Refactorización de Código en Eclipse

Introducción

La refactorización de código es una práctica esencial en el desarrollo de software que permite mejorar la calidad del código sin alterar su funcionalidad. En el entorno de desarrollo Eclipse, existen múltiples herramientas y opciones que facilitan esta tarea. En este trabajo, se abordarán cuatro técnicas comunes de refactorización: Cambio de Nombre, Extraer Interfaz, Encapsular Campo y Extraer Método.

Objetivos

- Identificar los patrones de refactorización más usuales.
- Aplicar las técnicas de refactorización en el entorno de desarrollo Eclipse.
- Realizar pruebas asociadas a la refactorización.

Técnicas de Refactorización en Eclipse

1. Cambio de Nombre (Rename)

El cambio de nombre permite mejorar la claridad y coherencia del código. En Eclipse, se puede acceder a esta opción haciendo clic derecho sobre la variable, método o clase, seleccionando 'Refactor' y luego 'Rename'. Esta práctica es especialmente útil en proyectos colaborativos, ya que facilita la comprensión del código por otros miembros del equipo.

Antes:

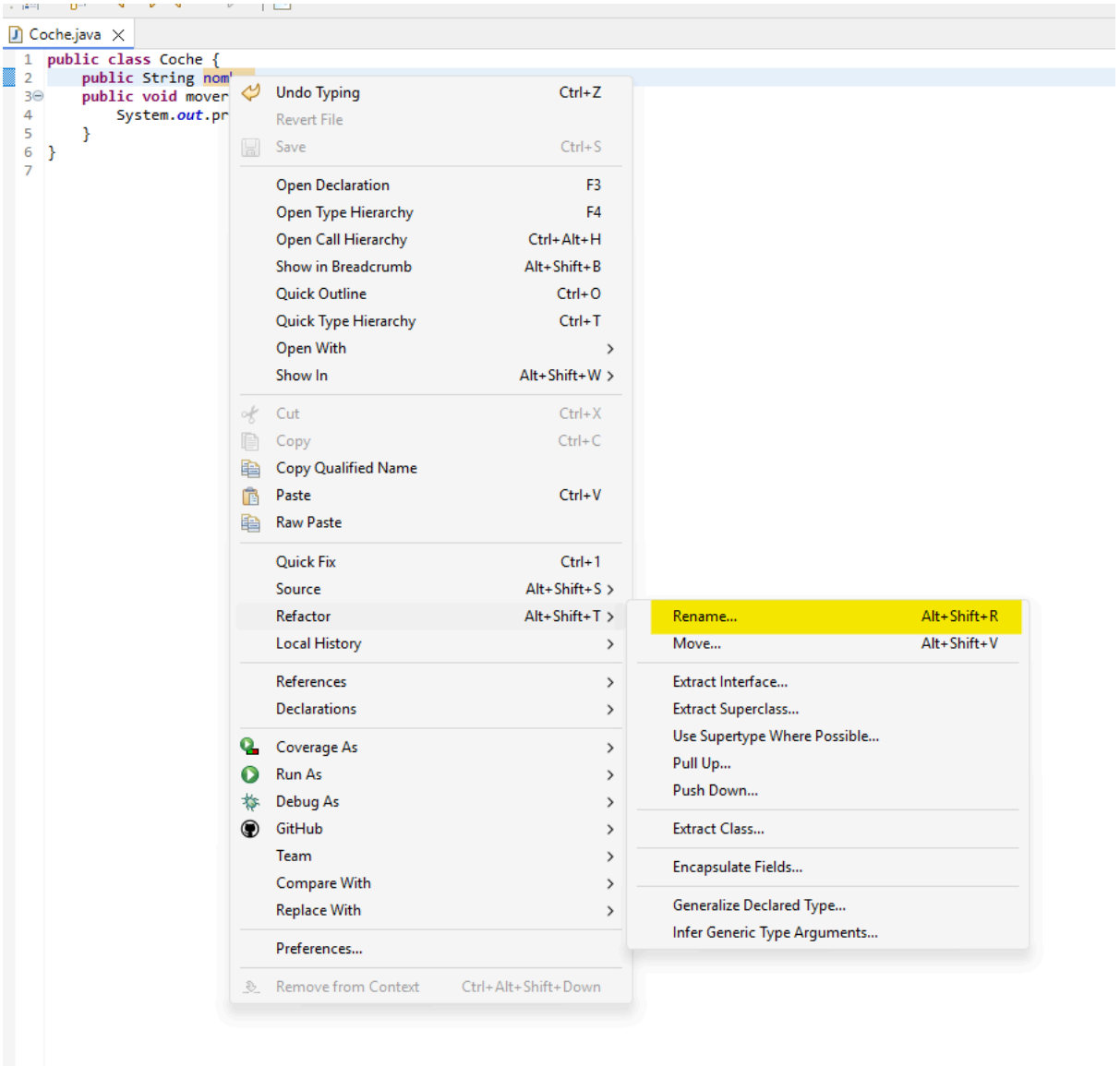
```
public class Coche {
    public String nombre;
    public void mover() {
        System.out.println("El coche se mueve");
    }
}
```

*Coche.java

```java
public class Coche {
    public String nombre;
    public void mover() {
        System.out.println("El coche se mueve");
    }
}
```

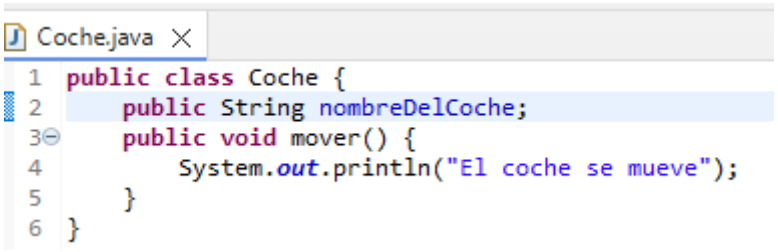
Proceso:

1. Seleccionar la variable nombre en Eclipse.
2. Hacer clic derecho y seleccionar Refactor > Rename o usar el atajo Shift + Alt + R.
3. Introducir el nuevo nombre nombreDelCoche.
4. Confirmar el cambio presionando Enter.



Después:

```
public class Coche {
    public String nombreDelCoche;
    public void mover() {
        System.out.println("El coche se mueve");
    }
}
```



Ventajas:

- Mejora la legibilidad del código.
- Reduce la posibilidad de errores.
- Facilita el mantenimiento del software.
- Ayuda en la comprensión del código en equipos de desarrollo colaborativo.

2. Extraer Interfaz (Extract Interface) y Encapsular Campo (Encapsulate Field)

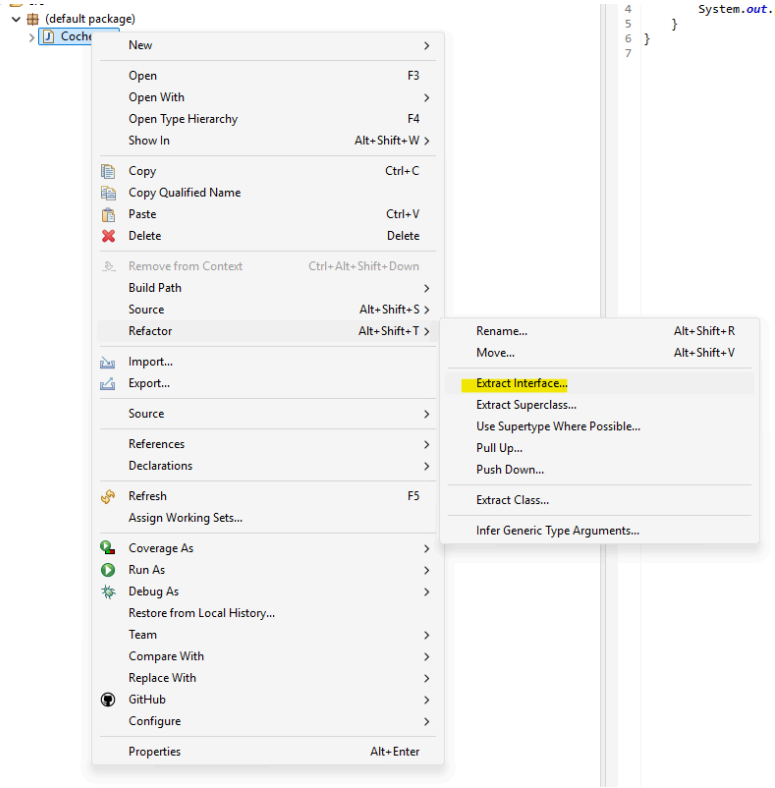
Esta técnica permite mejorar la modularidad y flexibilidad del código al crear una interfaz a partir de una clase existente. El encapsulamiento de campos mejora la seguridad del código al proteger el acceso directo a las variables. En Eclipse, se puede utilizar la opción 'Refactor > Extract Interface' y luego encapsular el campo con 'Source > Generate Getters and Setters'.

Antes:

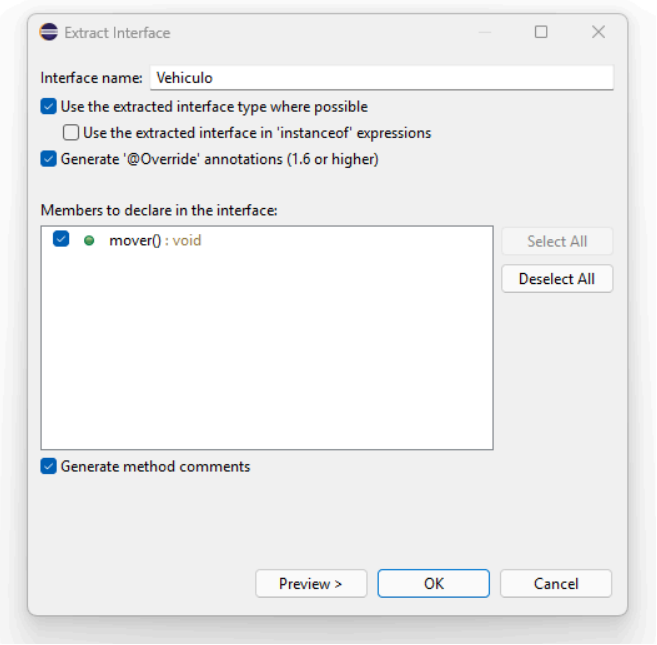
```
public class Coche {
    public String nombreDelCoche;
    public void mover() {
        System.out.println("El coche se mueve");
    }
}
```

Proceso:

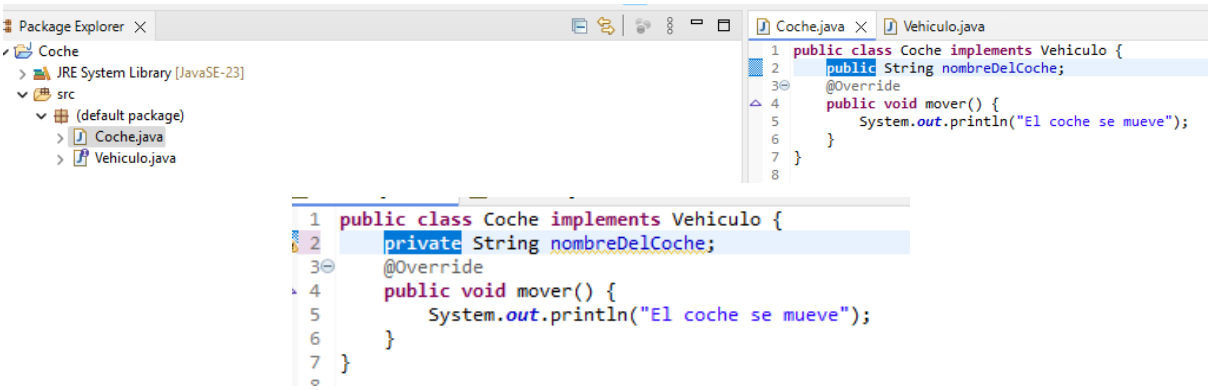
1. Seleccionar la clase Coche en el Package Explorer.
2. Hacer clic derecho y seleccionar Refactor > Extract Interface.



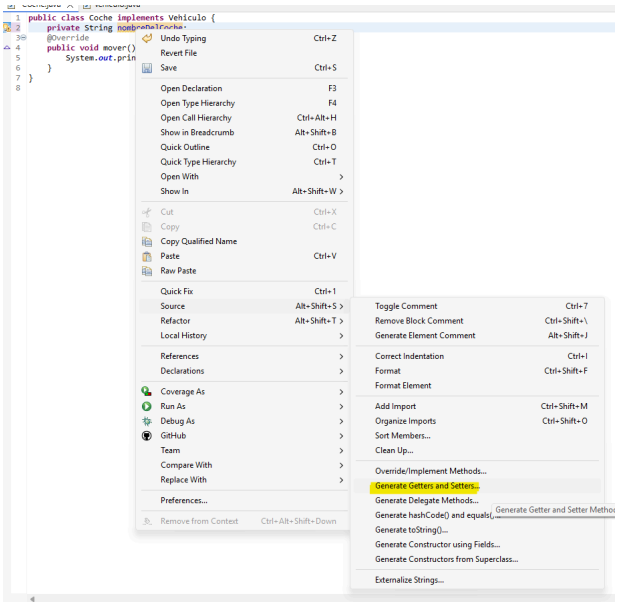
3. Escribir el nombre de la interfaz Vehiculo.
4. Seleccionar el método mover() en la lista de métodos disponibles.
5. Hacer clic en OK para aplicar los cambios.

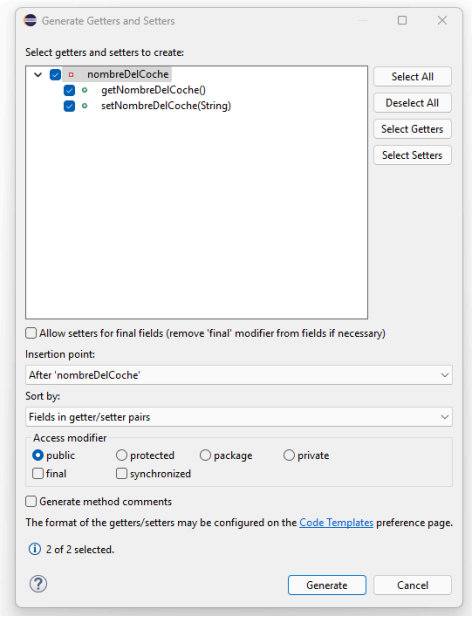


6. Encapsular el campo: Cambiar la visibilidad de nombreDelCoche a private.



7. Generar los getters y setters con Source > Generate Getters and Setters....

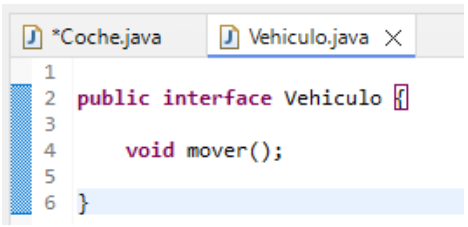




Después:

1. Vehículo.java:

```
public interface Vehiculo {
    void mover();
}
```



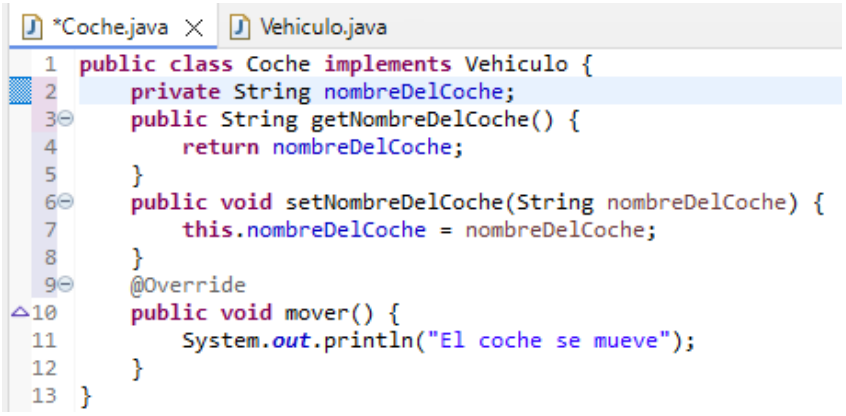
2. Coche.java:

```
public class Coche implements Vehiculo {
    private String nombreDelCoche;

    @Override
    public void mover() {
        System.out.println("El coche se mueve");
    }

    public String getNombreDelCoche() {
        return nombreDelCoche;
    }

    public void setNombreDelCoche(String nombreDelCoche) {
        this.nombreDelCoche = nombreDelCoche;
    }
}
```



Ventajas de extraer interfaz:

- Mejora la modularidad del código.
- Facilita la ampliación y reutilización de componentes.
- Permite la implementación de múltiples clases con comportamientos comunes.
- Fomenta un diseño orientado a interfaces, útil para patrones de diseño como Dependency Injection.

Ventajas de encapsular campo:

- Mejora la seguridad y control sobre los datos.
- Facilita la depuración y el mantenimiento.
- Permite aplicar validaciones al acceder o modificar campos.
- Asegura la consistencia de los datos a través de métodos controlados.

3. Extraer Método (Extract Method)

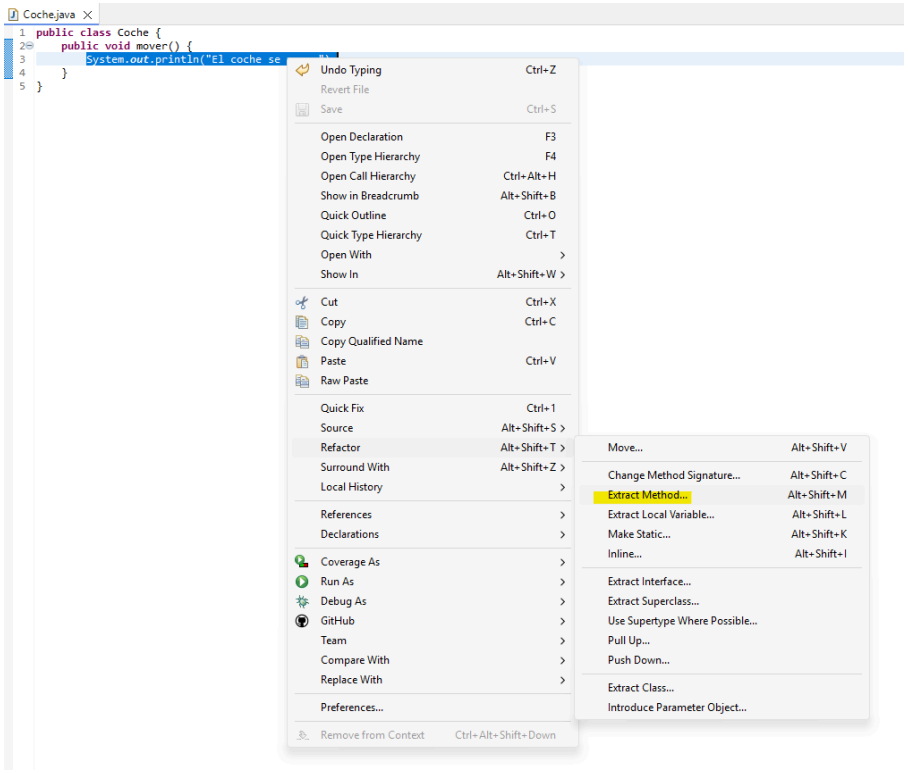
La extracción de métodos permite dividir grandes bloques de código en métodos más pequeños y manejables, mejorando la legibilidad y mantenibilidad del código. Esto es especialmente útil cuando se tiene lógica repetitiva o cuando se desea aplicar el principio de responsabilidad única (Single Responsibility Principle).

Antes:

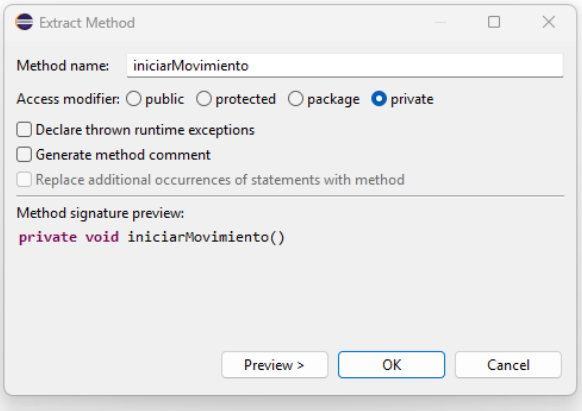
```
public class Coche {
    public void mover() {
        System.out.println("El coche se mueve");
    }
}
```

Proceso:

1. Seleccionar la línea de código System.out.println("El coche se mueve");.
2. Hacer clic derecho y seleccionar Refactor > Extract Method.



3. Introducir el nombre del nuevo método iniciarMovimiento.
4. Confirmar el cambio haciendo clic en OK.



Después:

```
public class Coche {
    public void mover() {
        iniciarMovimiento();
    }

    private void iniciarMovimiento() {
        System.out.println("El coche se mueve");
    }
}
```

```java
public class Coche {
    public void mover() {
        iniciarMovimiento();
    }

    private void iniciarMovimiento() {
        System.out.println("El coche se mueve");
    }
}
```

Ventajas:

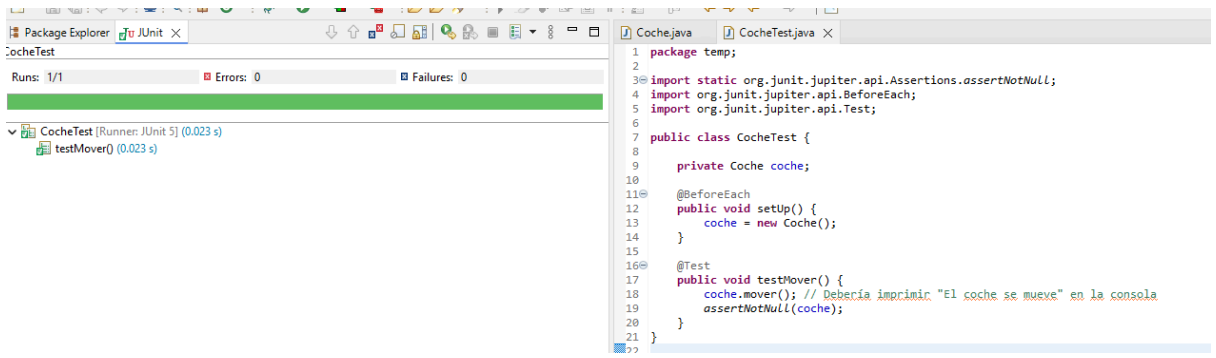
- Mejora la legibilidad del código.
- Reduce la duplicación de código.
- Facilita la realización de pruebas unitarias.
- Permite reutilizar la lógica en diferentes partes del programa.

Pruebas Asociadas a la Refactorización

Es fundamental realizar pruebas unitarias para garantizar que los cambios realizados no afecten la funcionalidad del software. Se recomienda utilizar **JUnit** en **Eclipse** para automatizar estas pruebas. Además, se puede utilizar un **analizador de código estático**, como **SonarQube**, para identificar posibles problemas de calidad en el código.

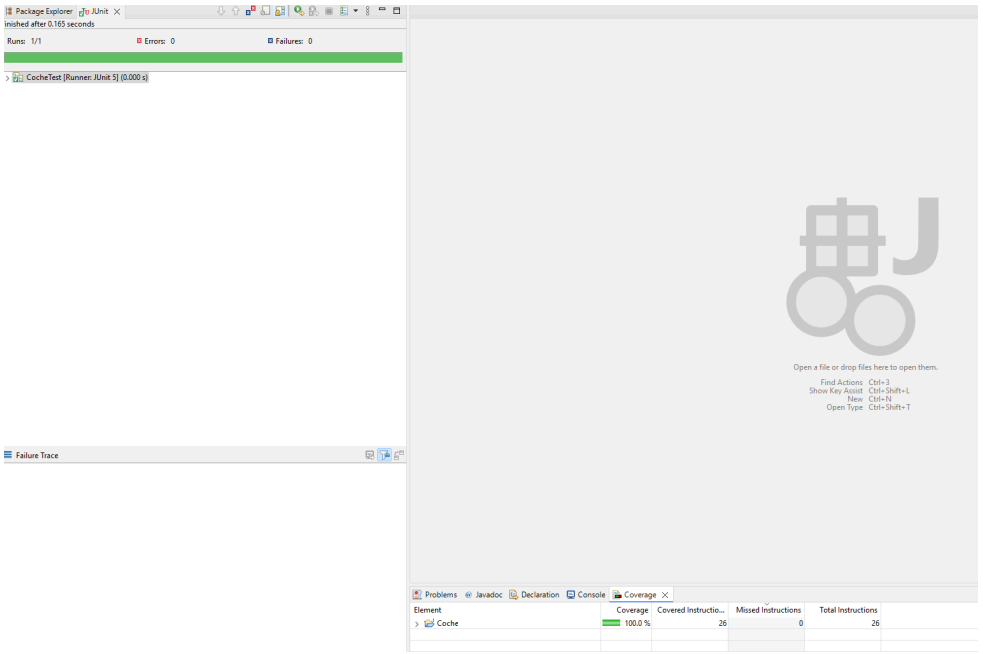
JUnit:

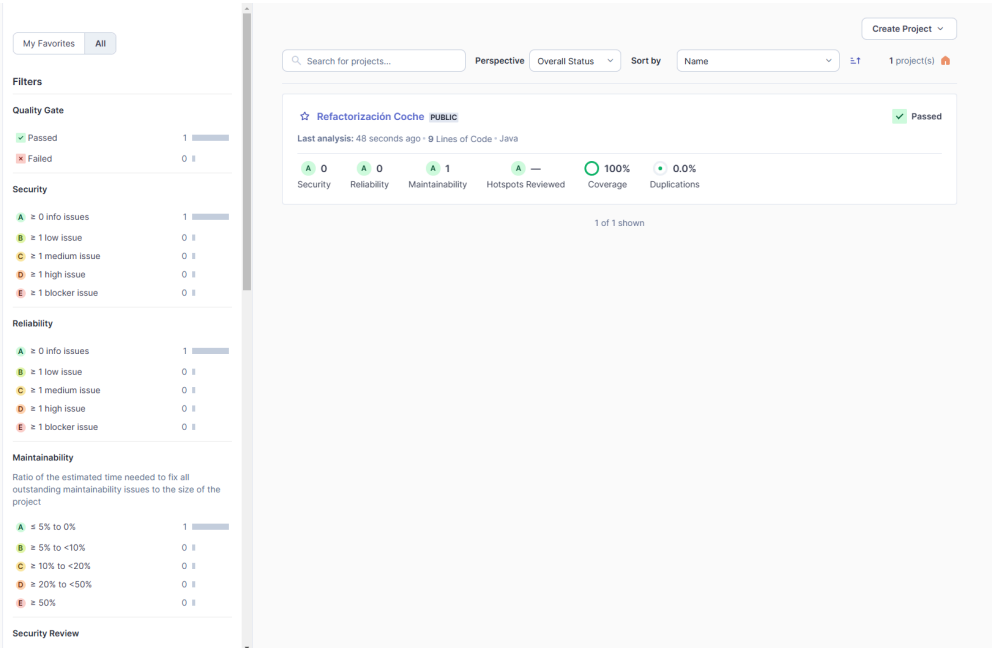
JUnit es un marco de pruebas unitarias para Java que permite automatizar la validación del comportamiento de métodos individuales dentro de una clase. A través de anotaciones como **@Test**, **@BeforeEach** y **@AfterEach**, se pueden estructurar pruebas para asegurarse de que cada componente del software funcione de manera independiente y conforme a las expectativas.



SonarQube:

SonarQube es una herramienta de análisis estático del código que ayuda a identificar problemas de calidad, seguridad y rendimiento. Proporciona métricas detalladas sobre **Code Smells**, **Bugs**, **Vulnerabilidades** y **Cobertura de Pruebas** automatizadas. Permite a los desarrolladores detectar problemas de forma temprana, aplicar buenas prácticas de programación y mejorar la calidad general del software.





Conclusión

La refactorización de código es fundamental para mantener un código legible, mantenible y escalable. El uso de herramientas de refactorización en Eclipse facilita la implementación de buenas prácticas de programación, permitiendo un desarrollo de software más robusto y eficiente.

El uso de **JUnit** asegura que las refactorizaciones no introduzcan errores, mientras que **SonarQube** proporciona una visión profunda de la calidad y seguridad del código.

Bibliografía

- Material de la unidad
- SonarQube Tutorials Playlist (Youtube) https://www.youtube.com/playlist?list=PLDhScTEBdP8ym3PaMkkTVuaaVHV2ktI0j
- How to SonarQube Setup from scratch (Youtube) https://youtu.be/6vdRvz_LnbQ?si=PfAaOabxK1W1cu4n
- SonarQube Playlist (Youtube) https://www.youtube.com/playlist?list=PLxzKY3wu0_FL3TzBnBeBoIMoRkXmYe3VB
- SonarQube Series (Youtube) https://www.youtube.com/playlist?list=PLJRhBldgqe1oWB8kGypExY0Ru2_QP4Hvy
- SonarQube (Youtube) https://www.youtube.com/watch?v=KHH9sInxLH0&list=PLrb1e2Mp6N_tmJpz1Yc0J4jj54GzRvLXM&ab_channel=I%C3%B1igoSerrano
- JUnit Basics Playlist (Youtube) https://www.youtube.com/watch?v=2E3WqYupx7c&list=PLqq-6Pq4lTTa4ad5JISViSb2FVG8Vwa4o&ab_channel=JavaBrains
- JUnit Testing Tutorial Videos (Youtube) https://www.youtube.com/watch?v=S0kChxadbSE&list=PLEiEAq2VkUUIRbjl0r1s2397CWs8UIhKU&ab_channel=Simplilearn