



**POLITÉCNICO
ESTELLA**

ASIGNATURA: ETORNOS DE DESARROLLO

**GRADO SUPERIOR EN DESARROLLO DE
APLICACIONES MULTIPLATAFORMA**

TAREA EDDE 08

Presentado por: Eugen Moga

Índice

Ejercicios 1	1
Ejercicios 2	2
Ejercicios 3	3
Ejercicios 4	4
Ejercicios 5	6
Ejercicios 6	6
Ejercicios 7	8
Ejercicios 8	9
Ejercicios 9	10
Ejercicios 10	12

Ejercicios 1

1. Lanza el analizador de javadocs para el package creado. Genera automáticamente las opciones que JavaDocs te propone. Completa los comentarios generados y crea (si procede y si no están) los comentarios Javadocs siguientes:

@author y @version para la clase main.

@param y @return para cada método

Actualiza el análisis Javadocs para comprobar que no faltan.

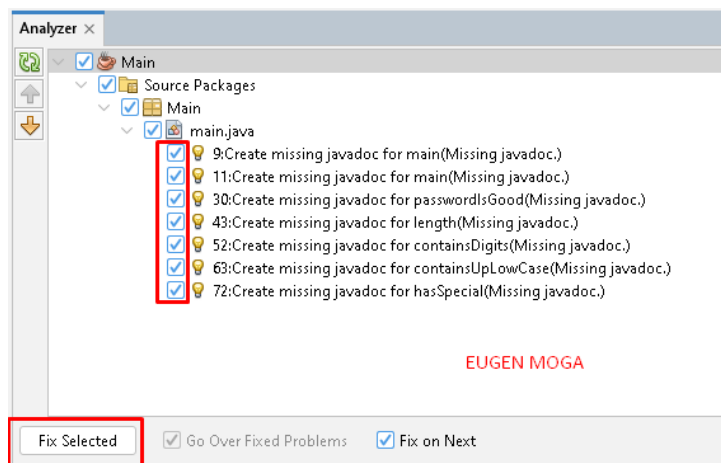
Termine la tarea 6 muy confundido y sin saber si lo que estoy haciendo es lo que se pide y empiezo esta tarea con la misma sensación de frustración.

¿Cómo lanzo el analizador de javadocs para el package creado?

No encuentro nada al respecto en el temario (curiosamente para Eclipse que no usamos si tenemos como poner comentarios JavaDoc).

Así que paso a explicar como pongo los comentarios javadoc automáticamente y completo la información que falta.

En la clase main.java clic derecho Tools > Analyze Javadoc selecciono la clase y todos los métodos y le pulso en Fix Selected



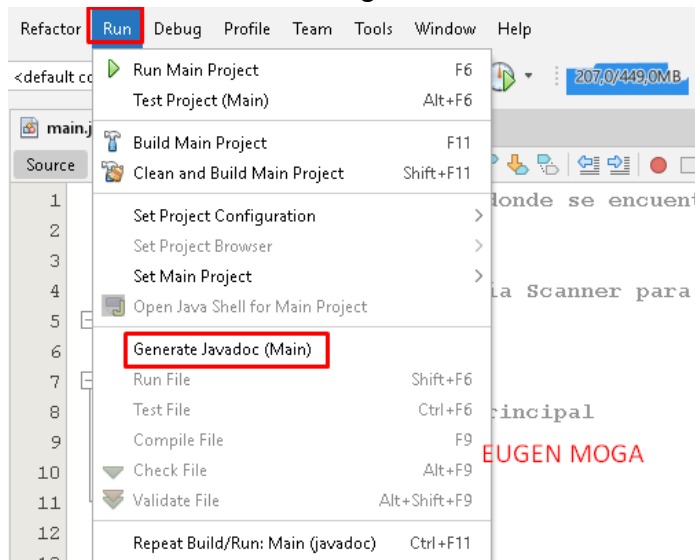
Ya se ponen todos los comentarios para todos los métodos y para la clase.

Paso a completarlo con la información que falta

Ejercicios 2

2. Genera la documentación Javadoc para el proyecto y comprueba que abarca todos los métodos y atributos de la clase main.

Una vez puestos los comentarios Javadoc para generar la documentación le doy clic al botón Run > Generate Javadoc (Main) y me abre la documentación en una ventana del navegador web.



¿Cuál es la extensión del fichero generado?

Es un fichero con extensión .html

¿A quién está destinado este documento?

Este fichero está destinado para los programadores que tengan que mantener dicho programa o para futuros programadores que trabajen el proyecto.

¿De qué tipo de documentación se trata?

Se trata de documentación técnica generada a partir de los comentarios Javadoc del código. Es una referencia para entender la estructura del programa sin necesidad de leer el código fuente.

¿Todos tus comentarios de la tarea 6 han sido añadidos al fichero?

En mi caso no guarde el proyecto de la tarea 6. Pero en la tarea 6 no puse ningún comentario Javadoc por tanto no saldrían los comentarios de la tarea 6.

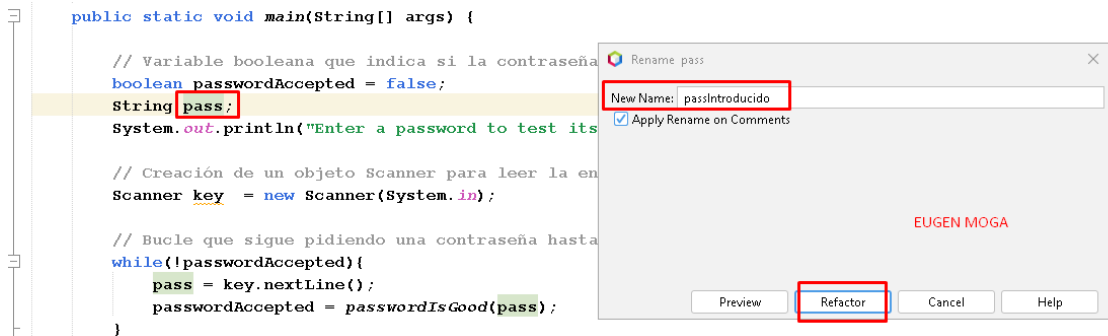
Adjunta el fichero generado en la entrega de la tarea.

Ejercicios 3

REFACTORIZACIÓN

Usando las opciones de refactorización de Netbeans, realiza los siguientes cambios y comprueba los resultados obtenidos:

En el método main le doy clic derecho en la variable pass > Refactor > Rename y le cambio el nombre a passIntroducido



3. Cambia el nombre de la variable “pass” del método main() por “passIntroducido”.

¿Ha cambiado ese nombre en los argumentos de los otros métodos?

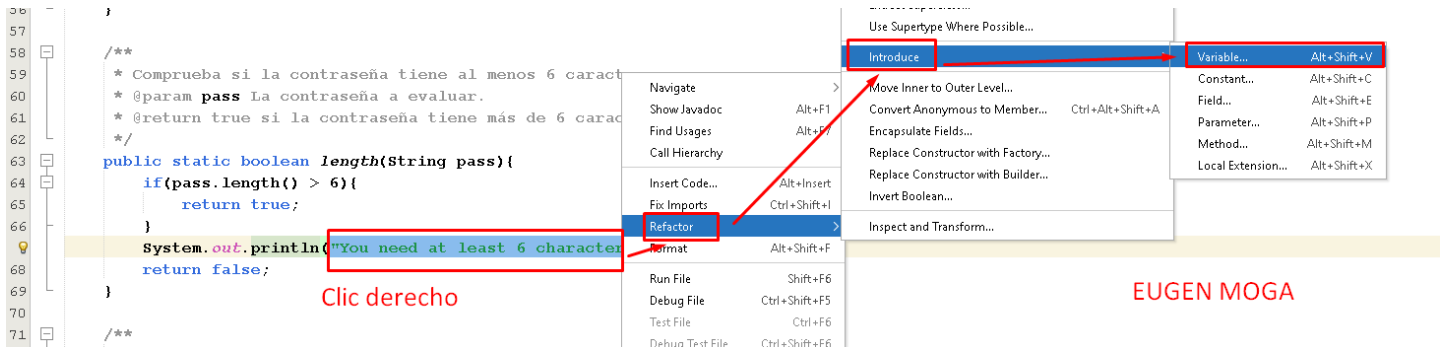
No solo ha cambiado el nombre de la variable en el método main en los demás métodos el nombre sigue siendo pass

```
20 public static void main(String[] args) {
21
22     // Variable booleana que indica si la contraseña es aceptada o no
23     boolean passwordAccepted = false;
24     String passIntroducido;
25     System.out.println("Enter a password to test its strength.");
26
27     // Creación de un objeto Scanner para leer la entrada del usuario
28     Scanner key = new Scanner(System.in);
29
30     // Bucle que sigue pidiendo una contraseña hasta que se acepte
31     while(!passwordAccepted){
32         passIntroducido = key.nextLine();
33         passwordAccepted = passwordIsGood(passIntroducido);
34     }
35
36     key.close();
37 }
38
39 /**
40  * Verifica si una contraseña cumple con todos los requisitos de seguridad.
41  *
42  * @param pass La contraseña ingresada por el usuario.
43  * @return true si la contraseña es fuerte, false en caso contrario.
44  */
45
46 public static boolean passwordIsGood(String pass){
47
48     // Comprueba si la contraseña cumple todas las reglas de seguridad
49     if(length(pass) && containsDigits(pass) &&
50        containsUpLowerCase(pass) && hasSpecial(pass)){
51         System.out.println("Your password is strong!");
52         return true;
53     }
54     System.out.println("Weak password. Try another.");
55     return false;
56 }
```

Ejercicios 4

4. Para cada uno de los 4 métodos de comprobación de la contraseña, selecciona el texto de ayuda de la sentencia "system.out.println()", aplica las refactorizaciones siguientes y comenta su resultado:

Texto del método length(): opción "introduce->variable"



En este caso convierte el mensaje en una variable local de tipo String y le he puesto el nombre a la variable mensajeError

```
public static boolean length(String pass){  
    if(pass.length() > 6){  
        return true;  
    }  
    String mensajeError = "You need at least 6 characters.";  
    System.out.println(mensajeError);  
    return false;  
}
```

Texto del método containsDigits(): opción "introduce->constant"

```
public static boolean containsDigits(String pass){  
  
    // Verifica si la contraseña no tiene un número  
    if(!pass.matches(".*\\d.*")){  
        return true;  
    }  
    System.out.println("YOU_NEED_A_DIGIT");  
    return false;  
}  
private static final String YOU_NEED_A_DIGIT = "You need a digit.";
```

Esto convierte el mensaje en una constante y su valor no cambiara también puede ser reutilizado en toda la clase

Texto del método containsUpLowerCase():opción “introduce->field”

Esta opción convierte el mensaje en una variable de instancia y permite su reutilización en toda la clase

```
public static boolean containsUpLowerCase(String pass){
    if(!pass.equals(pass.toLowerCase()) && !pass.equals(pass.toUpperCase())){
        return true;
    }
    lowerUpperCase = "You need a lower and upper case letter.";
    System.out.println(lowerUpperCase);
    return false;
}
private static String lowerUpperCase;
```

Le he cambiado el nombre porque si no es muy largo también cabe destacar que dejo los modificadores de acceso por defecto.

Texto del método hasSpecial():opción “introduce->parameter”

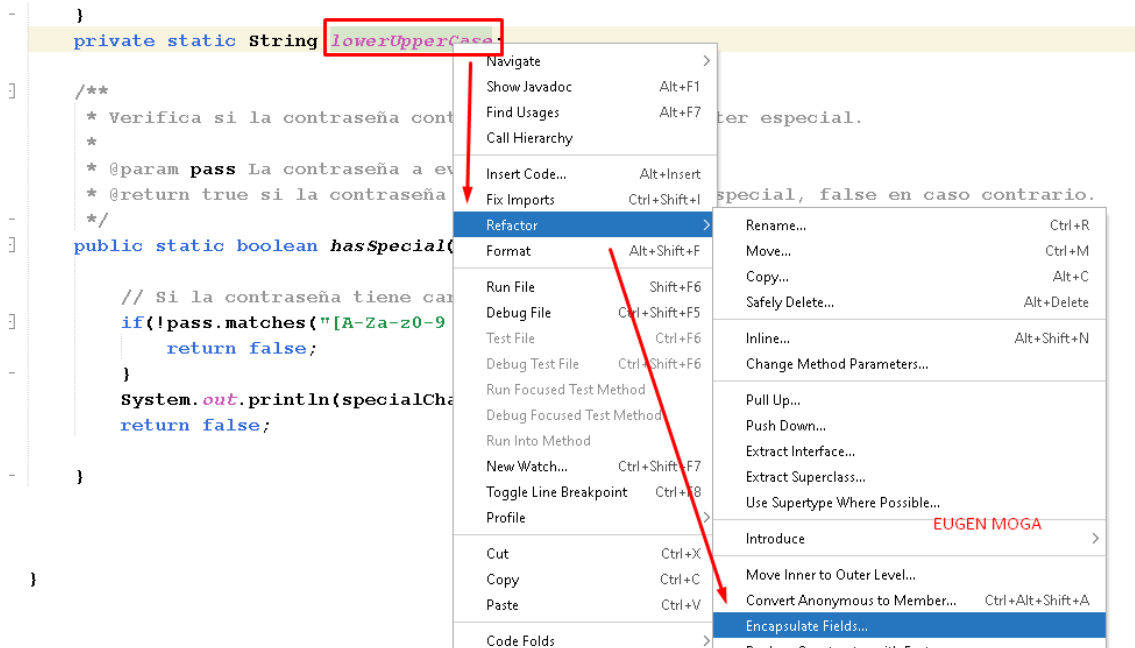
```
//
public static boolean hasSpecial(String pass, String specialCharacter){

    // Si la contraseña tiene caracteres especiales
    if(!pass.matches("[A-Za-z0-9 ]*")){
        return false;
    }
    System.out.println(specialCharacter);
    return false;
```

Esta opción convierte el mensaje en un parámetro del método y permite personalizar el mensaje al llamar la función.

Ejercicios 5

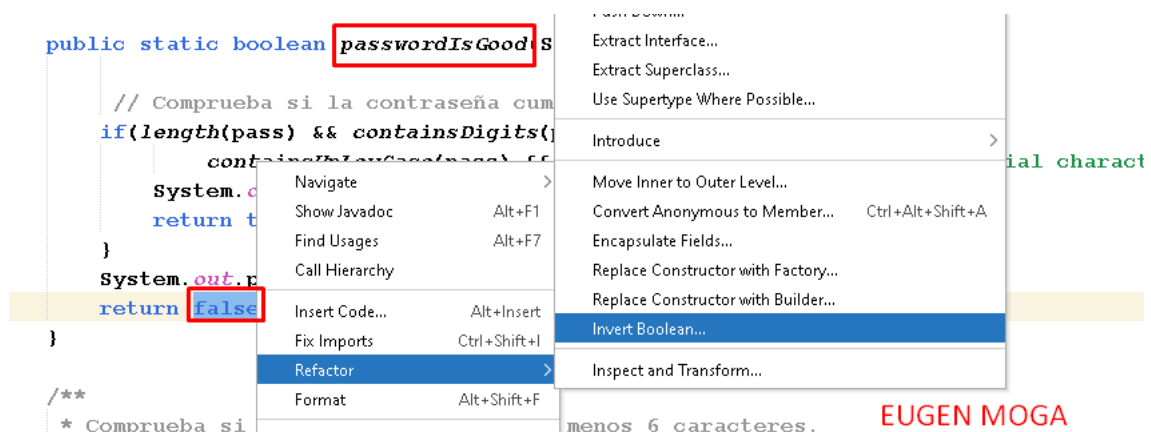
5. Encapsula una de las variables creadas. ¿Qué cambios ves en el código?



Al encapsular la variable significa que la convierto en privada y NetBeans genera automáticamente los métodos de acceso getters y setters

Ejercicios 6

6. Selecciona la palabra “false” del método passwordIsGood() y refactoriza con la opción “invertboolean”, indicando un nuevo nombre de método más coherente que este cambio.



En el método selecciono la palabra false le doy clic derecho > Refactor > Invert Boolean

Renombro el método passwordIsWeak()

```
public static boolean passwordIsWeak(String pass){  
    // Comprueba si la contraseña cumple todas las reglas de seguridad  
    if(length(pass) && containsDigits(pass) &&  
        containsUpLowerCase(pass) && hasSpecial(pass, "You need a special character.")){  
        System.out.println("Your password is strong!");  
        return false;  
    }  
    System.out.println("Weak password. Try another.");  
    return true;  
}
```

EUGEN MOGA

¿Qué cambios ves en el código? ¿Hemos cambiado el funcionamiento de ese método?

Aparte del cambio de nombre del método en la verificación hasSpecial(pass)) ahora se ha añadido el parámetro "You need a special character"

En teoría no se ha cambiado el resultado del método, solo su lógica interna. El funcionamiento es el mismo solo que ahora el método expresa lo contrario

Pero en la practica estoy probando y me sale siempre el error que necesito un dígito, voy a solucionarlo para probar correctamente el funcionamiento.

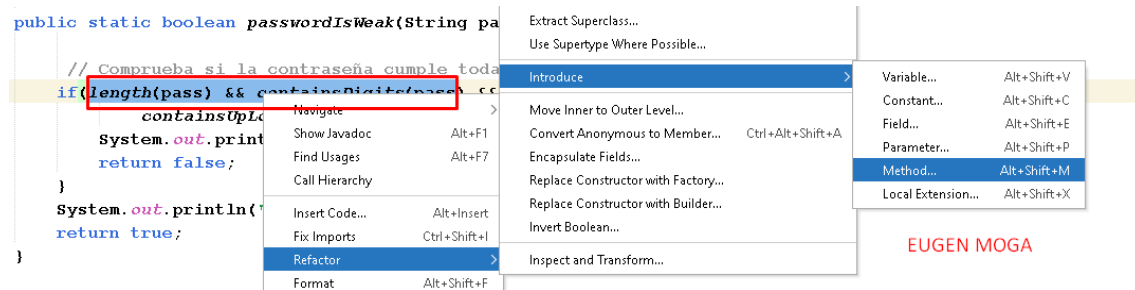
El error lo tengo en este método que comprueba lo contrario: me decía que necesitaba un dígito, cuando le ponía el dígito y me decía que la contraseña era fuerte cuando no llevaba un dígito.

```
public static boolean containsDigits(String pass){  
    // Verifica si la contraseña no tiene un número  
    if(!pass.matches(".*\\d.*")){  
        return true;  
    }  
    System.out.println(getYOU_NEED_A_DIGIT());  
    return false;  
}
```

Ahora puedo decir que no hemos cambiado el funcionamiento del método solo su lógica interna pero el método sigue comprobando correctamente.

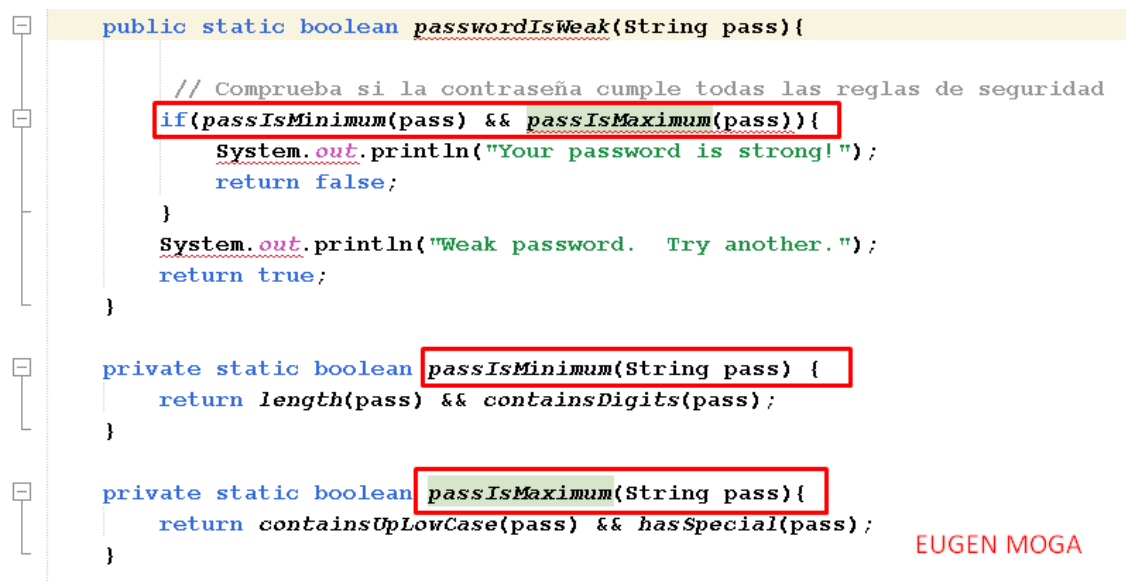
Ejercicios 7

7. Selecciona el código “length(pass) &&containsDigits(pass)” y refactoriza con la opción “introduce->metodo”, dando nombre del nuevo método “passIsMinimum()”.



Selecciona el código “containsUpLowerCase(pass) &&hasSpecial(pass)” y refactorizo con la opción “introduce->metodo”, dando nombre del nuevo método “passIsMaximum()”.

Este paso lo hago manualmente por el error comentado en el foro



¿Qué cambios ves en el código?

El cambio es que al refactorizar introduciendo un método, las condiciones que verifican la longitud, si contiene dígitos etc pasan a un método que se accede por la variable passIsMinimum y passIsMaximum.

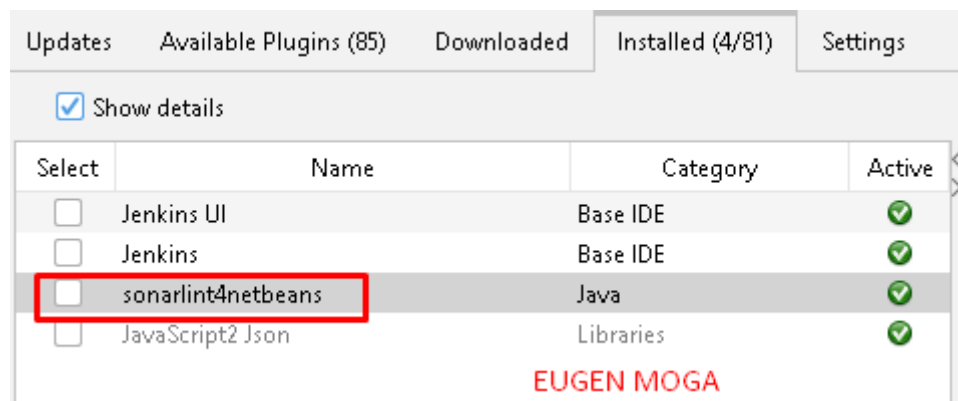
Ahora el método passwordIsWeak esta mas organizado, se ve más limpio y es más fácil de entender

Ejercicios 8

Instala el plugin del analizador de código SonarLint:

<https://plugins.netbeans.apache.org/catalogue/?id=21>

Accedo a la web selecciono el archivo NB 23 de la version 3.0.1 y se descarga. Desde la pestaña Downloaded lo selecciono y le doy a instalar, pero NetBeans me decía que ya está instalado así que en la pestaña de Installed marque la opción Show details y lo busque para activarlo.



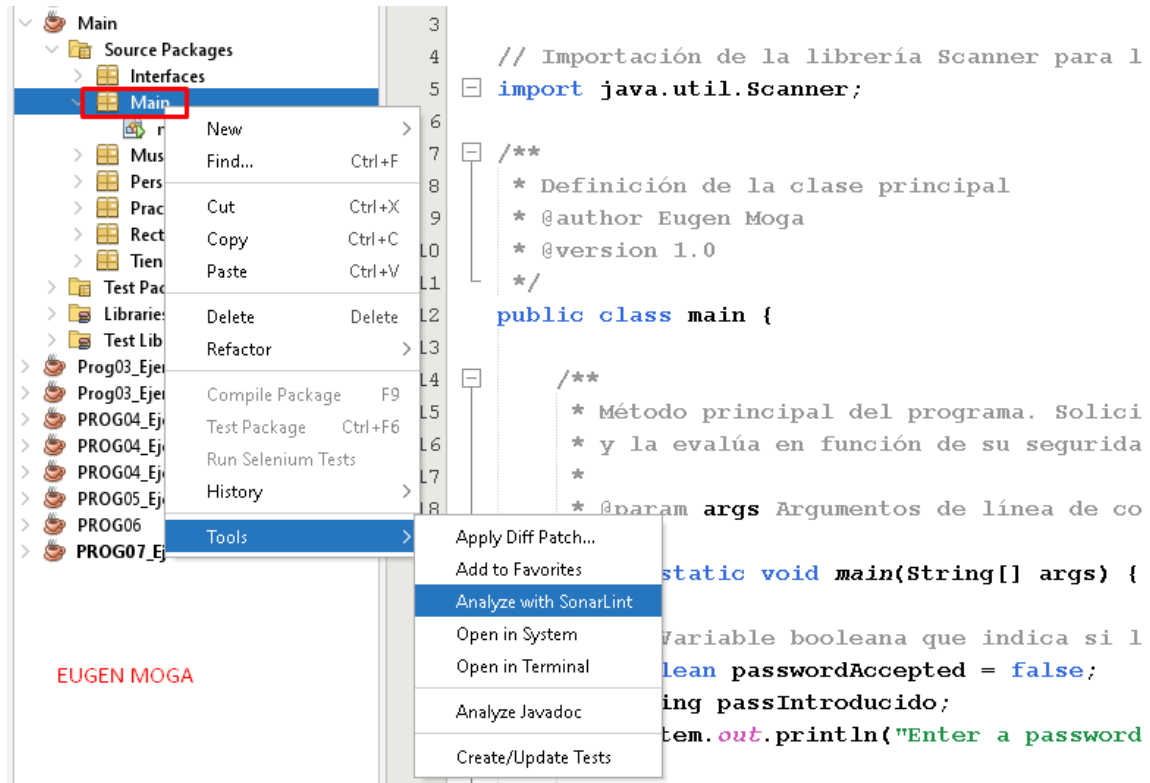
Ejercicios 9

Lanza el análisis para el proyecto.

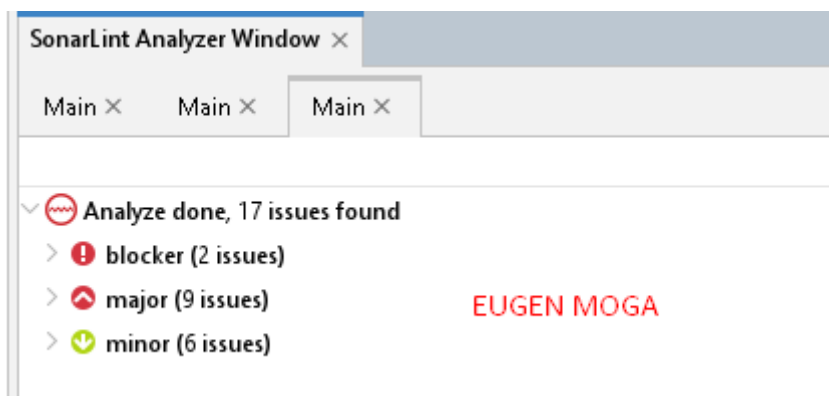
<https://www.youtube.com/watch?v=f0KOFe-EVJI>

Comenta tres de los errores o avisos generados por el analizador.

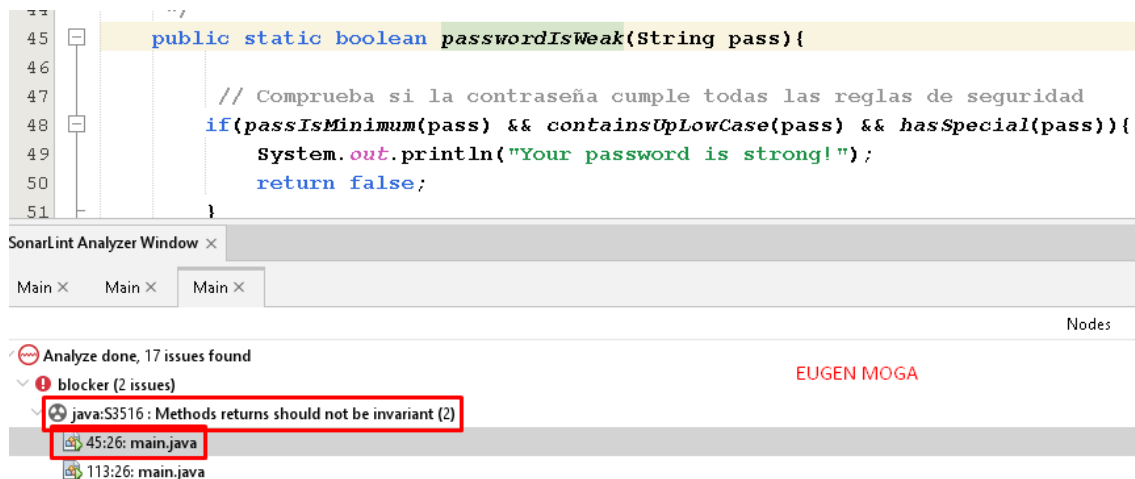
Para lanzar el análisis le doy clic derecho en el proyecto, Analyze with SonarLint pero en mi caso tengo mas trabajos en el proyecto asi que le doy clic derecho al paquete Main > Tools > Analyze with SonarLint



SonarLint organiza los errores en tres categorías: blocker, mayor y menor



Blocker son errores críticos que pueden afectar a la ejecución del programa.



En este caso tengo dos errores blocker que dice que los métodos que me marca en la línea 45 y 113 devuelven valores constantes o invariantes que no cambian, de momento este error esta relacionado con el ejercicio 7 que tengo sin terminar por el error que he puesto en el foro.

Después de realizar el ejercicio 7 el error persiste,

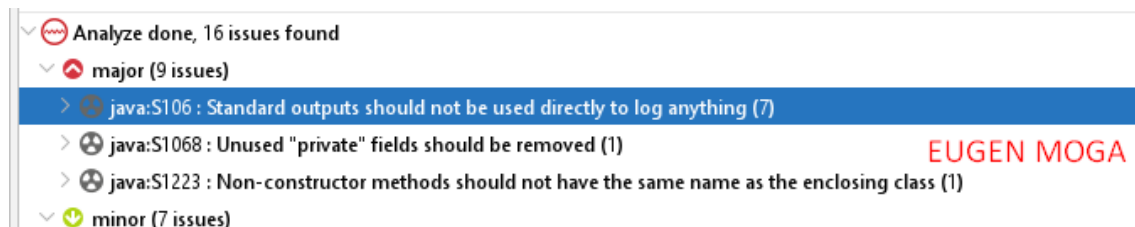
Un error está en el método hasSpecial que siempre devuelve false da igual la condición que se cumple. Lo soluciono.

El otro error Blocker esta en el método getYOU_NEED_A_DIGIT que siempre devuelve (you need a digit), soluciono esos problemas para que el programa funcione correctamente.

Major hace referencia a error grave, código propenso a errores o difícil de mantener.

Uno de los errores que sale en este apartado es el106 Standard outputs should not be used directly to log anything.

Que básicamente dice que el uso de System.out.println para mostrar mensajes en consola no es una buena practica en Java y que se recomienda el uso de logger



Minor es un error leve que puede considerarse como una recomendación de mejora

Aquí puedo encontrar el error 100 que dice Method names should comply with a naming convention.

Esto dice que los nombres de los métodos no siguen las convenciones de nomenclatura recomendadas en Java y es debido que a la hora de refactorizar no les he cambiado el nombre a todos los métodos y he dejado el que salía por defecto.

▼ minor (7 issues)

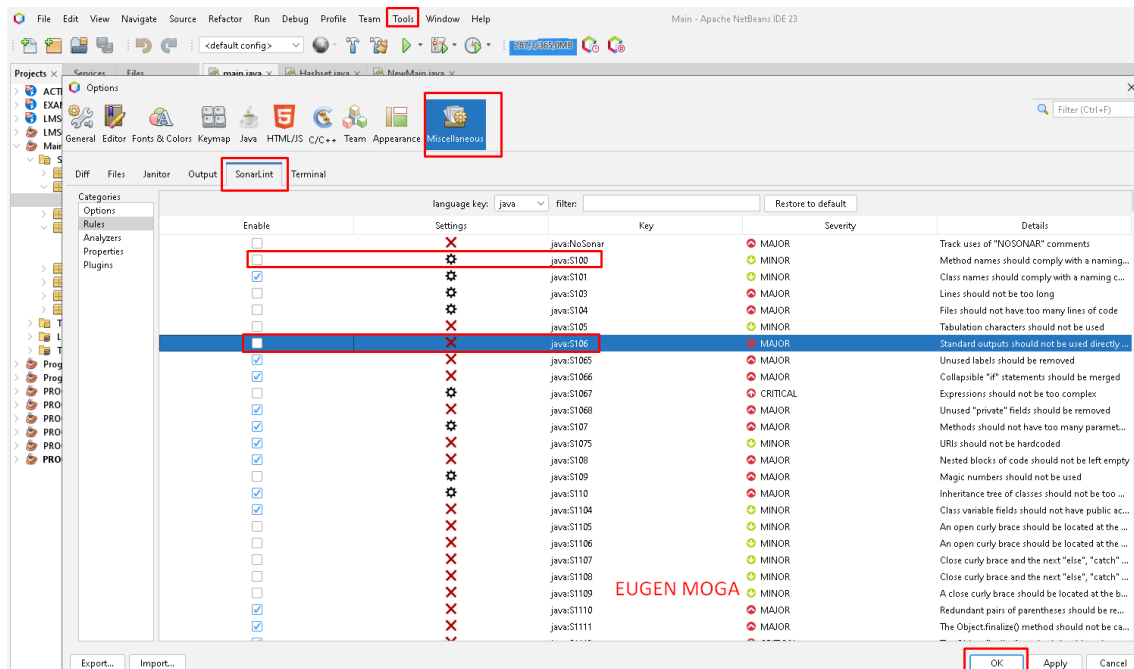
- java:S100 : Method names should comply with a naming convention (2)
- java:S101 : Class names should comply with a naming convention (1)
- java:S117 : Local variable and method parameter names should comply with a naming convention (1)
- java:S120 : Package names should comply with a naming convention (1)
- java:S1450 : Private fields only used as local variables in methods should become local variables (1)
- java:S3008 : Static non-final field names should comply with a naming convention (1)

EUGEN MOGA

Ejercicios 10


10.Deshabilita aquellos tres avisos en la configuración de reglas del analizador y lanza de nuevo el analizador



Para deshabilitar los avisos voy a Tools > Options > Miscellaneous y en la pestaña de SonarLint aparecen todos los avisos. Lo filtro por lenguaje java deshabilito el aviso 100 y 106









EUGEN MOGA

Vuelvo a lanzar SonarLint y se puede observar que el error 100 y 106 ya no salen en el aviso.

 **major (2 issues)**

- >  java:S1068 : Unused "private" fields should be removed (1)
- >  java:S1223 : Non-constructor methods should not have the same name as the enclosing class (1)

 **minor (5 issues)**

- >  java:S101 : Class names should comply with a naming convention (1) **EUGEN MOGA**
- >  java:S117 : Local variable and method parameter names should comply with a naming convention (1)
- >  java:S120 : Package names should comply with a naming convention (1)
- >  java:S1450 : Private fields only used as local variables in methods should become local variables (1)
- >  java:S3008 : Static non-final field names should comply with a naming convention (1)