

Programación I

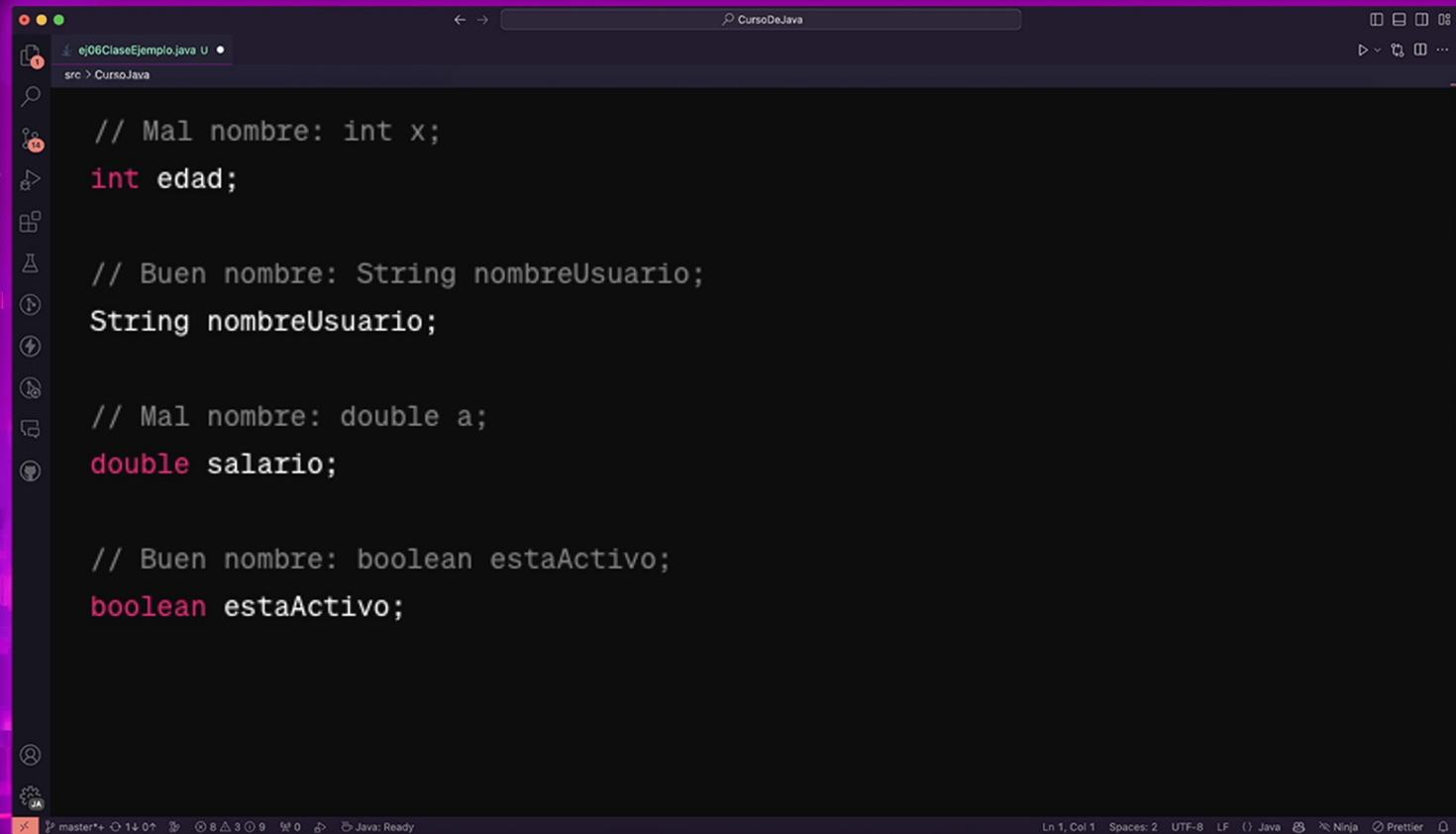
Buenas prácticas de programación

Introducción al Lenguaje Java

Nombres descriptivos

Los nombres descriptivos hacen que el código sea más legible, comprensible y mantenible para ti y otros desarrolladores que trabajen en el proyecto.

Nombres descriptivos



The image shows a code editor window with a dark theme. The title bar at the top says "CursoDeJava". The editor contains four lines of Java code, each with a comment above it. The comments are in Spanish and describe the variables. The code is as follows:

```
// Mal nombre: int x;  
int edad;  
  
// Buen nombre: String nombreUsuario;  
String nombreUsuario;  
  
// Mal nombre: double a;  
double salario;  
  
// Buen nombre: boolean estaActivo;  
boolean estaActivo;
```

The status bar at the bottom shows "master+ 140", "8 3 9", "0", "Java: Ready", "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "Java", "Ninja", "Prettier", and a search icon.

Comentarios claros y concisos

Los comentarios bien escritos ayudan a los desarrolladores a comprender rápidamente el código y facilitan el mantenimiento y la depuración en el futuro.

Comentarios claros y concisos



```
// Método para calcular el área de un círculo.
public double calcularArea(double radio) {
    return Math.PI * radio * radio;
}

// Verificar si el número es positivo o negativo.
if (numero >= 0) {
    // El número es positivo.
    System.out.println("El número es positivo.");
} else {
    // El número es negativo.
    System.out.println("El número es negativo.");
}

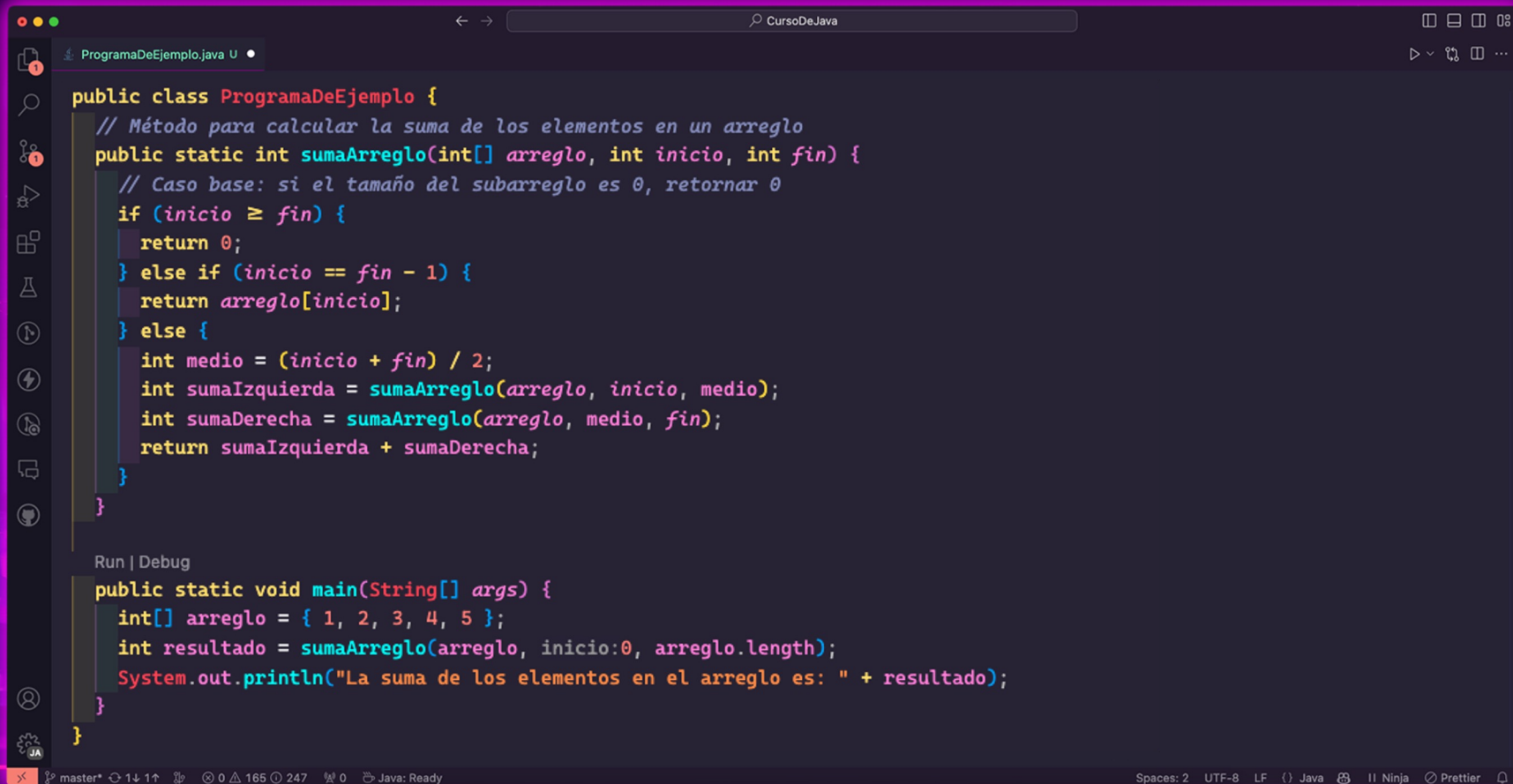
// TODO: Implementar la lógica para calcular el impuesto sobre la renta.
```

The image shows a code editor window with a dark theme. The title bar indicates the file is 'ej06ClaseEjemplo.java' and the project is 'CursoDeJava'. The code is written in Java and includes several comments. The first comment is a single-line block comment describing the purpose of the 'calcularArea' method. The second is a single-line block comment above an 'if' statement. Inside the 'if' block, there are two more single-line block comments. The 'else' block also contains a single-line block comment. At the bottom, there is a single-line block comment starting with 'TODO:'. The code is formatted with standard Java syntax highlighting: keywords in blue, identifiers in red, and literals in green. The editor's interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, and Extensions. The status bar at the bottom shows 'master' branch, 140 commits, 8 files, 3 folders, 9 lines, 0 characters, and indicates the language is 'Java: Ready'. It also shows the current cursor position as 'Ln 1, Col 1' and the encoding as 'UTF-8'.

Divide y vencerás

Consiste en dividir un problema grande en problemas más pequeños y manejables, resolver cada uno de manera independiente y luego combinar las soluciones para obtener el resultado final. Esto hace que el proceso de resolución de problemas sea más manejable y facilita la comprensión y el mantenimiento del código.

Divide y vencerás



The image shows a code editor window with a dark theme. The title bar at the top indicates the file is 'ProgramaDeEjemplo.java U'. The editor contains a Java class named 'ProgramaDeEjemplo' with two methods. The first method, 'sumaArreglo', is a recursive function that calculates the sum of elements in an array from a given start index to an end index. It uses a divide-and-conquer approach: if the range is empty, it returns 0; if it's a single element, it returns that element; otherwise, it splits the array in half, recursively sums each half, and returns the total. The second method, 'main', initializes an array with the values [1, 2, 3, 4, 5] and calls 'sumaArreglo' to calculate the sum, printing the result to the console. The status bar at the bottom shows the current branch is 'master', there are 1 commit, 165 lines of code, and 247 characters. It also indicates the Java IDE is ready.

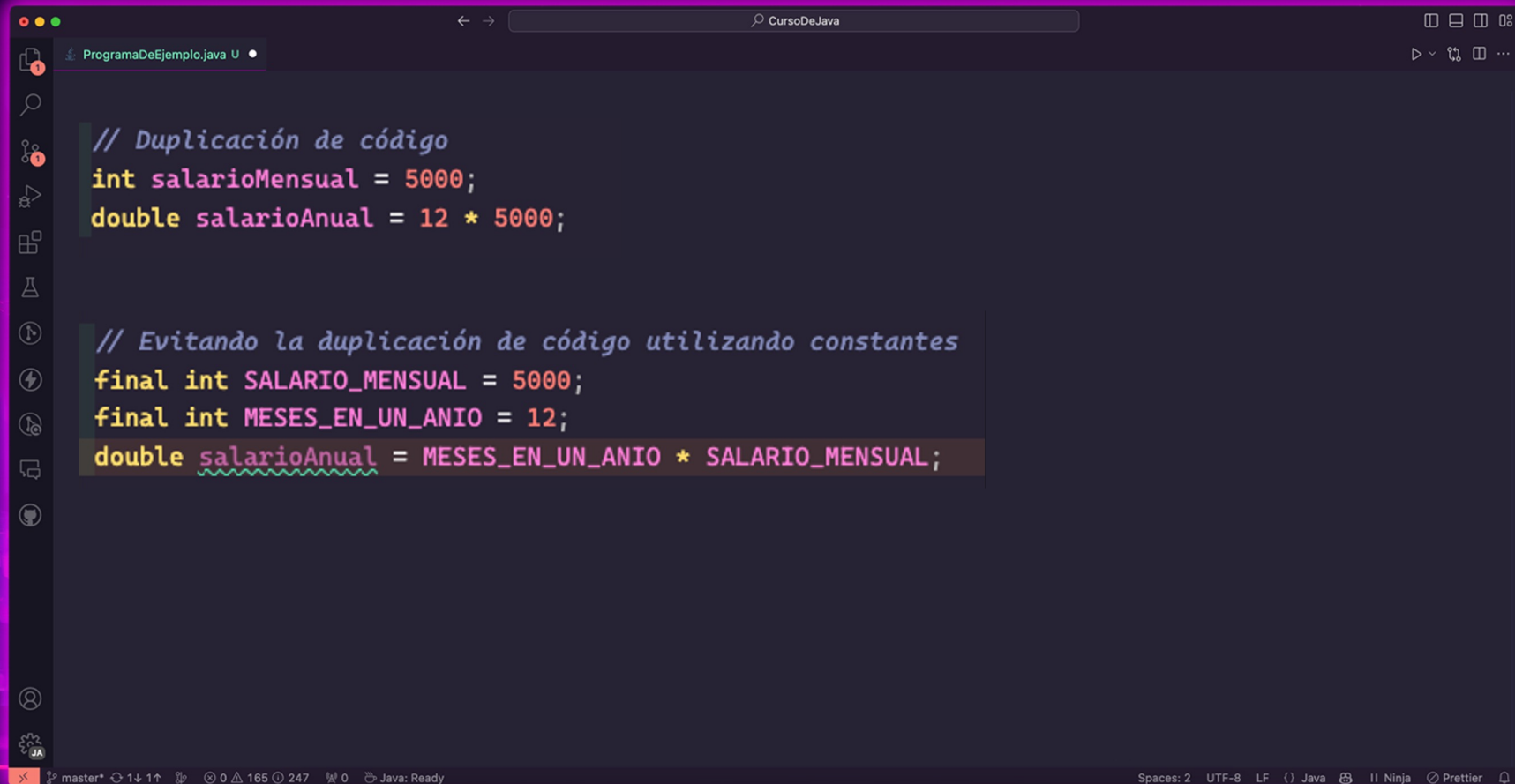
```
public class ProgramaDeEjemplo {  
    // Método para calcular la suma de los elementos en un arreglo  
    public static int sumaArreglo(int[] arreglo, int inicio, int fin) {  
        // Caso base: si el tamaño del subarreglo es 0, retornar 0  
        if (inicio ≥ fin) {  
            return 0;  
        } else if (inicio == fin - 1) {  
            return arreglo[inicio];  
        } else {  
            int medio = (inicio + fin) / 2;  
            int sumaIzquierda = sumaArreglo(arreglo, inicio, medio);  
            int sumaDerecha = sumaArreglo(arreglo, medio, fin);  
            return sumaIzquierda + sumaDerecha;  
        }  
    }  
  
    Run | Debug  
    public static void main(String[] args) {  
        int[] arreglo = { 1, 2, 3, 4, 5 };  
        int resultado = sumaArreglo(arreglo, inicio:0, arreglo.length);  
        System.out.println("La suma de los elementos en el arreglo es: " + resultado);  
    }  
}
```

master* 1 commit 165 lines 247 characters Java: Ready Spaces: 2 UTF-8 LF () Java II Ninja Prettier

Evita la duplicación de código

Evitar la duplicación de código es una buena práctica de programación que implica escribir código de manera que evite la repetición innecesaria de fragmentos de código idénticos o similares. Esto no solo hace que el código sea más claro y fácil de mantener, sino que también reduce la posibilidad de errores y facilita las futuras modificaciones.

Evita la duplicación de código



```
ProgramaDeEjemplo.java U •  
  
// Duplicación de código  
int salarioMensual = 5000;  
double salarioAnual = 12 * 5000;  
  
// Evitando la duplicación de código utilizando constantes  
final int SALARIO_MENSUAL = 5000;  
final int MESES_EN_UN_ANIO = 12;  
double salarioAnual = MESES_EN_UN_ANIO * SALARIO_MENSUAL;
```

master* 1↓ 1↑ 0 165 247 0 Java: Ready Spaces: 2 UTF-8 LF () Java II Ninja Prettier