Documentación del Uso del IDE

Documentación del uso del IDE - Eclipse

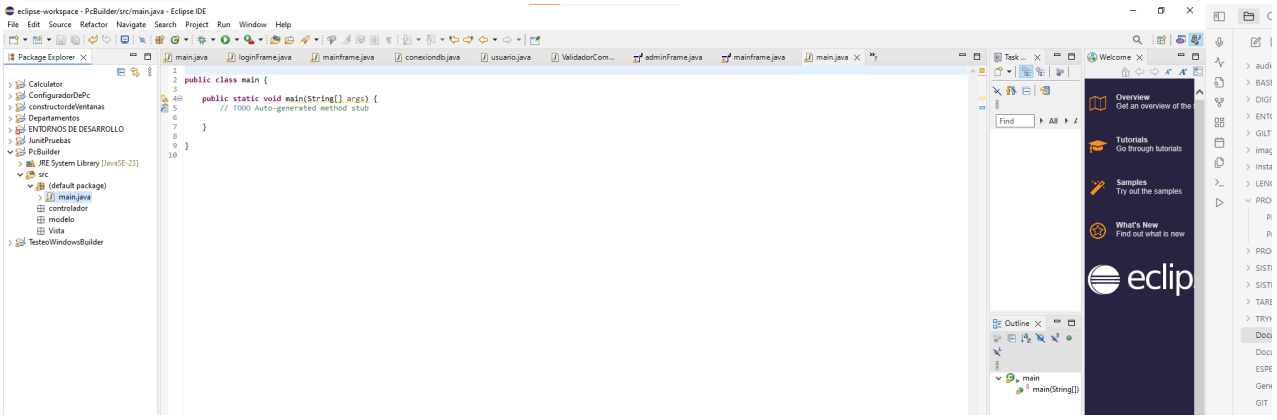
1. CONFIGURACIÓN INICIAL

Instalación de Eclipse

1. Descargar Eclipse IDE for Java Developers
2. Instalar Java JDK 8 o superior
3. Configurar JAVA_HOME en variables de entorno

Creación del proyecto

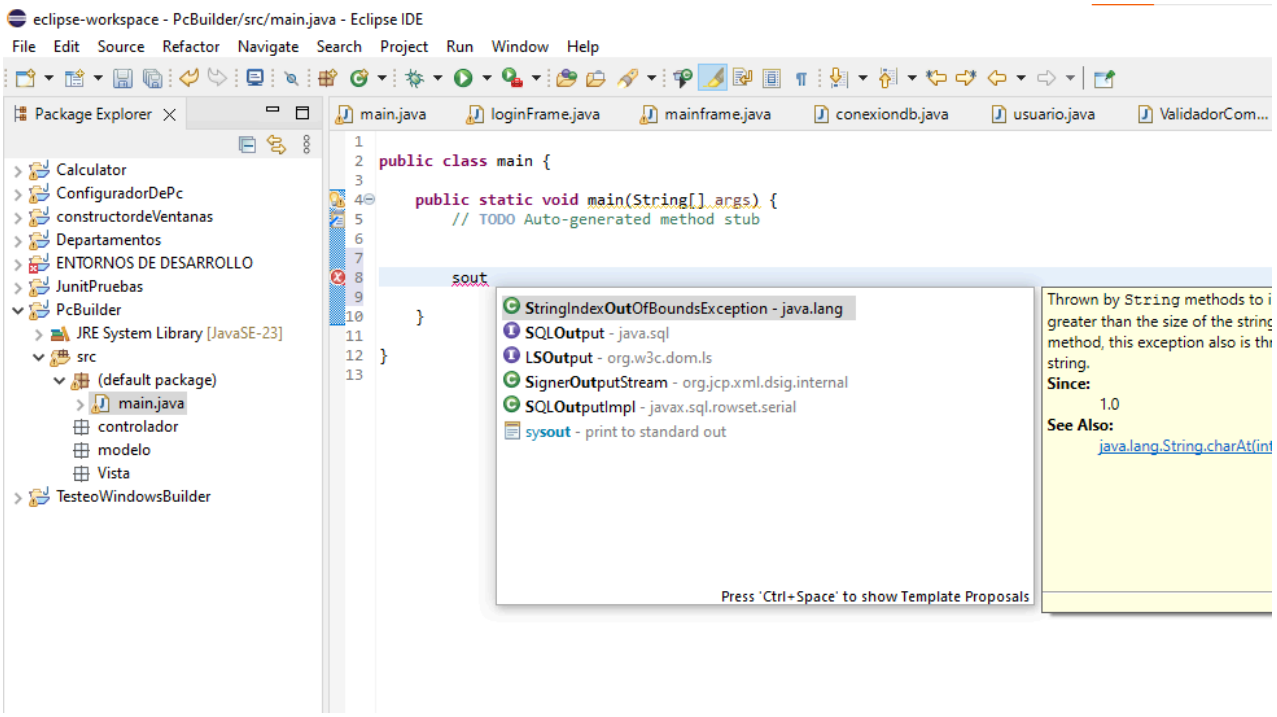
1. File → New → Java Project
2. Nombre: PCBuilder
3. Crear estructura de paquetes:
   - `modelo`
   - `vista`
   - `controlador`



2. FUNCIONALIDADES UTILIZADAS

Edición de código

- **Autocompletado:** Ctrl + Space
  -



- **Formateo automático:** Ctrl + Shift + F

-

```java
package controlador;

import modelo.usuario;
import vista.adminFrame;

/**
 * Controlador para la vista del panel de administración.
 *
 * <p>
 * Este controlador se encarga de inicializar la interfaz de administrador y
 * pasarle el usuario autenticado.
 * </p>
 *
 * @author Abdelghaffar EL AKHDAR
 * @version 1.6
 * @since 2025
 */
public class adminController {
    private adminFrame frame;
    private usuario admin;

    /**
     * Constructor que inicializa el frame de administrador.
     *
     * @param usuario Usuario autenticado con rol de administrador.
     */
    public adminController(usuario usuario) {
        this.admin = usuario;
        this.frame = new adminFrame();
        frame.setVisible(true);
    }
}

```

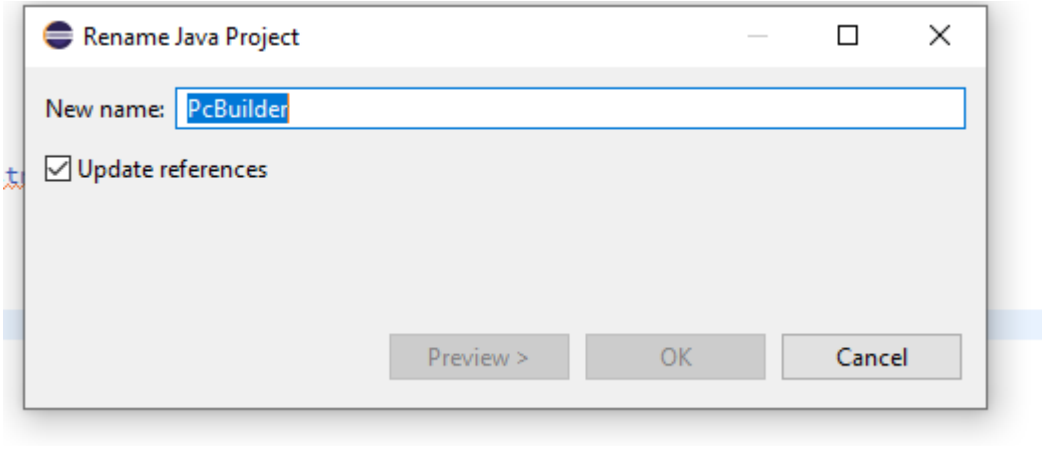
- **Organizar imports:** Ctrl + Shift + O

```java
package controlador;

import modelo.usuario;
import vista.adminFrame;

/**
 * Controlador para la vista del panel de administración.
 *
 * <p>
 * Este controlador se encarga de inicializar la interfaz de administrador y
 * pasarle el usuario autenticado.
 * </p>
 *
 * @author Abdelghaffar EL AKHDAR
```

- **Renombrar:** Alt + Shift + R

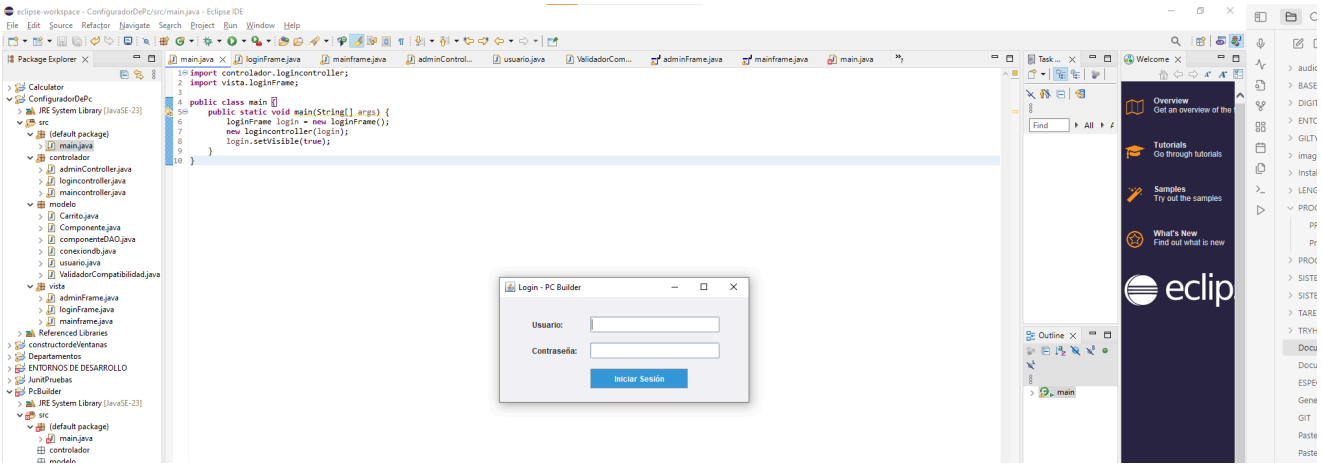


Asistentes de IA

- **GitHub Copilot** (si está instalado)
- **Sugerencias de Eclipse:** Quick Fix (Ctrl + 1)
- **Generación automática:**
  - Getters/Setters: Alt + Shift + S → Generate
  - Constructor: Alt + Shift + S → Generate Constructor

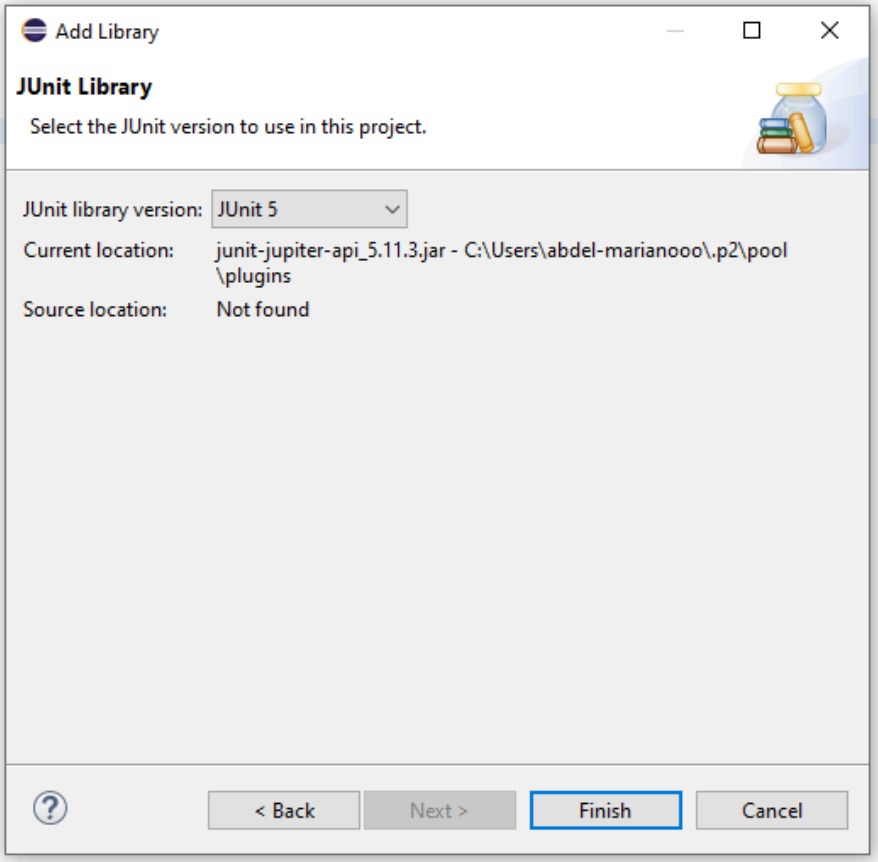
Pruebas

- **Ejecución:** Click derecho → Run As → Java Application

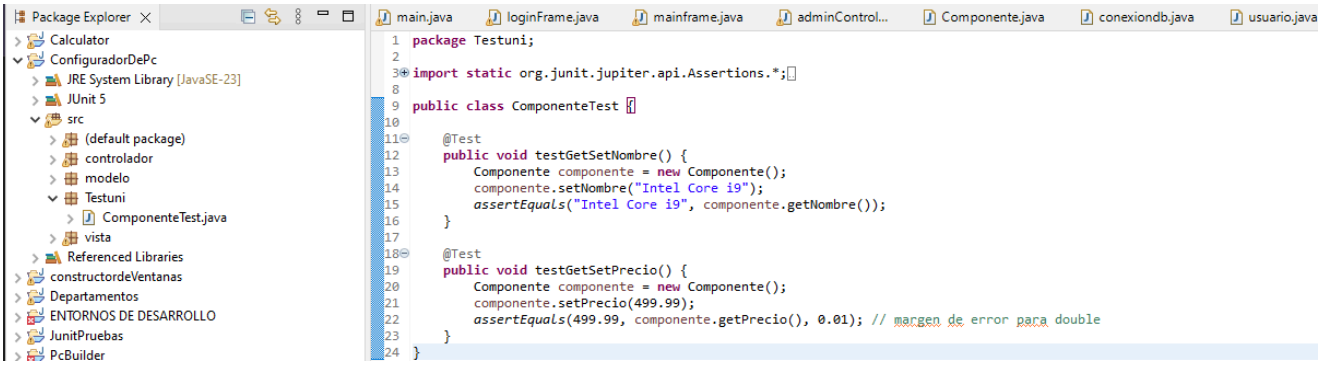


- **Pruebas unitarias:** JUnit (no implementado en este proyecto)

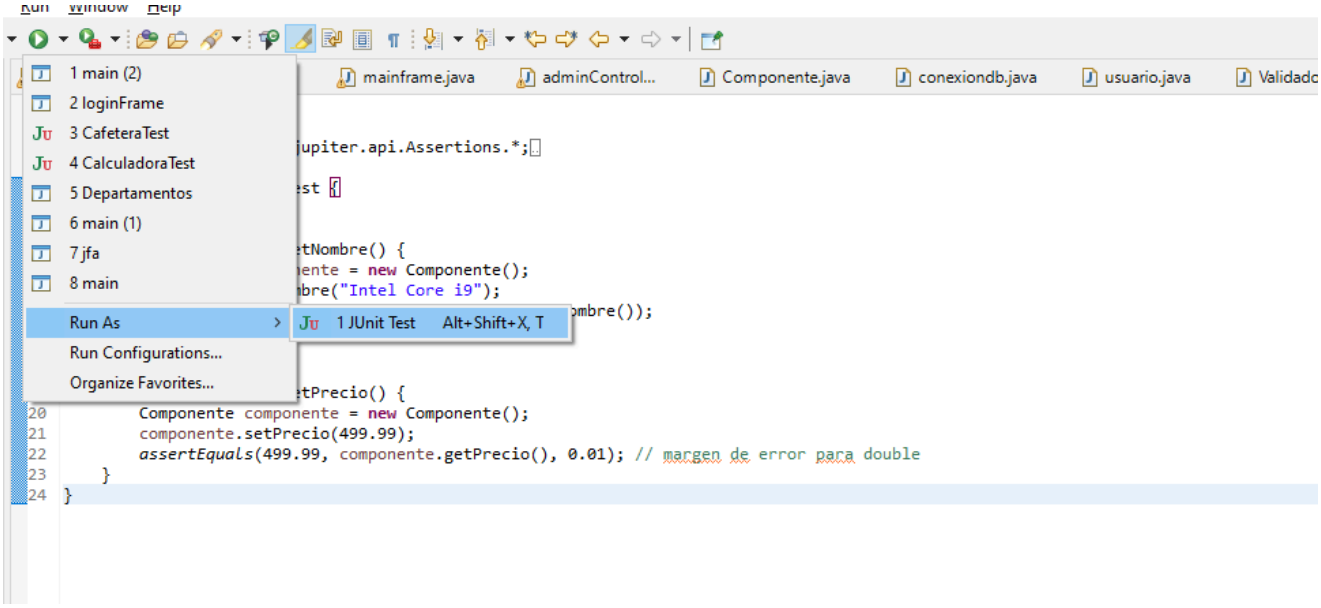
Primero añadimos Junit al Build Path
Haz clic derecho sobre tu proyecto → `Build Path` → `Add Libraries...`



-Creamos un paquete test y una clase Test

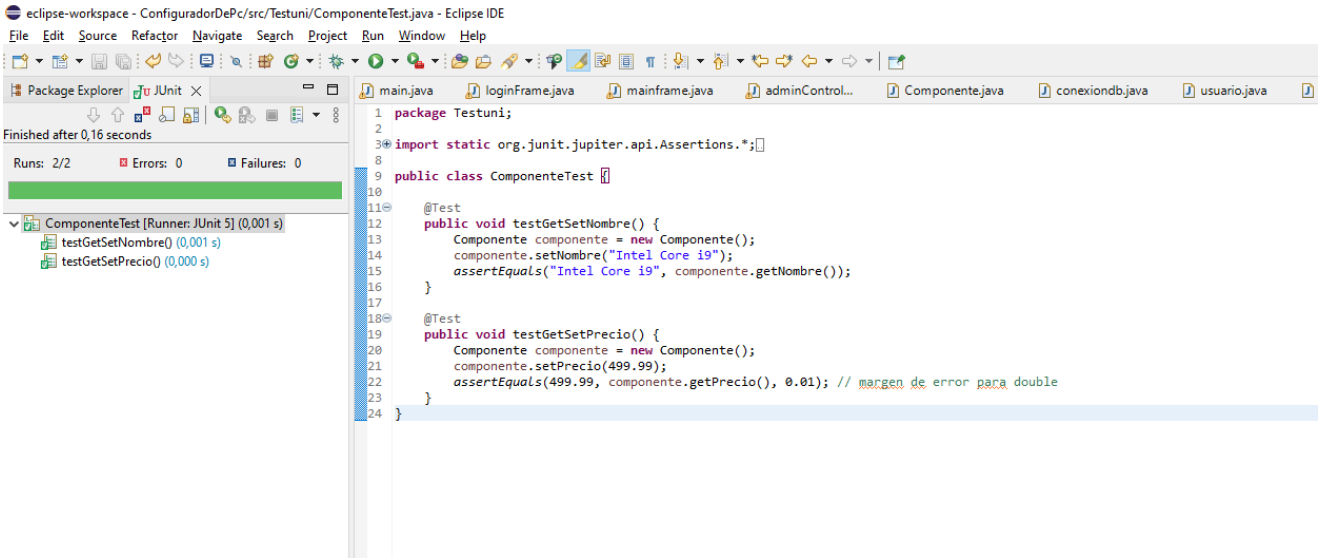


-Ahora vamos a correr el Test



-Resultados

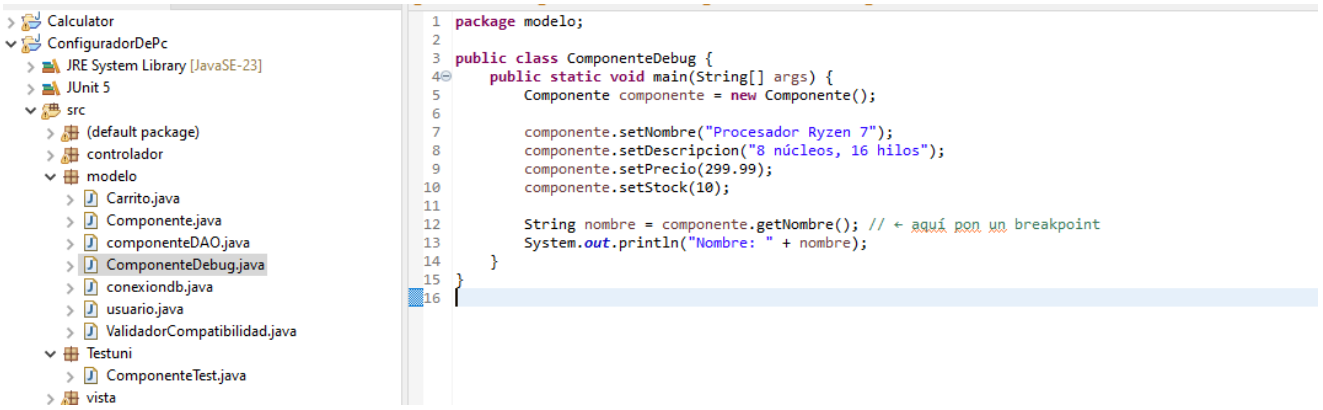
y vemos que funcionan pues eh echo esto con casi todo pero da toda la pereza hacer capturas



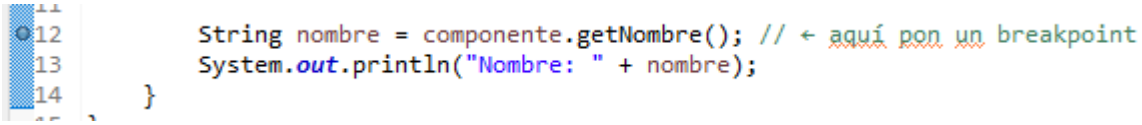
Depuración

1. Establecer breakpoints: Doble click en el margen izquierdo
2. Debug As → Java Application
3. Ventanas de depuración:
   - Variables: Ver valores actuales
   - Expressions: Evaluar expresiones
   - Step Over (F6): Siguiente línea
   - Step Into (F5): Entrar en método

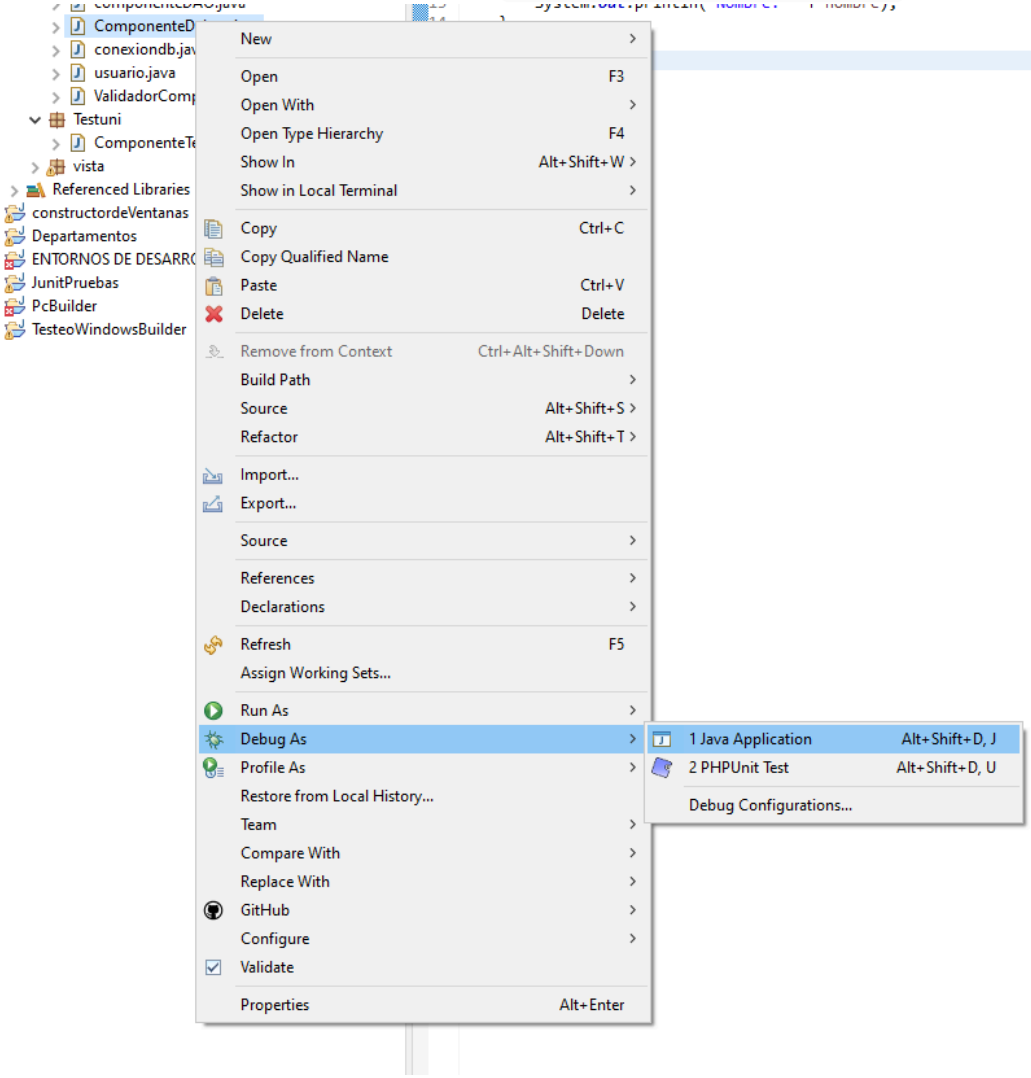
Haber para hacer la depuración voy a crear un main en el que invocare la clase componente



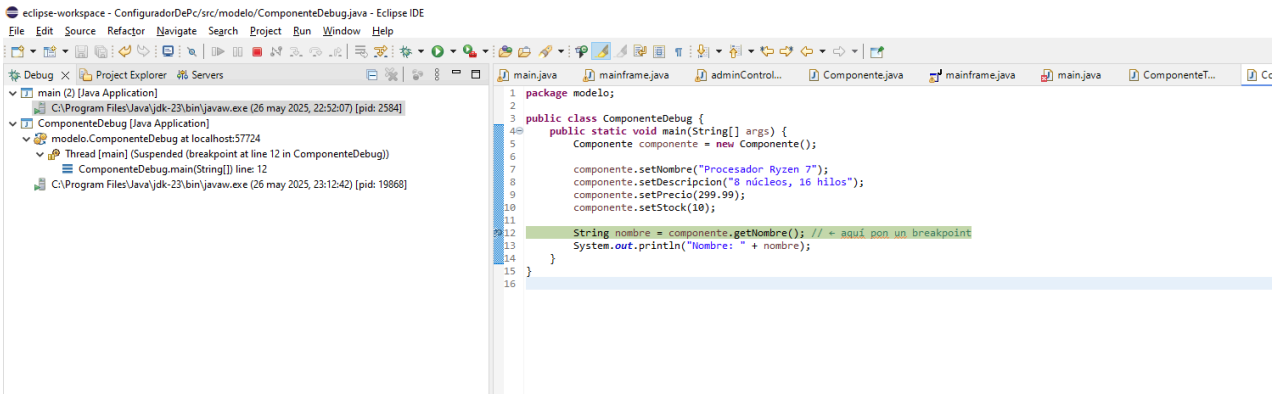
- Hay que hacer **doble clic en la línea** `String nombre = componente.getNombre();` para poner un **breakpoint** cuando se pone el breakpoint sale una bola azul detras del numero de la linea.



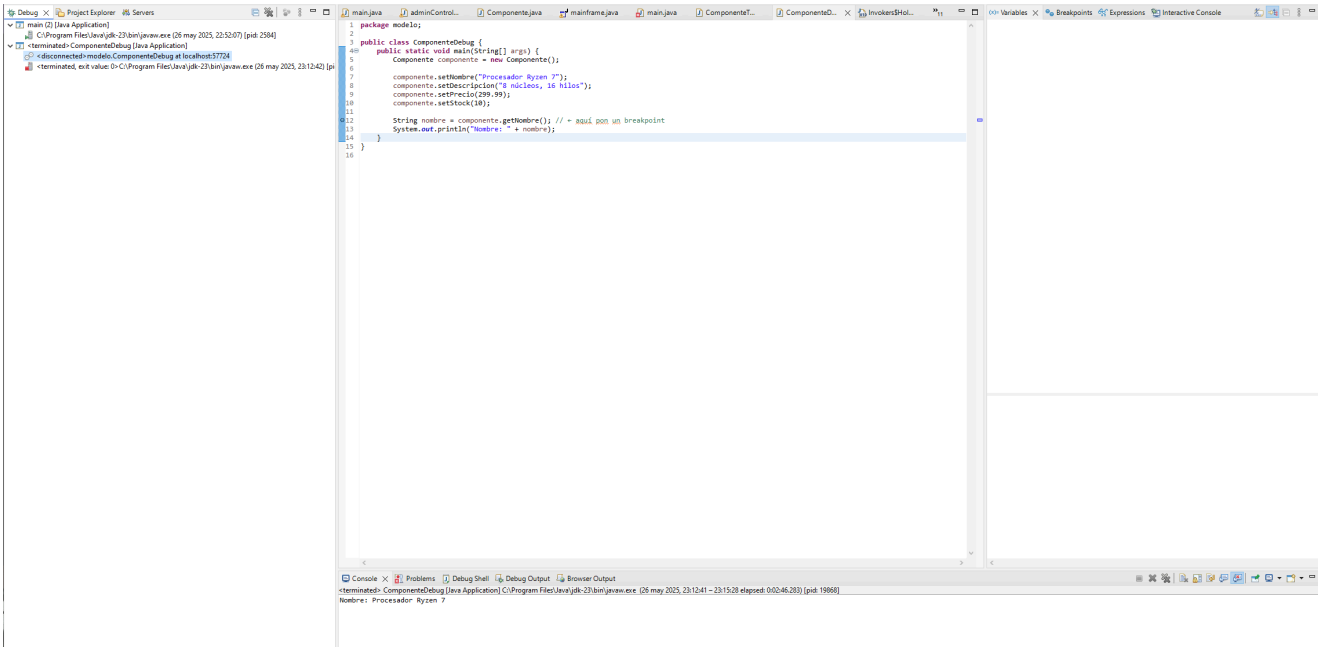
- Luego hacer clic derecho en la clase → `Debug As → Java Application`



- Eclipse se detendrá ahí → podrás ver el estado completo del objeto `componente` (nombre, precio, stock, etc.).



- Usa `F6` y `F5` como mencionamos antes para avanzar o entrar en los métodos.



Refactorización

```
vamos a tomar como ejemplo la clase Componente otra vez
```

- **Extraer método:** Alt + Shift + M
- **Mover clases:** Arrastrar entre paquetes
- **Cambiar firma de método:** Alt + Shift + C

package modelo;

/**

- Clase que representa un componente de PC en el sistema.
- Almacena toda la información necesaria de cada componente
- incluyendo características técnicas para validar compatibilidad.
-
- @author Abdel
- @version 1.7
  */
  public class Componente {
  private int id;

private String nombre;
private String descripcion;
private double precio;
private int stock;
private String tipo;
private String socket;
private String tipoRam;
private Integer potenciaRecomendada;
private String tamanoPlaca;

```
// 🔧 Constructor recomendado
public Componente(int id, String nombre, String descripcion, double precio, int
stock, String tipo) {
    this.id = id;
    this.nombre = nombre;
    this.descripcion = descripcion;
    this.precio = precio;
    this.stock = stock;
    this.tipo = tipo;
}

// ✅ Getters y setters con nombres corregidos
public int getId() { return id; }
public void setId(int id) { this.id = id; }

public String getNombre() { return nombre; }
public void setNombre(String nombre) { this.nombre = nombre; }

public String getDescripcion() { return descripcion; }
public void setDescripcion(String descripcion) { this.descripcion = descripcion; }

public double getPrecio() { return precio; }
public void setPrecio(double precio) { this.precio = precio; }

public int getStock() { return stock; }
public void setStock(int stock) { this.stock = stock; }

public String getTipo() { return tipo; }
public void setTipo(String tipo) { this.tipo = tipo; }

public String getSocket() { return socket; }
public void setSocket(String socket) { this.socket = socket; }

public String getTipoRam() { return tipoRam; }
public void setTipoRam(String tipoRam) { this.tipoRam = tipoRam; }

public Integer getPotenciaRecomendada() { return potenciaRecomendada; }
public void setPotenciaRecomendada(Integer potenciaRecomendada) {
this.potenciaRecomendada = potenciaRecomendada; }
```

```
    public String getTamanoPlaca() { return tamanoPlaca; }
    public void setTamanoPlaca(String tamanoPlaca) { this.tamanoPlaca = tamanoPlaca; }

    // 🧪 Opcional: método toString() para depuración
    @Override
    public String toString() {
        return "Componente{" +
                "id=" + id +
                ", nombre='" + nombre + '\'' +
                ", descripcion='" + descripcion + '\'' +
                ", precio=" + precio +
                ", stock=" + stock +
                ", tipo='" + tipo + '\'' +
                ", socket='" + socket + '\'' +
                ", tipoRam='" + tipoRam + '\'' +
                ", potenciaRecomendada=" + potenciaRecomendada +
                ", tamanoPlaca='" + tamanoPlaca + '\'' +
                '}';
    }
}
```

Refactorización aplicada a la clase `Componente`:

Se corrigieron los nombres de los métodos getters y setters para seguir la convención estándar de Java (por ejemplo, `gettipo()` se renombró a `getTipo()`). Se añadió un constructor para facilitar la creación de instancias y un método `toString()` útil durante la depuración. Estas mejoras no cambian la funcionalidad del código, pero lo hacen más legible, mantenible y preparado para futuras ampliaciones.

VENTAJAS DEL IDE

1. 🔍 Detección de errores en tiempo real

- El IDE subraya errores de sintaxis (rojo) o advertencias (amarillo) mientras escribes el código.
- Por ejemplo, si escribes `gettamanoPlaca()` sin definir el método, te lo marcará al momento.
- Así evitas compilar con errores y pierdes menos tiempo buscando fallos.

---

2. 🧠 Autocompletado inteligente

- Mientras escribes código, el IDE te sugiere métodos, atributos y clases automáticamente.
- Por ejemplo: escribes `comp.` y te sugiere `getId()`, `getTipo()`, `setPrecio()`, etc.
- Ahorra tiempo, evita errores y ayuda cuando no recuerdas exactamente el nombre de algo.

---

3. 🚀 Navegación rápida entre clases y métodos

- Puedes hacer **Ctrl + click** en cualquier clase, método o variable para ir a su definición.
- Útil cuando trabajas con muchas clases como `Componente`, `ValidadorCompatibilidad`, etc.
- También puedes buscar archivos o clases con `Ctrl + Shift + R` (Eclipse) o `Double Shift` (IntelliJ).

---

4. 🌱 Integración con Git

- Puedes conectar el proyecto a Git directamente desde el IDE.
- Permite hacer commits, ver el historial, comparar versiones y hacer push/pull sin usar la terminal.
- Ideal para control de versiones en equipo o tener copias de seguridad de tus avances.

---

5. ⚙️ Gestión de librerías (como MySQL Connector)

- Puedes añadir fácilmente bibliotecas externas (JAR) como el **MySQL Connector**.
- El IDE te permite añadirlas al classpath del proyecto sin complicaciones.
- Por ejemplo: conectar con la base de datos para cargar los `Componente` desde MySQL.

📜 ¿Cómo mostrarlo en clase?

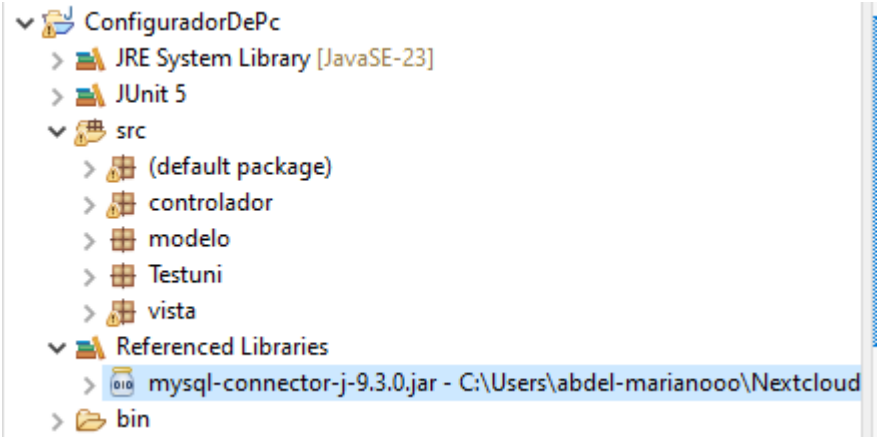
- Haz capturas de pantalla reales mostrando cada ventaja en acción con tu proyecto.
- Por ejemplo:
  - Un error subrayado en rojo mientras programas.
  - El autocompletado mostrando métodos de `Componente`.
  - El explorador de clases con `Componente.java`, `adminFrame.java`, etc.
  - El panel de Git mostrando los cambios.
  - El proyecto mostrando que incluye `mysql-connector-java.jar`.

4. CONFIGURACIÓN ESPECÍFICA DEL PROYECTO

Agregar MySQL Connector

1. Click derecho en proyecto → Build Path → Configure Build Path
2. Libraries → Add External JARs

3. Seleccionar mysql-connector-java-9.3.0.jar



Configuración de codificación

- Window → Preferences → General → Workspace
- Text file encoding: UTF-8