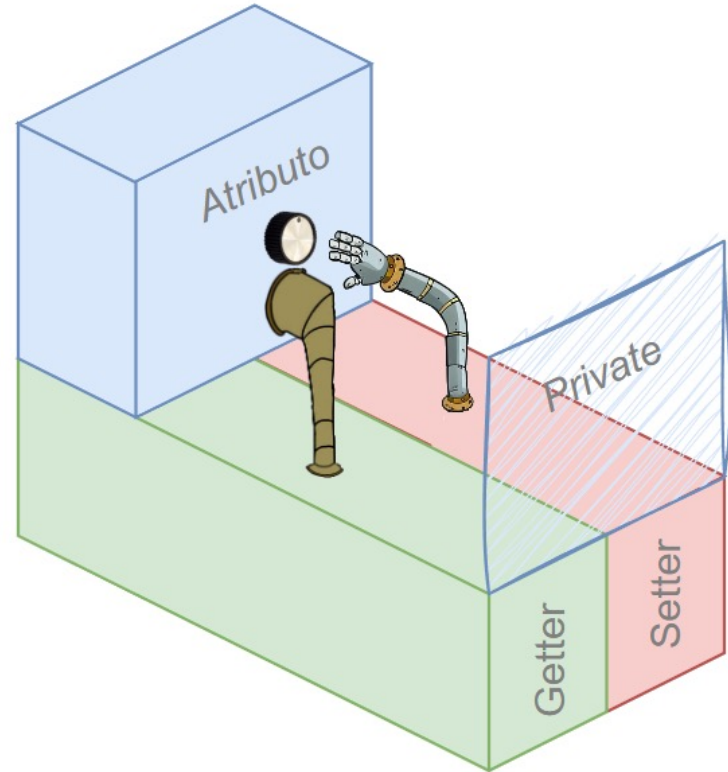
Abstracción

Ocultar los detalles complejos mostrando solo la información esencial.

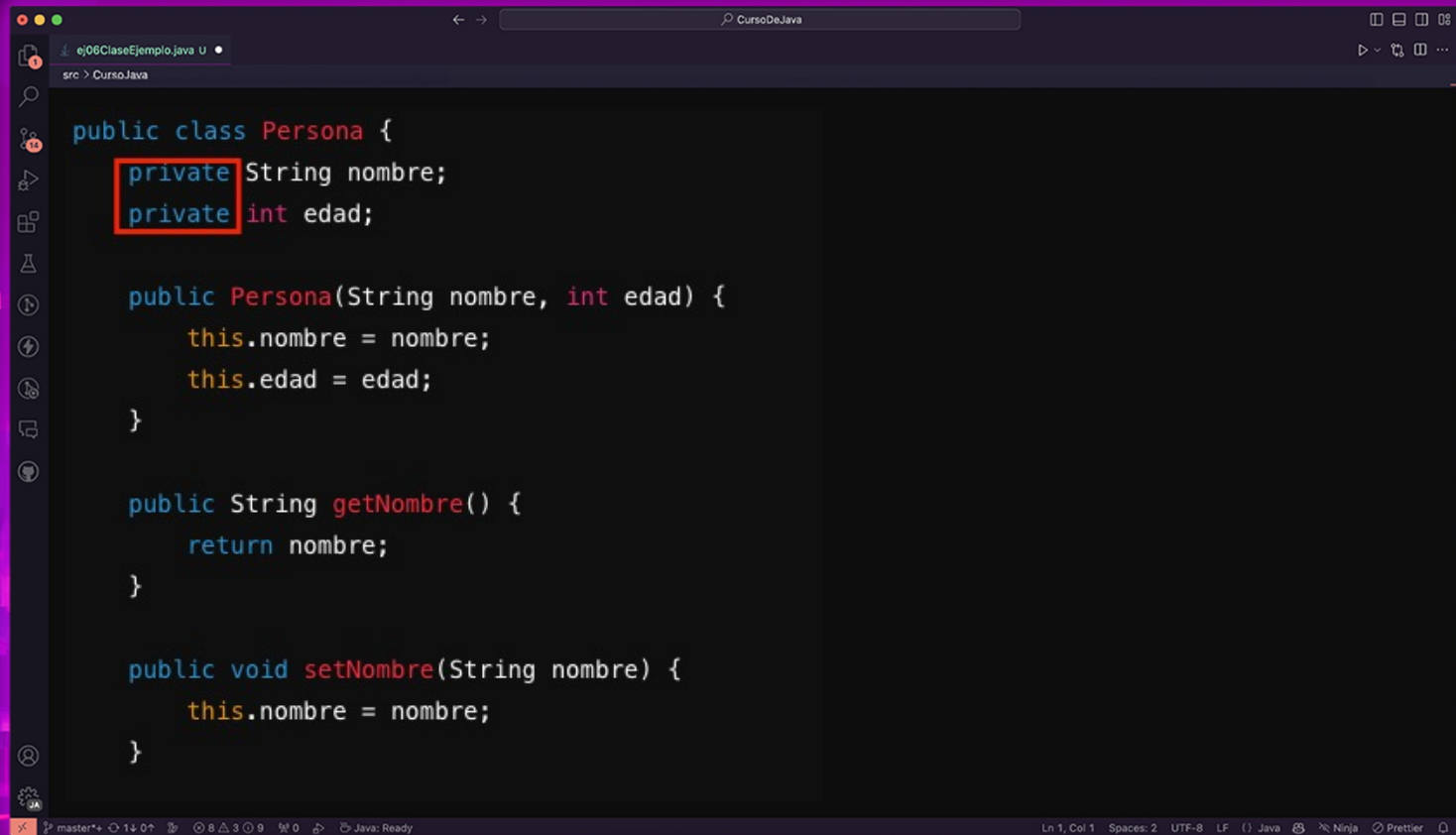


Encapsulamiento

El encapsulamiento en Java es el concepto de ocultar los detalles internos de una clase y proporcionar una interfaz pública para interactuar con la clase. Esto se logra mediante el uso de modificadores de acceso como `private`, `public` y `protected`, así como mediante el uso de métodos de acceso (getters) y métodos de modificación (setters) para acceder a los datos de la clase.

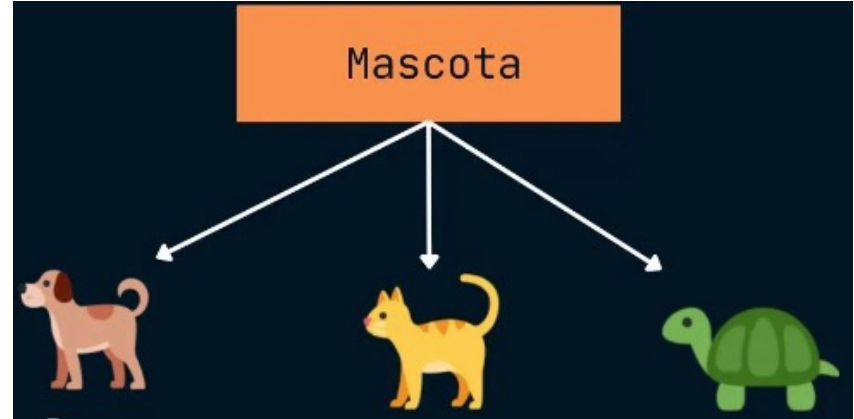


Encapsulamiento

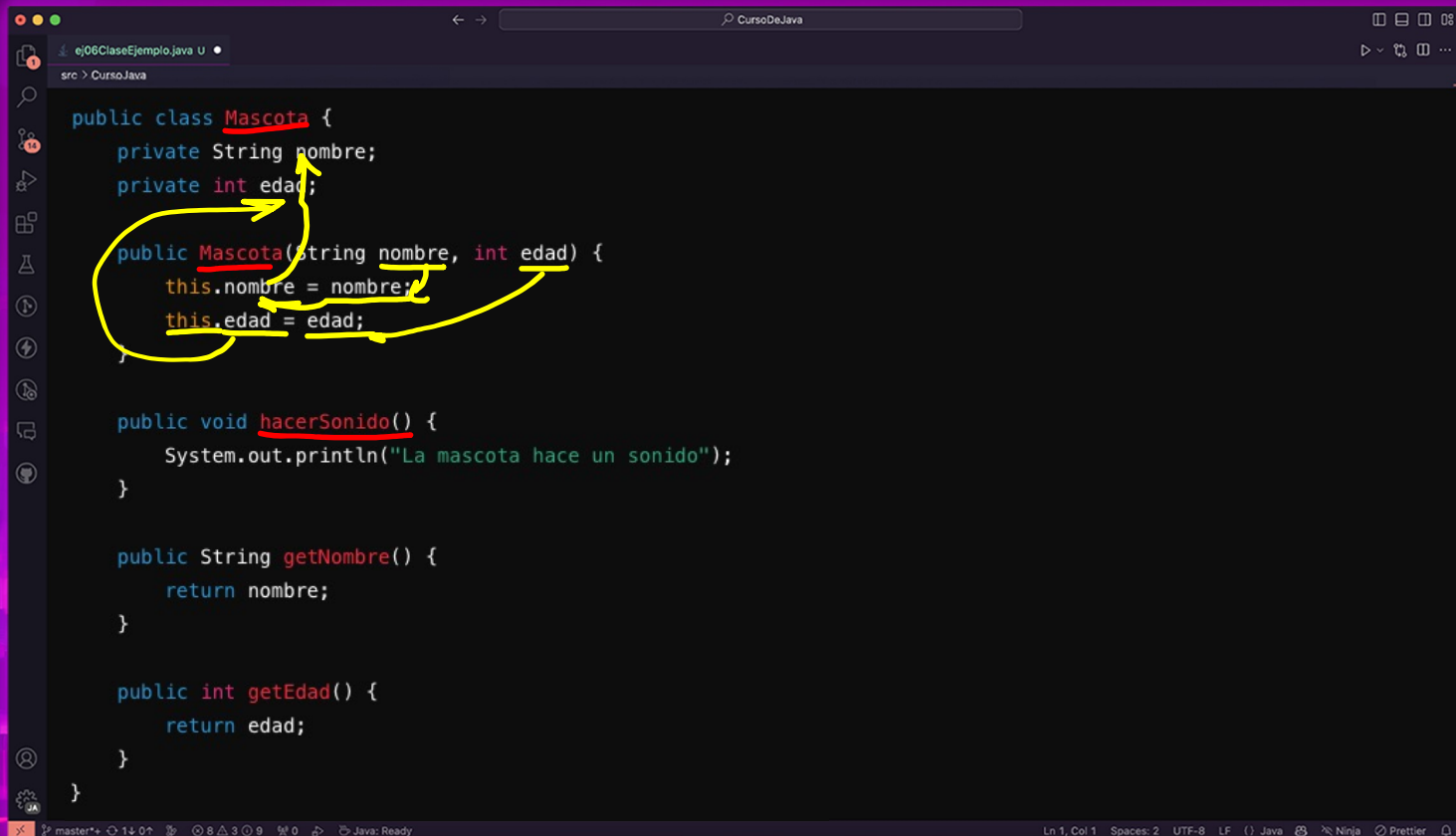


Herencia

Permite crear nuevas clases basadas en clases existentes. La clase nueva, llamada subclase o clase derivada, hereda los atributos y métodos de la clase existente, llamada superclase o clase base. Esto facilita la reutilización del código y establece una relación "es-un" entre las clases.



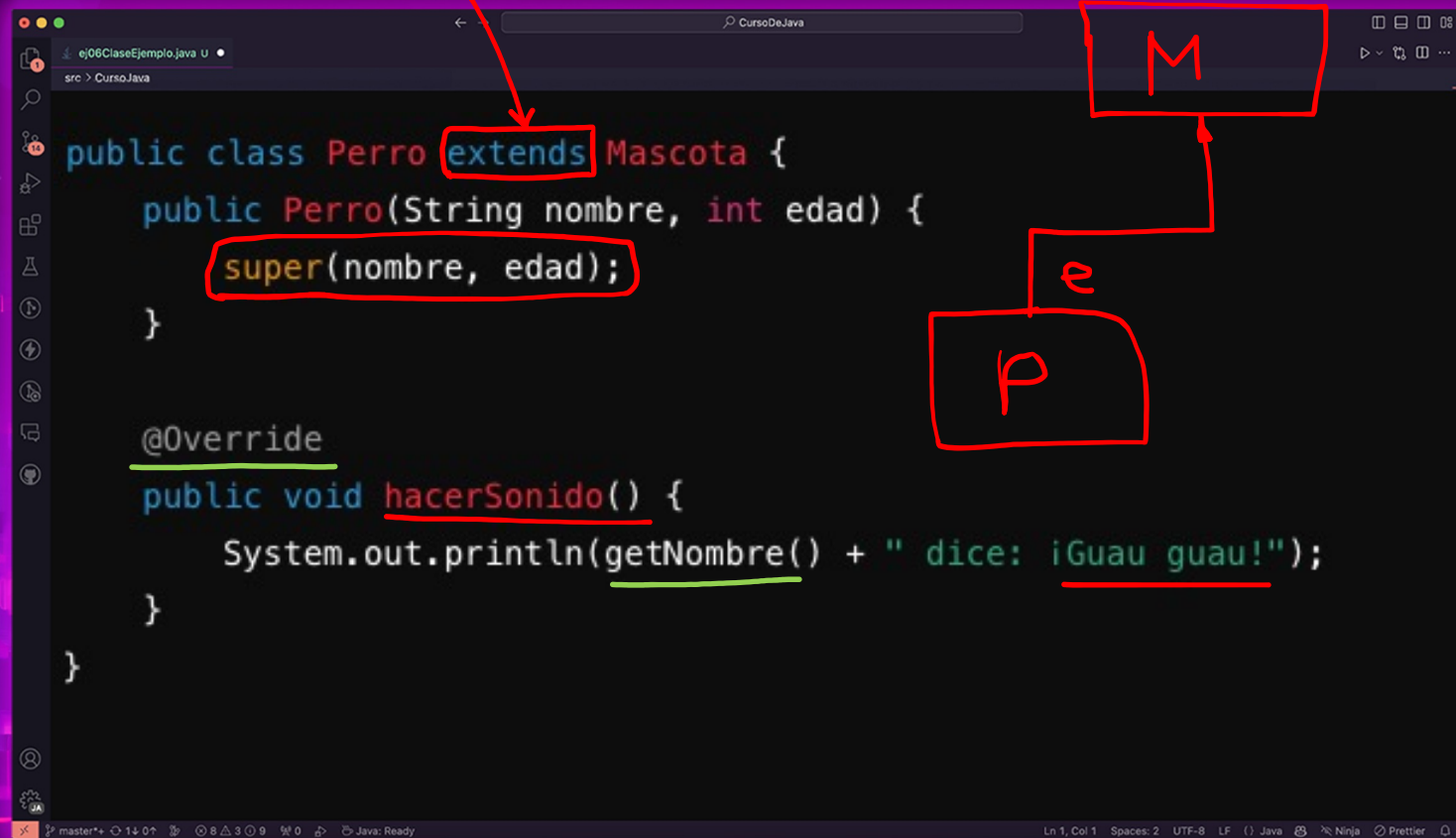
Herencia



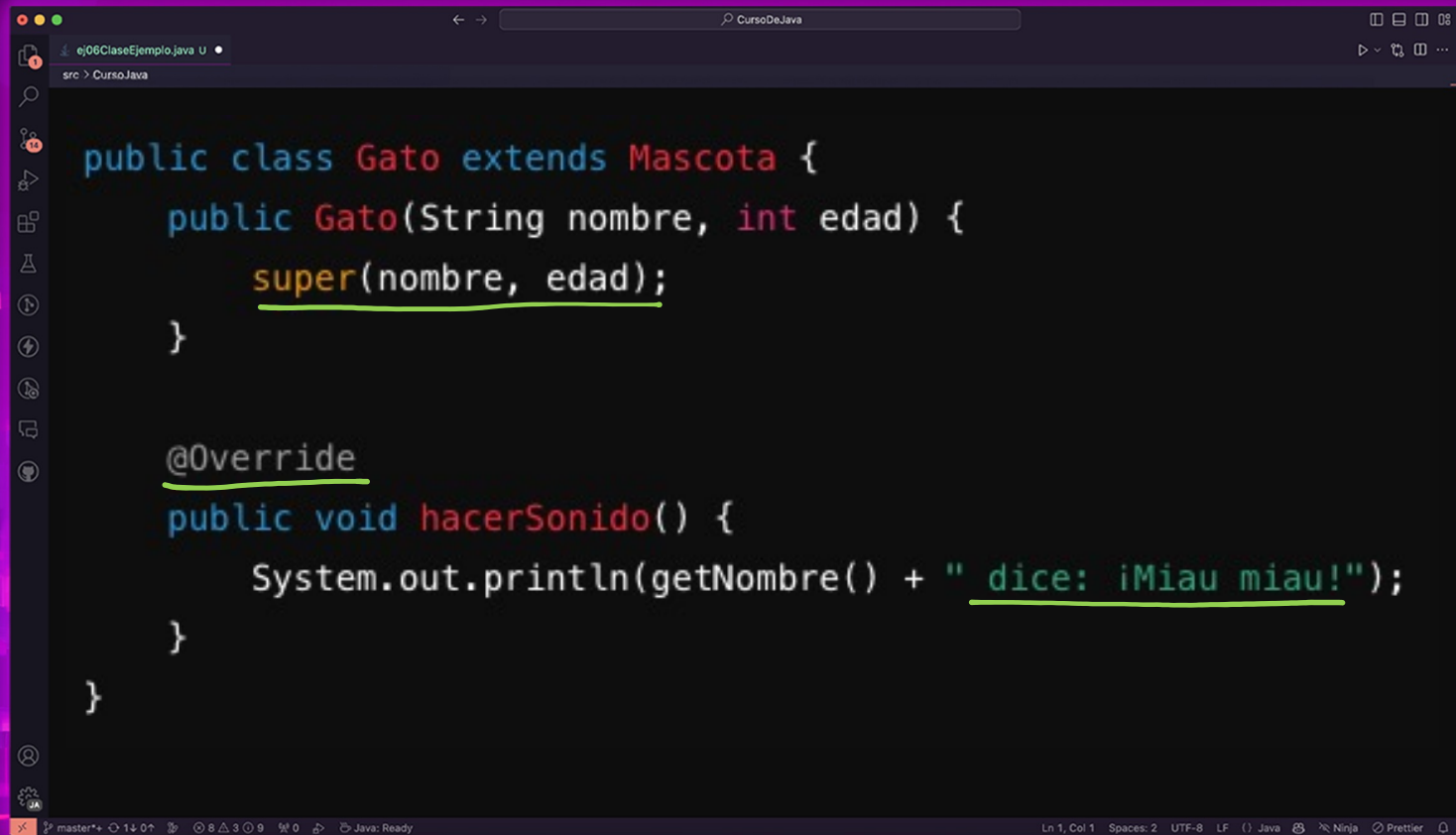
```
public class Mascota {  
    private String nombre;  
    private int edad;  
  
    public Mascota(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
  
    public void hacerSonido() {  
        System.out.println("La mascota hace un sonido");  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public int getEdad() {  
        return edad;  
    }  
}
```

The screenshot shows a code editor with a Java class named `Mascota`. The class has two private attributes: `String nombre` and `int edad`. It has a constructor `Mascota(String nombre, int edad)` that initializes these attributes using `this.nombre = nombre;` and `this.edad = edad;`. There are also two getter methods, `getNombre()` and `getEdad()`, and one method `hacerSonido()` that prints a message. Annotations include a red underline on the class name `Mascota`, a red underline on the constructor name `Mascota`, and a red underline on the `hacerSonido()` method name. Yellow arrows point from the `nombre` parameter in the constructor to `this.nombre`, and from the `edad` parameter to `this.edad`. The IDE interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, Extensions, Testing, Remote Explorer, Docker, and Settings. The top bar shows the file path `src > CursoJava` and the file name `ej06ClaseEjemplo.java`. The bottom status bar shows `Ln 1, Col 1`, `Spaces: 2`, `UTF-8`, `LF`, `() Java`, `Ninja`, and `Prettier`.

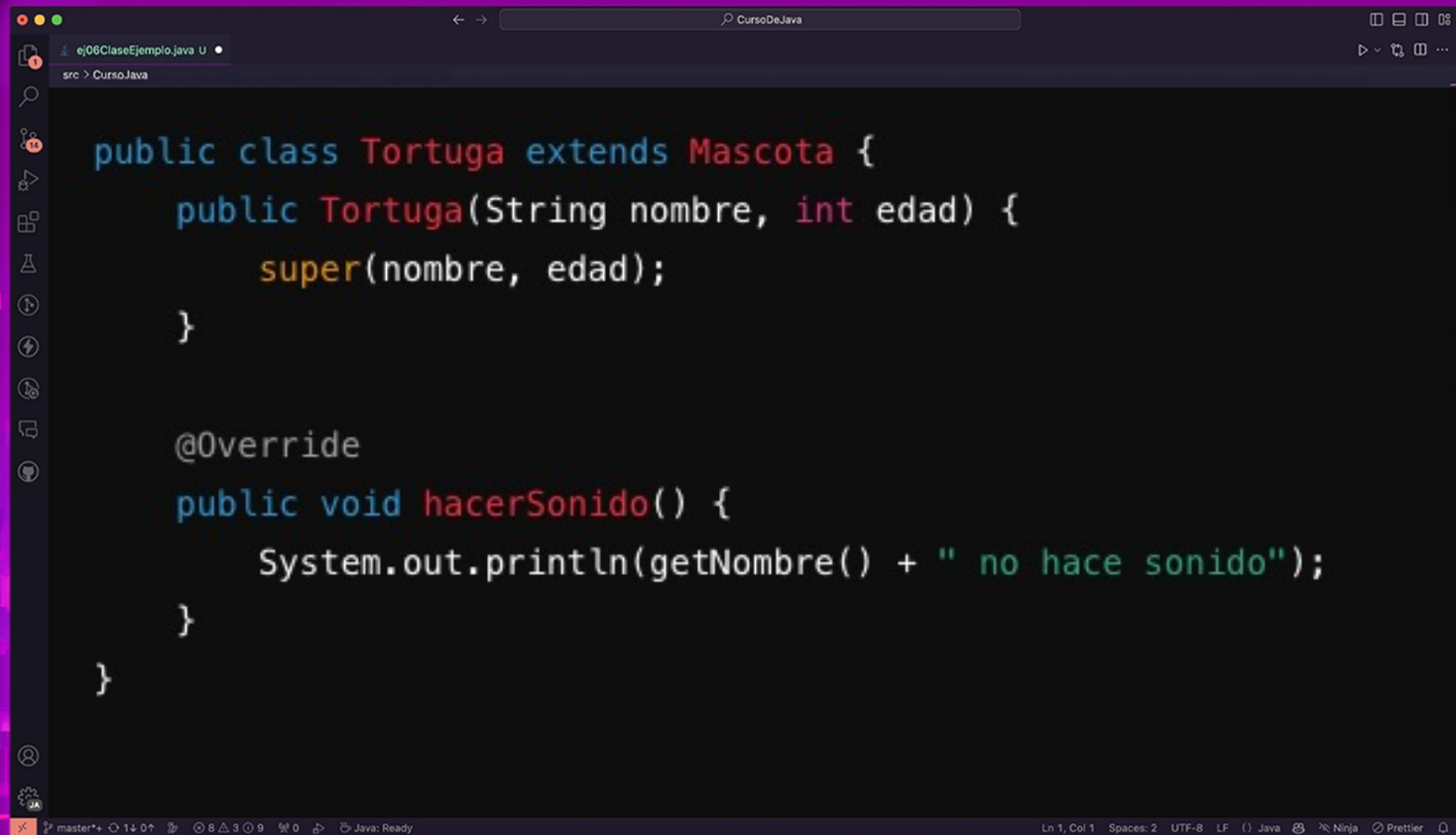
Herencia



Herencia

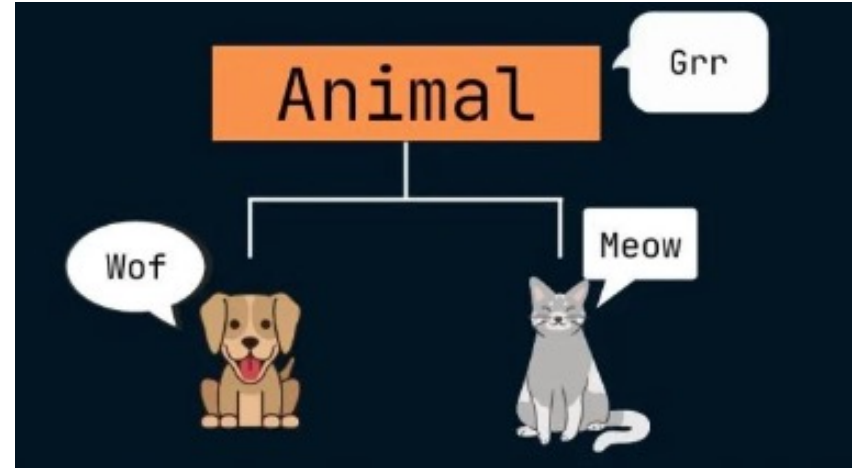


Herencia

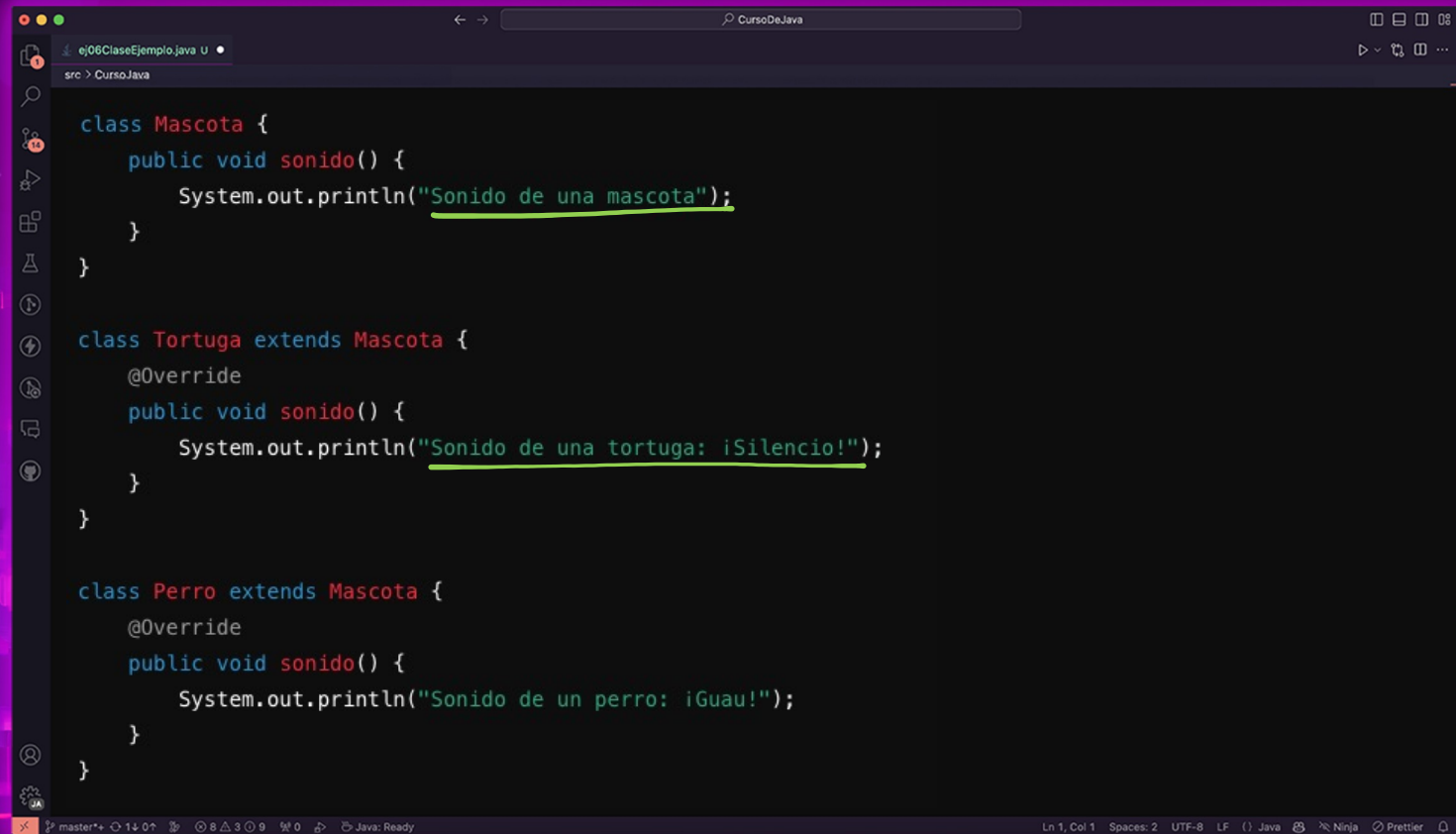


Polimorfismo

Capacidad de objetos de diferentes clases de responder al mismo mensaje o llamada a un método de manera distinta, según la implementación específica de cada clase. En otras palabras, el polimorfismo permite que un objeto pueda comportarse de múltiples formas.



Polimorfismo



```
class Mascota {  
    public void sonido() {  
        System.out.println("Sonido de una mascota");  
    }  
}  
  
class Tortuga extends Mascota {  
    @Override  
    public void sonido() {  
        System.out.println("Sonido de una tortuga: ¡Silencio!");  
    }  
}  
  
class Perro extends Mascota {  
    @Override  
    public void sonido() {  
        System.out.println("Sonido de un perro: ¡Guau!");  
    }  
}
```

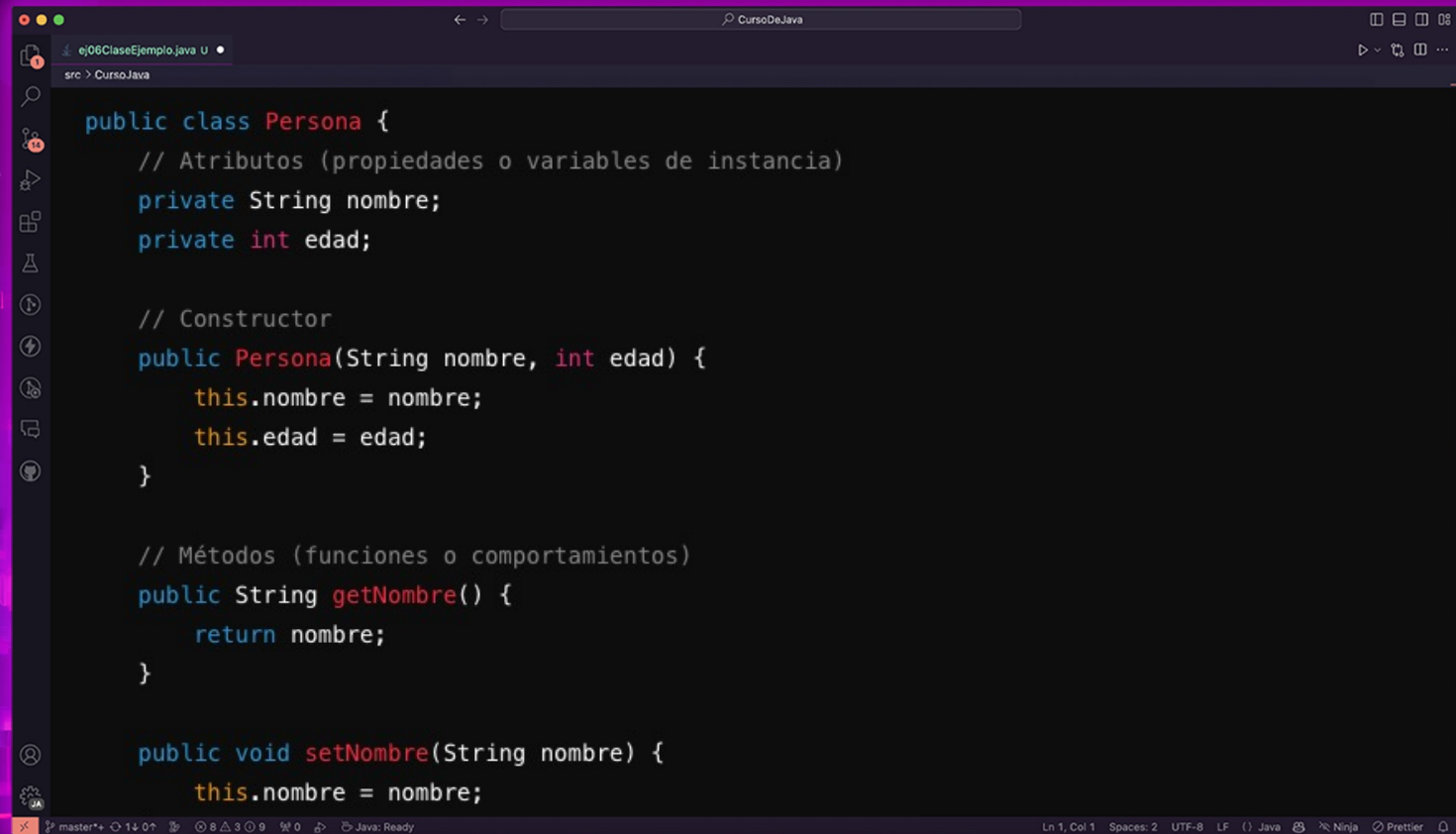
The image shows a code editor window with the title 'CursoDeJava'. The file being edited is 'ej06ClaseEjemplo.java'. The code defines a base class 'Mascota' with a 'sonido()' method that prints 'Sonido de una mascota'. Two subclasses, 'Tortuga' and 'Perro', extend 'Mascota'. 'Tortuga' overrides 'sonido()' to print 'Sonido de una tortuga: ¡Silencio!', and 'Perro' overrides it to print 'Sonido de un perro: ¡Guau!'. The IDE interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, Extensions, Testing, Docker, Remote Explorer, and Settings. The status bar at the bottom shows 'master+ 140', '8 3 0', 'Java: Ready', and 'Ln 1, Col 1 Spaces: 2 UTF-8 LF () Java Ninja Prettier'.

Clases

Es un molde para crear objetos. Define un tipo de objeto según sus atributos (datos) y métodos (funciones).

Es un modelo que especifica qué atributos y métodos tendrán los objetos creados a partir de esa clase.

Clases



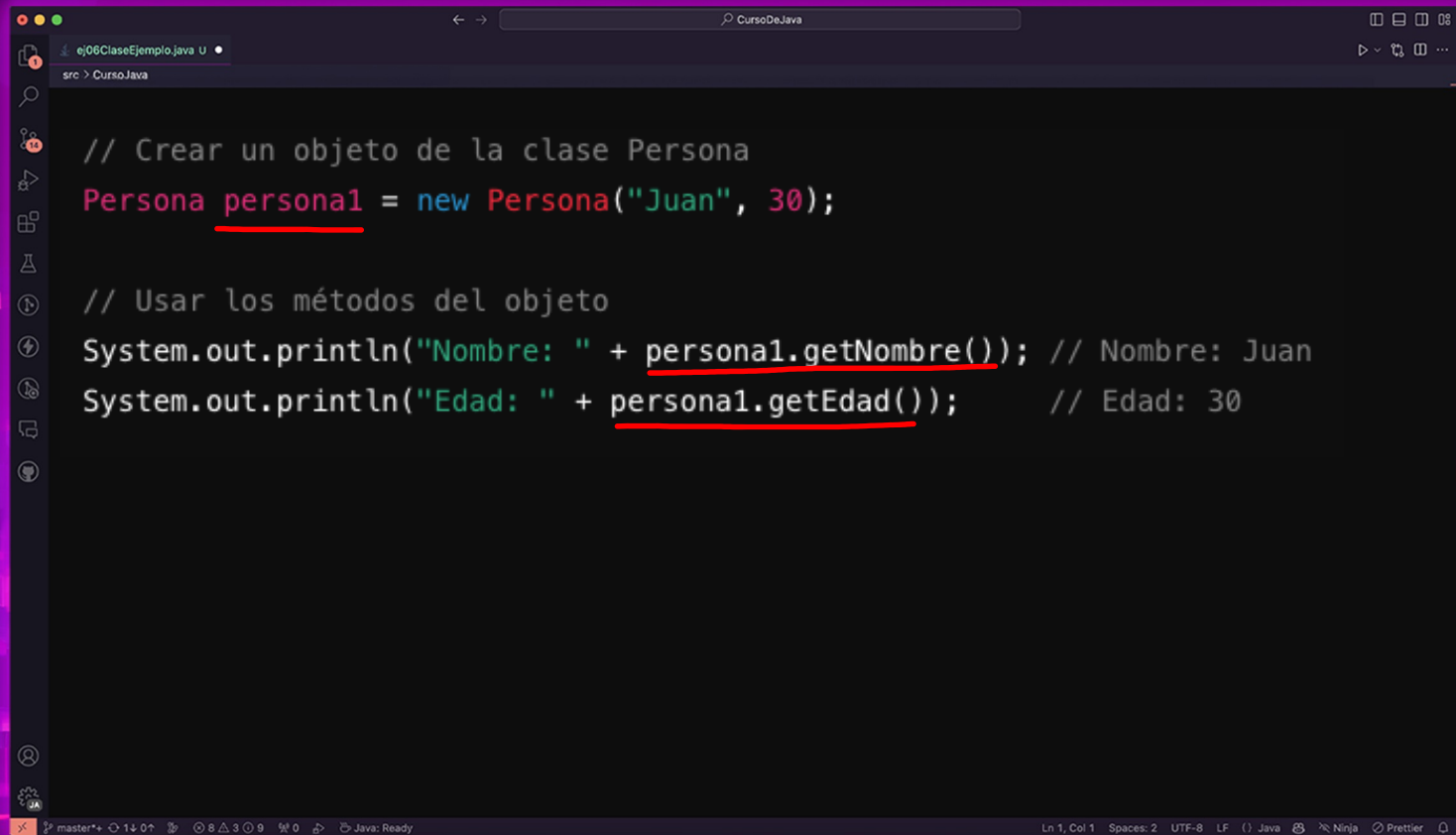
Objeto

Es una instancia de una clase.

Es una entidad concreta que tiene un estado y un comportamiento definido por la clase a partir de la cual se creó.

Cada objeto tiene su propio conjunto de valores para los atributos definidos en su clase.

Objeto



```
// Crear un objeto de la clase Persona
Persona persona1 = new Persona("Juan", 30);

// Usar los métodos del objeto
System.out.println("Nombre: " + persona1.getNombre()); // Nombre: Juan
System.out.println("Edad: " + persona1.getEdad());      // Edad: 30
```

The image shows a code editor window with a dark theme. The file name is 'ej06ClaseEjemplo.java'. The code is in Java and demonstrates creating a 'Persona' object and using its methods. The variable 'persona1' and the method calls 'getNombre()' and 'getEdad()' are underlined in the original image. The IDE interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, and Extensions. The status bar at the bottom shows 'master+', 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'Java', and 'Prettier'.

Diferencias entre Clases y Objetos

Clase: Define un tipo de objeto con atributos y métodos.

Objeto: Una instancia de una clase que tiene un estado particular y puede ejecutar métodos definidos en la clase.

Diferencias entre Clases y Objetos

Definición vs. Instancia

Clase

Es una definición, una plantilla. No es una entidad tangible.

Objeto

Es una instancia concreta de una clase. Es una entidad tangible que ocupa memoria y tiene un estado específico.

Diferencias entre Clases y Objetos

Abstracción vs. Concreción

Clase

Representa la **abstracción** de algo. Define qué características y comportamientos deberían tener los objetos.

Objeto

Representa la **concreción** de esa abstracción. Tiene características y comportamientos específicos definidos por la clase.

Diferencias entre Clases y Objetos

Reusabilidad

Clase

Puede ser utilizada para crear múltiples objetos.

Objeto

Es una única instancia de una clase específica.

Diferencias entre Clases y Objetos

Estructura vs. Estado

Clase

Define la estructura (atributos) y el comportamiento (métodos) comunes a todos los objetos de ese tipo.

Objeto

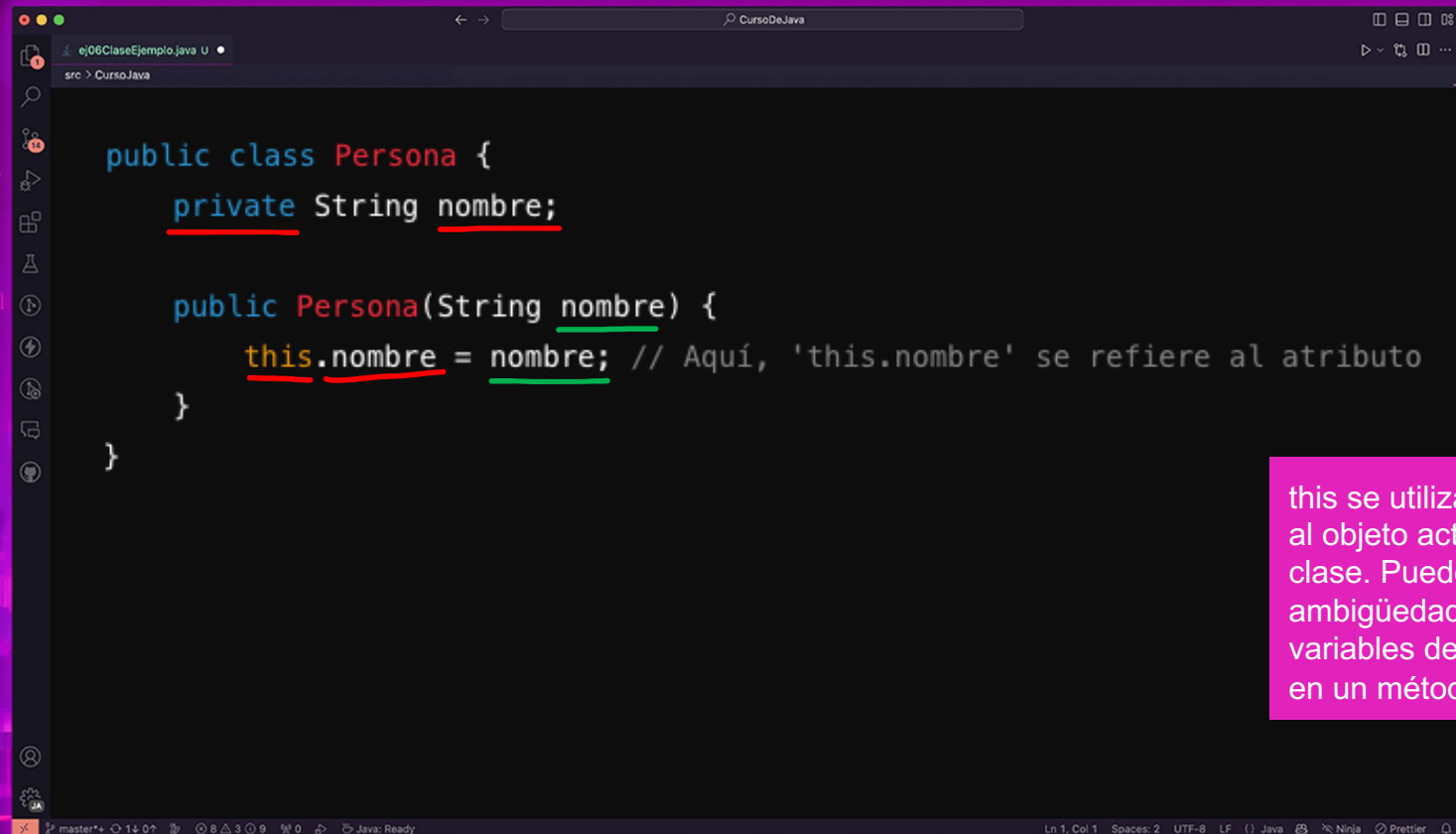
Tiene un estado particular (valores de atributos) y puede ejecutar comportamientos definidos por su clase.



this

La palabra clave `this` en Java se refiere a la instancia actual de una clase. Tiene varios usos importantes en la programación orientada a objetos

Referencia al objeto actual



```
public class Persona {  
    private String nombre;  
  
    public Persona(String nombre) {  
        this.nombre = nombre; // Aquí, 'this.nombre' se refiere al atributo  
    }  
}
```

this se utiliza para hacer referencia al objeto actual dentro de la propia clase. Puede ser útil cuando hay ambigüedad entre los nombres de variables de instancia y parámetros en un método.

Llamada al constructor de la misma clase

```
ej06ClaseEjemplo.java U
src > CursoDeJava

public class Persona {
    private String nombre;
    private int edad;

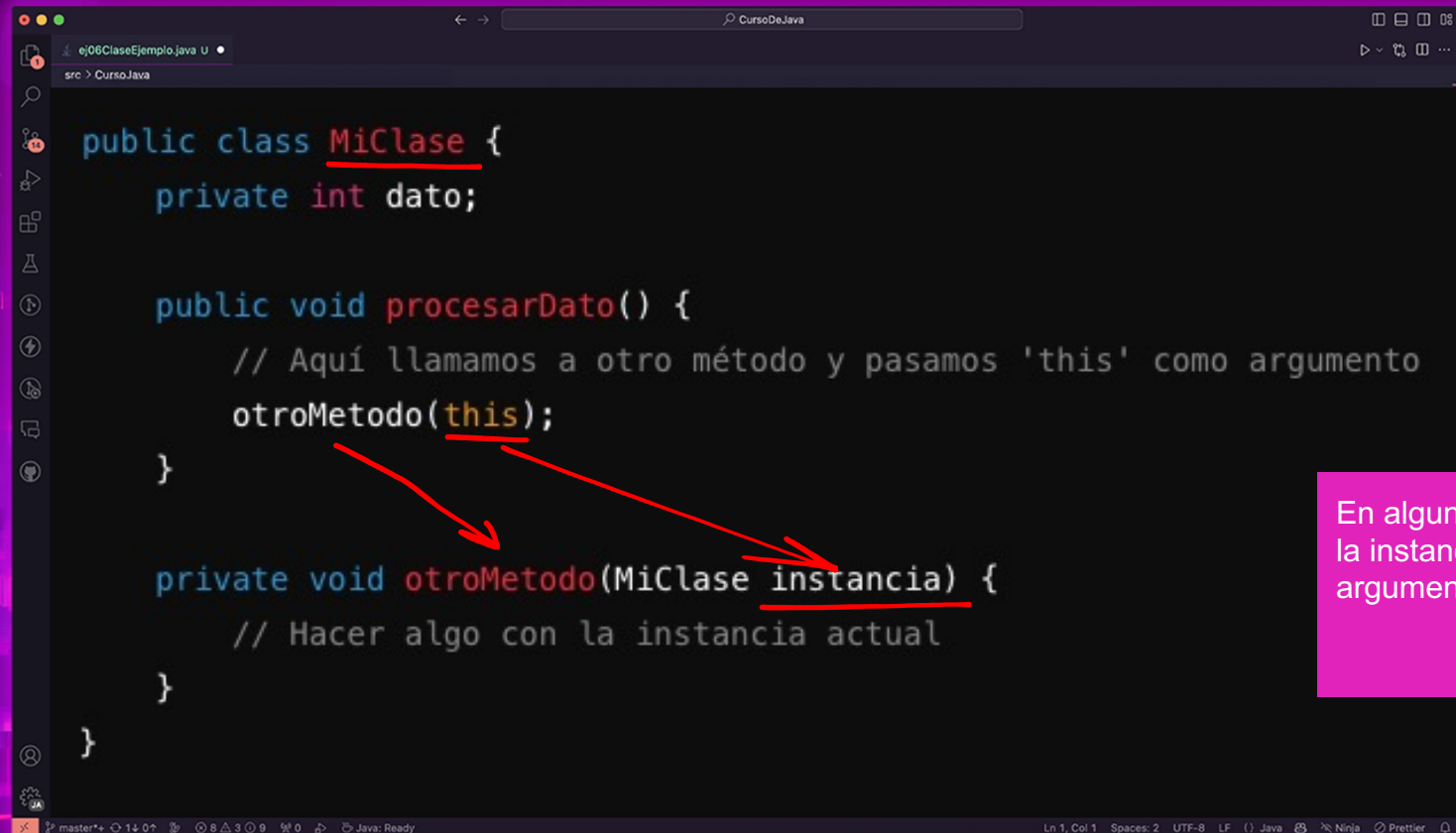
    public Persona(String nombre) {
        this(nombre, 0); // Llama al otro constructor de Persona con dos parámetros
    }

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

this se puede usar para llamar a otro constructor de la misma clase desde un constructor. Esta técnica se llama "reutilización de constructores" o "llamada al constructor".



Llamada al constructor de la misma clase



```
public class MiClase {  
    private int dato;  
  
    public void procesarDato() {  
        // Aquí llamamos a otro método y pasamos 'this' como argumento  
        otroMetodo(this);  
    }  
  
    private void otroMetodo(MiClase instancia) {  
        // Hacer algo con la instancia actual  
    }  
}
```

En algunos casos, es útil pasar la instancia actual como argumento a otro método.

Sobrecarga de métodos

Métodos con el Mismo
Indicador y Distinto Número
de Parámetros

Sobrecarga de métodos

```
ej06ClaseEjemplo.java U
src > CursoDeJava

class Matematica {
    public int sumar(int a, int b) {
        return a + b;
    }

    public int sumar(int a, int b, int c) {
        return a + b + c;
    }
}
```

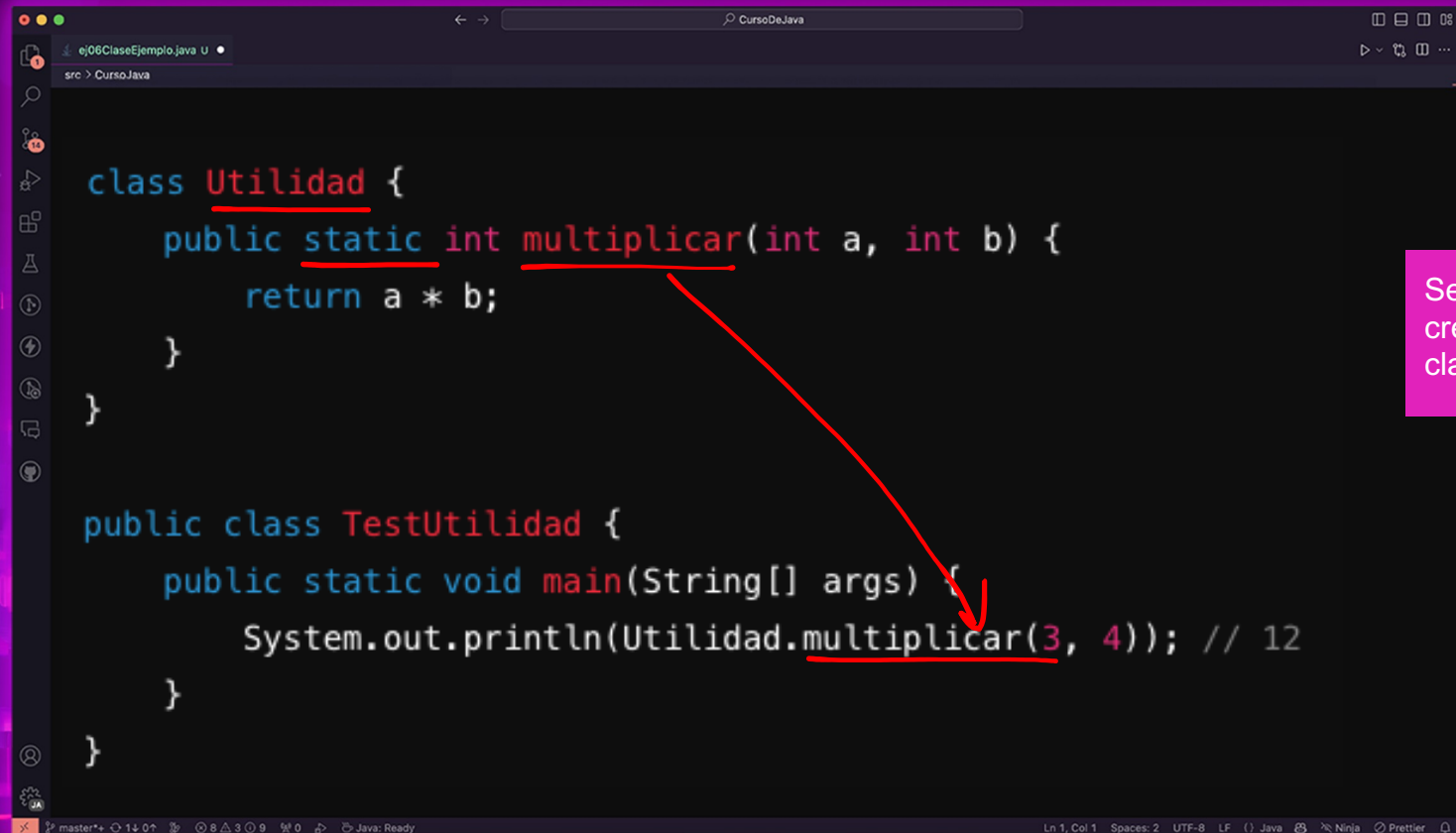
Ln 1, Col 1 Spaces: 2 UTF-8 LF () Java Ninja Prettier

En Java, se pueden definir múltiples métodos con el mismo nombre pero diferentes listas de parámetros.

Métodos Estáticos

Los métodos estáticos pertenecen a la clase, no a una instancia de la clase. Se pueden llamar sin crear un objeto de la clase.

Métodos Estáticos



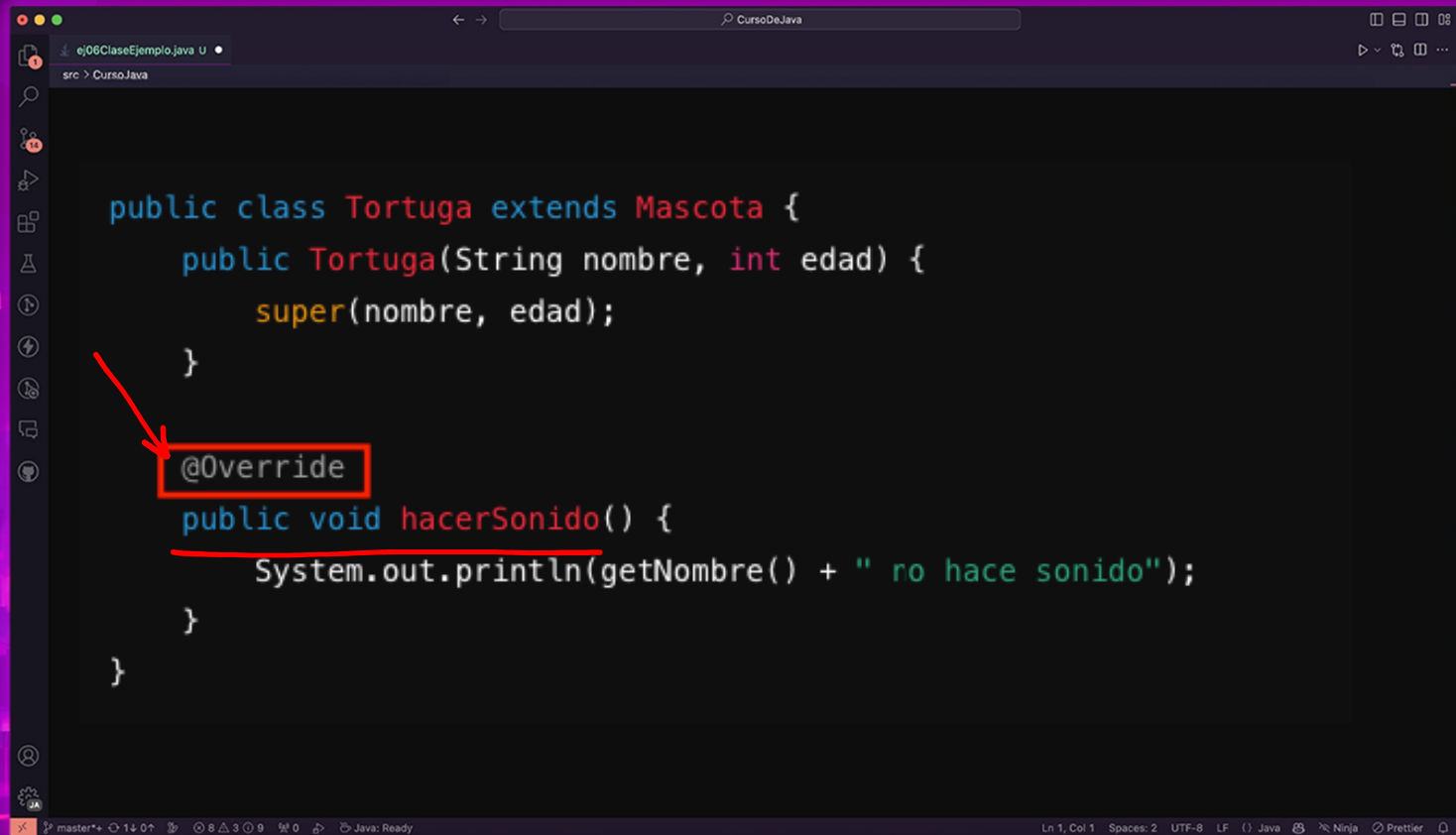
```
class Utilidad {  
    public static int multiplicar(int a, int b) {  
        return a * b;  
    }  
}  
  
public class TestUtilidad {  
    public static void main(String[] args) {  
        System.out.println(Utilidad.multiplicar(3, 4)); // 12  
    }  
}
```

Se pueden llamar sin crear un objeto de la clase.

Sobrescritura

La sobrescritura (`@Override`) permite que una subclase proporcione una implementación específica de un método que ya está definido en su superclase.

Sobrescritura



Recursividad de Métodos

Un método recursivo es un método que se llama a sí mismo para resolver un problema.

Recursividad de Métodos

```
ej06ClaseEjemplo.java U
src > CursoJava

class Matematica {
    public static int factorial(int n) {
        if (n == 0) {
            return 1;
        }
        return n * factorial(n - 1);
    }
}

public class TestMatematica {
    public static void main(String[] args) {
        System.out.println(Matematica.factorial(5)); // 120
    }
}
```

Factorial de un
Número

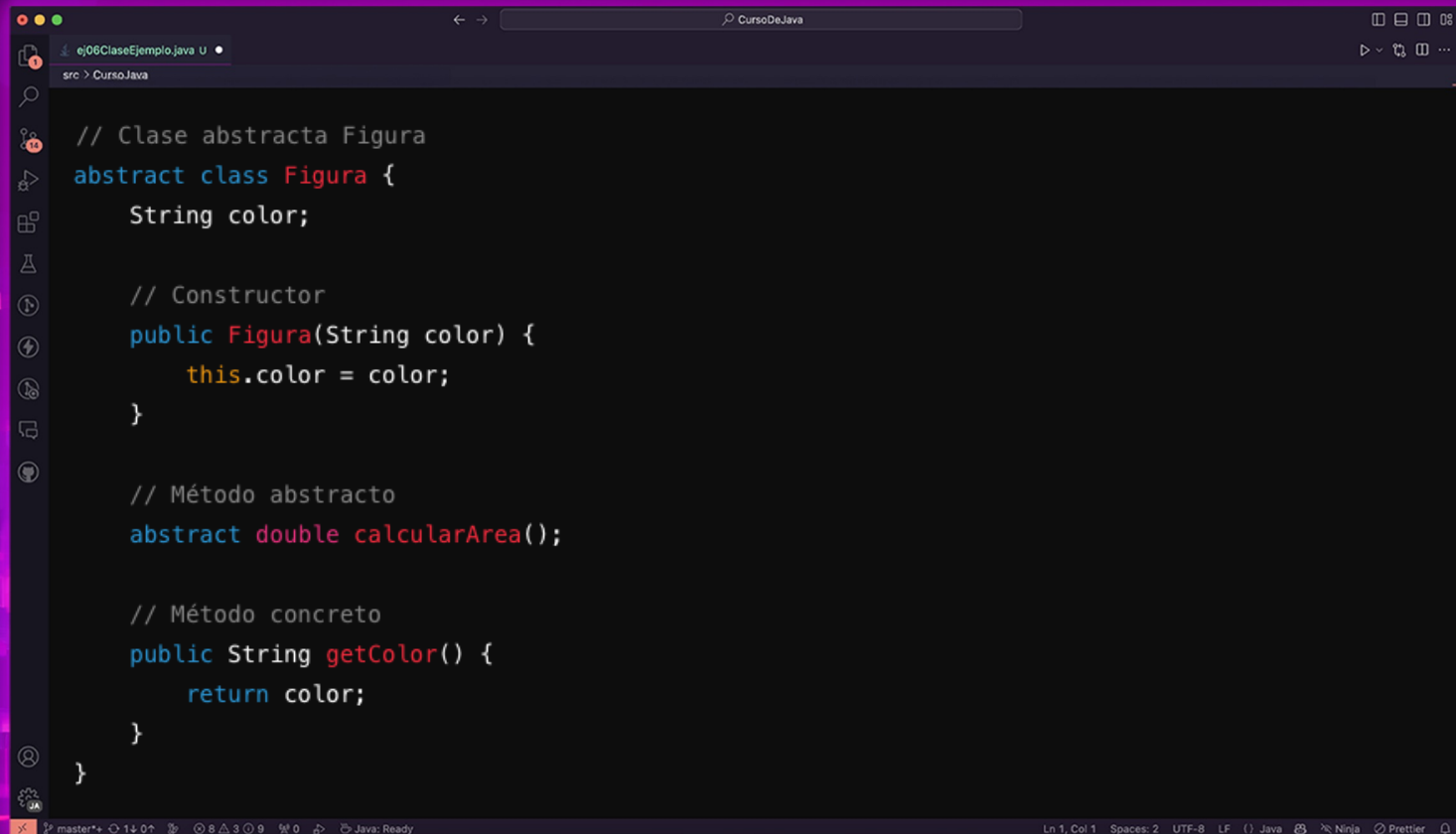
Clases Abstractas

Una clase abstracta en Java es una clase que no se puede instanciar, es decir, no se puede crear un objeto directamente a partir de ella. Las clases abstractas están diseñadas para ser extendidas por otras clases que implementan los métodos abstractos declarados en la clase abstracta. Una clase abstracta puede contener métodos abstractos (sin implementación) y métodos concretos (con implementación).

Características de las Clases Abstractas

1. **No se pueden instanciar:** No puedes crear objetos de una clase abstracta.
2. **Métodos abstractos:** Puede tener métodos abstractos, que son métodos sin cuerpo que deben ser implementados por las clases derivadas.
3. **Métodos concretos:** Puede tener métodos concretos, que son métodos con cuerpo.
4. **Constructores:** Aunque no se puede instanciar, una clase abstracta puede tener constructores que pueden ser llamados por sus subclases.

Clase Abstracta Básica



The image shows a code editor window with a dark theme. The title bar at the top says "CursoDeJava". The file name in the tab is "ej06ClaseEjemplo.java". The code is written in Java and defines an abstract class named "Figura". It includes a private field "color" of type "String", a constructor "Figura(String color)" that initializes "this.color", an abstract method "calcularArea()" of type "double", and a concrete method "getColor()" of type "String" that returns the "color" field. The editor has a sidebar on the left with various icons for file management and a status bar at the bottom showing "master+", "140", "8", "3", "0", "9", "0", "Java: Ready", "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "Java", "Ninja", and "Prettier".

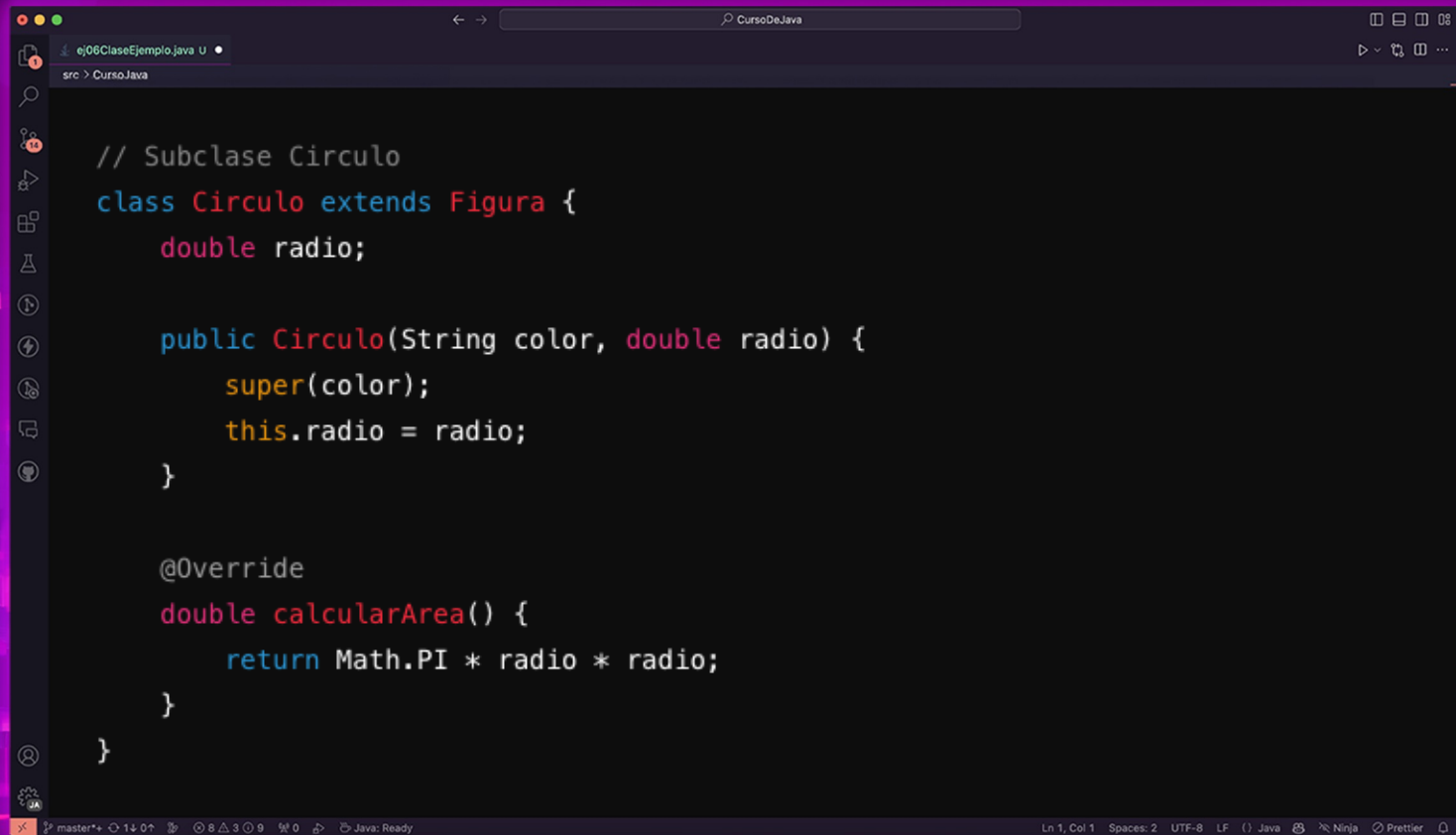
```
// Clase abstracta Figura
abstract class Figura {
    String color;

    // Constructor
    public Figura(String color) {
        this.color = color;
    }

    // Método abstracto
    abstract double calcularArea();

    // Método concreto
    public String getColor() {
        return color;
    }
}
```

Clase Abstracta Básica



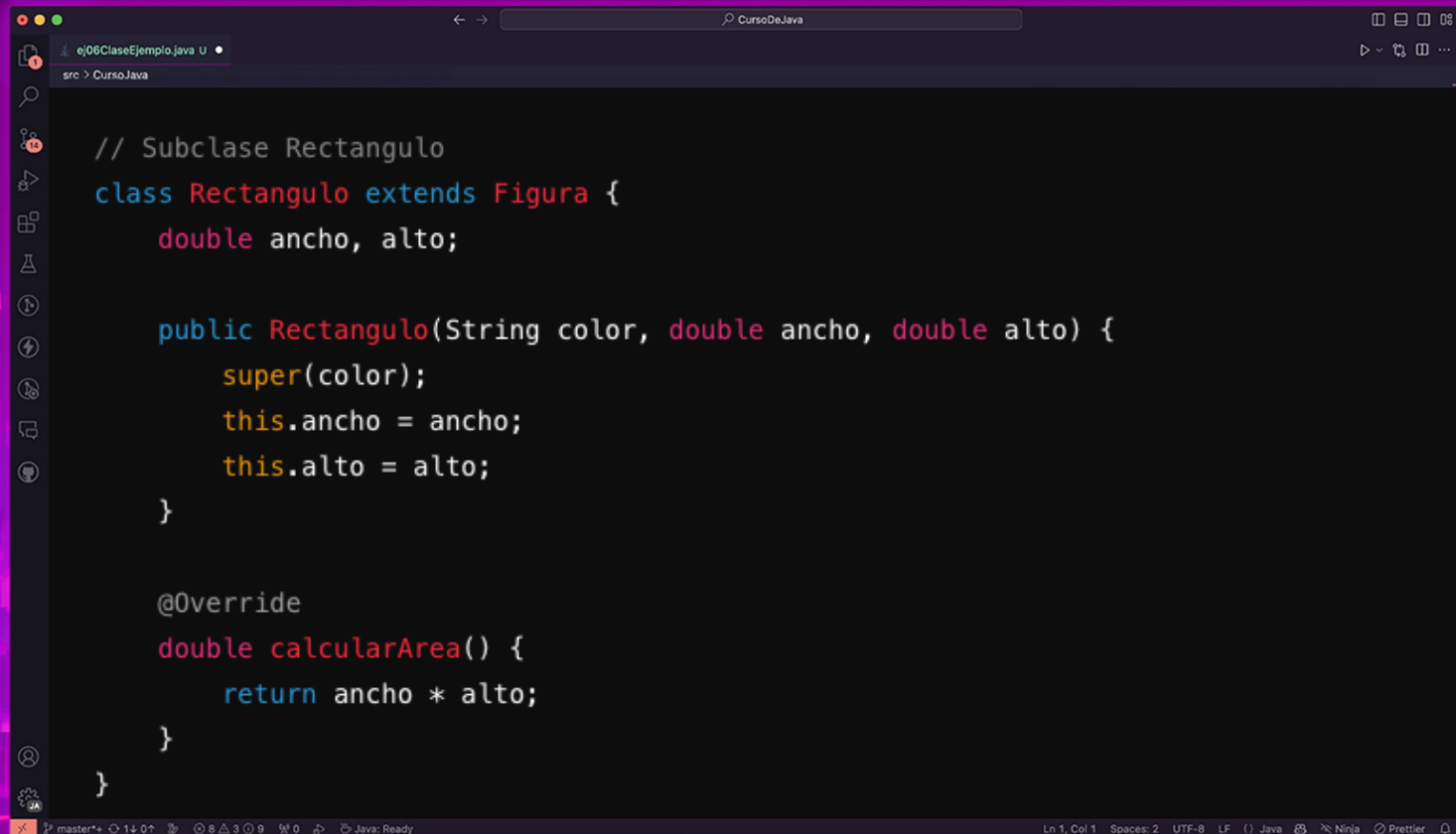
The image shows a screenshot of a code editor window with a dark theme. The title bar at the top indicates the file is 'ej06ClaseEjemplo.java' and the project is 'CursoDeJava'. The code defines a 'Circulo' class that extends a 'Figura' class. It includes a 'radio' attribute, a constructor, and an overridden 'calcularArea()' method. The status bar at the bottom shows the current cursor position as 'Ln 1, Col 1' and lists various settings like 'Spaces: 2', 'UTF-8', and 'LF'.

```
// Subclase Circulo
class Circulo extends Figura {
    double radio;

    public Circulo(String color, double radio) {
        super(color);
        this.radio = radio;
    }

    @Override
    double calcularArea() {
        return Math.PI * radio * radio;
    }
}
```

Clase Abstracta Básica



The screenshot shows an IDE window with a dark theme. The title bar at the top says "CursoDeJava". The file explorer on the left shows a project named "ej06ClaseEjemplo.java". The main editor area contains the following Java code:

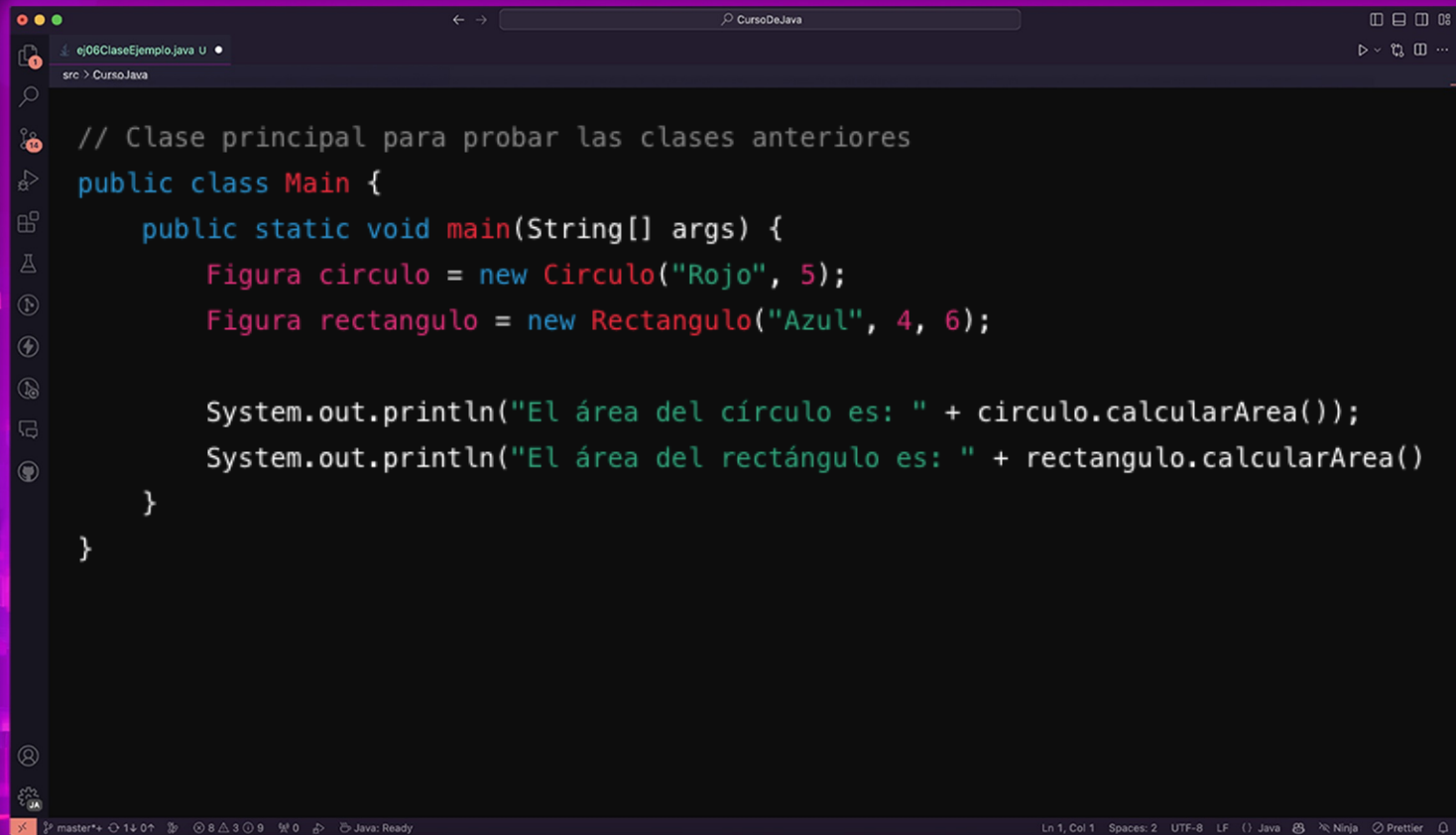
```
// Subclase Rectangulo
class Rectangulo extends Figura {
    double ancho, alto;

    public Rectangulo(String color, double ancho, double alto) {
        super(color);
        this.ancho = ancho;
        this.alto = alto;
    }

    @Override
    double calcularArea() {
        return ancho * alto;
    }
}
```

The status bar at the bottom indicates the current line and column as "Ln 1, Col 1", the file encoding as "UTF-8", and the line ending as "LF". It also shows the Java version as "Java: Ready" and the formatter as "Prettier".

Clase Abstracta Básica

A screenshot of a code editor window with a dark theme. The editor shows a Java file named 'ej06ClaseEjemplo.java'. The code is a 'Main' class with a 'main' method. Inside the 'main' method, two objects are created: 'circulo' of type 'Circulo' with color 'Rojo' and radius 5, and 'rectangulo' of type 'Rectangulo' with color 'Azul', width 4, and height 6. Then, the 'calcularArea()' method is called on both objects, and the results are printed to the console. The editor has a sidebar on the left with various icons, a top bar with a search icon and the text 'CursoDeJava', and a bottom status bar showing 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'Java', and 'Prettier'.

```
// Clase principal para probar las clases anteriores
public class Main {
    public static void main(String[] args) {
        Figura circulo = new Circulo("Rojo", 5);
        Figura rectangulo = new Rectangulo("Azul", 4, 6);

        System.out.println("El área del círculo es: " + circulo.calcularArea());
        System.out.println("El área del rectángulo es: " + rectangulo.calcularArea());
    }
}
```