Laboratorio para Mejora de Calidad de Código con SonarLint/SonarQube en IntelliJ IDEA

¿Qué es SonarLint/SonarQube?

SonarLint es una herramienta de análisis estático de código que se integra directamente en tu IDE (como IntelliJ IDEA) y te ayuda a detectar errores, vulnerabilidades y malas prácticas de codificación *en tiempo real*, mientras escribes código.

Ventajas principales:

- Detecta bugs, code smells y vulnerabilidades.

1. Bugs (Errores de programación)

¿Qué son?

Los bugs son errores de lógica o comportamiento en el código que pueden provocar fallos en el tiempo de ejecución. Son defectos que generalmente afectan el funcionamiento correcto del software.

Ejemplos:

- División por cero.
- Comparaciones incorrectas (`if (a = b)` en lugar de `if (a == b)`).
- Uso de objetos `null` sin verificación previa (lo que puede causar un `NullPointerException`).
- Acceder a un índice fuera de los límites de un array.

¿Por qué importan?

Los bugs afectan directamente la correctitud del software y pueden causar bloqueos, errores visibles para el usuario o resultados incorrectos.

Code Smells (Malos olores de código)

¿Qué son?

Los code smells son estructuras de código que, aunque no provocan errores directamente, indican que el código podría ser difícil de entender, mantener o extender. Son señales de que algo no está bien diseñado.

Ejemplos:

- Métodos muy largos.
- Código duplicado.
- Variables o métodos no usados.
- Nombres poco descriptivos.
- Condiciones complejas o innecesarias (if (x > 0) return true; else return false;).

¿Por qué importan?

No rompen el programa, pero hacen que el código sea más propenso a errores futuros, difícil de leer y costoso de mantener.

3. Vulnerabilidades (Fallos de seguridad)

¿Qué son?

Las vulnerabilidades son problemas en el código que pueden ser explotados por atacantes para comprometer la seguridad del sistema. Suelen permitir accesos no autorizados, fugas de datos o ejecución de código malicioso.

Ejemplos:

- Inyecciones SQL o de comandos.
- No cerrar recursos como InputStreams o Scanners, lo que puede llevar a fugas de memoria.
- Usar contraseñas o claves en texto plano en el código fuente.
- Uso de cifrados inseguros o algoritmos obsoletos.

¿Por qué importan?

Pueden permitir que un atacante robe información, tome control del sistema o cause daños. Son críticos en aplicaciones web, móviles y empresariales.

- Funciona sin conexión o conectado a SonarQube.
- Es compatible con Java, JavaScript, Python, C#, entre otros lenguajes.

- Se integra con IntelliJ IDEA, VS Code, Eclipse, y más.

Es como tener un revisor de código automático a tu lado.

Objetivo del laboratorio

- Integrar SonarLint en IntelliJ IDEA.
- Analizar un proyecto Java con errores de calidad.
- Identificar y corregir al menos 10 problemas de diferentes categorías.
- Aplicar buenas prácticas de programación.

Estructura del Proyecto

```
sonarlint-advanced-lab/
 src/
  └── main/
     └── java/
        └── com/example/app/
           ├── App.java
           ├── Calculator.java
           └── UserService.java
```

Código Base con Errores Intencionales

App.java

```
package com.example.app;

import java.util.Scanner;

public class App {

    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        int a = 10;
        int b = 0;
        System.out.println("Sum: " + calculator.sum(a, b));
        System.out.println("Division: " + calculator.divide(a, b)); // División por cero

        for (int i = 0; i < 3; i++) {
            System.out.println("Iteration " + i);
        }
```

```
        for (int i = 0; i < 3; i++) { // Código duplicado
            System.out.println("Iteration " + i);
        }

        UserService userService = new UserService();
        userService.processUser(null); // Posible NPE

        Scanner scanner = new Scanner(System.in); // Recurso no cerrado
        System.out.println("Enter a number:");
        int input = scanner.nextInt();
    }
}
```

Calculator.java

```
package com.example.app;

public class Calculator {

    public int sum(int a, int b) {
        return a + b;
    }

    public int divide(int a, int b) {
        return a / b;
    }

    public boolean isPositive(int num) {
        if (num > 0) {
            return true;
        } else {
            return false;
        }
    }

    public void unusedMethod() {
        int x = 42;
    }
}
```

UserService.java

```
package com.example.app;

public class UserService {
```

```
    public void processUser(String name) {
        System.out.println("User: " + name.toUpperCase());
    }

    public void doNothing() {
        // Método vacío
    }

    public void longMethod() {
        int total = 0;
        for (int i = 0; i < 10; i++) {
            total += i;
        }
        for (int i = 0; i < 10; i++) {
            total += i;
        }
        for (int i = 0; i < 10; i++) {
            total += i;
        }
        for (int i = 0; i < 10; i++) {
            total += i;
        }
    }
}
```

Errores a Detectar con SonarLint

	Archivo	Problema detectado	Tipo
1	**App.java**	División por cero	Bug
2	**App.java**	Código duplicado (dos bucles for)	Code Smell
3	**App.java**	Scanner no cerrado	Vulnerabilidad
4	**App.java**	Posible NullPointerException	Bug
5	**Calculator.java**	Método booleano innecesariamente verboso (isPositive)	Code Smell
6	**Calculator.java**	Método no utilizado (unusedMethod)	Code Smell

7	**Calculator.java**	Variable no usada (x)	Code Smell
8	**UserService.java**	Método vacío (doNothing)	Code Smell
9	**UserService.java**	Método muy largo con código repetitivo (longMethod)	Code Smell

Instrucciones Paso a Paso

1. Configura tu entorno

- Abre IntelliJ IDEA.
- Crea el proyecto sonarlint-advanced-lab copia y pega el código en cada uno de los ficheros java que se te han dado anteriormente.
- Instala y habilita el plugin SonarLint (File > Settings > Plugins > Marketplace).

2. Analiza el proyecto

- Haz clic derecho sobre el proyecto > Analyze with SonarLint.
- Examina el panel de resultados de SonarLint.
- Identifica y anota los errores detectados (al menos 9).

3. Corrige los problemas (ejemplos)

- **Evitar división por cero**

```
public int divide(int a, int b) {
   if (b == 0) throw new IllegalArgumentException("Divisor no puede ser cero");
   return a / b;
}
```

- **Validar null en processUser**

```
public void processUser(String name) {
   if (name != null) {
      System.out.println("User: " + name.toUpperCase());
   } else {
      System.out.println("Nombre no puede ser null");
   }
```

```
}
```

- **Cerrar recursos**

```
try (Scanner scanner = new Scanner(System.in)) {
    System.out.println("Enter a number:");
    int input = scanner.nextInt();
}
```

- **Simplificar isPositive**

```
public boolean isPositive(int num) {
    return num > 0;
}
```

- **Eliminar código duplicado y métodos vacíos/no usados**

Refactorizar o eliminar según corresponda.

4. Verifica tus correcciones

- Vuelve a ejecutar Analyze with SonarLint.
- Verifica que todos los errores hayan desaparecido.
- Si persiste alguno, consulta el mensaje detallado de SonarLint para resolverlo.

Actividad Final

- Documenta los 10 errores identificados y cómo los corregiste.
- Entrega el código corregido junto con un breve informe de tu proceso.
- Sube el proyecto corregido a GitHub.