

Actividad

09

CURSO 2016-2017

<Actividad09>

<FRANCISCO JAVIER MARQUÉS GAONA>

**PROCESOS DE LA INGENIERÍA DEL
SOFTWARE II**

4º GRADO EN INGENIERÍA INFORMÁTICA



Departamento de Informática
Universidad de Almería

REFACTORIZACIÓN PROYECTO 8 REINAS

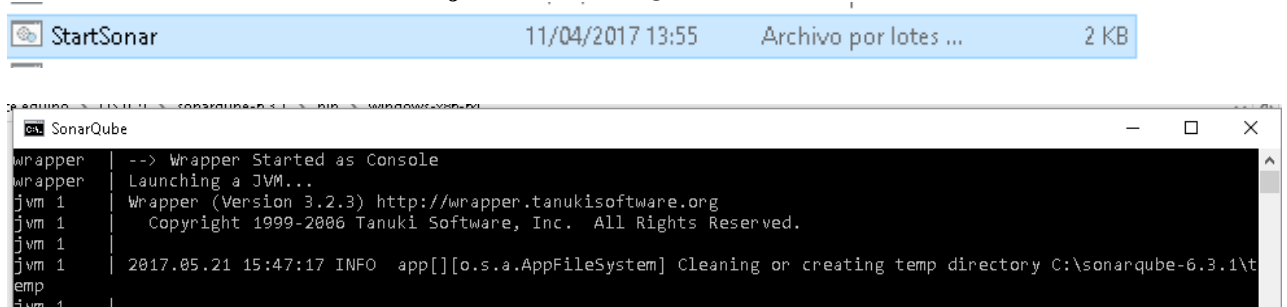
Se procederá a la refactorización del proyecto realizado en la actividad. El relacionado con los monomios, polinomios y las respectivas operaciones que debíamos realizar.

1. Mostrando el proyecto en SonarQube

Para mostrar el proyecto en SonarQube seguimos las instrucciones proporcionadas en la Web CT y que ya se han utilizado en actividades anteriores.

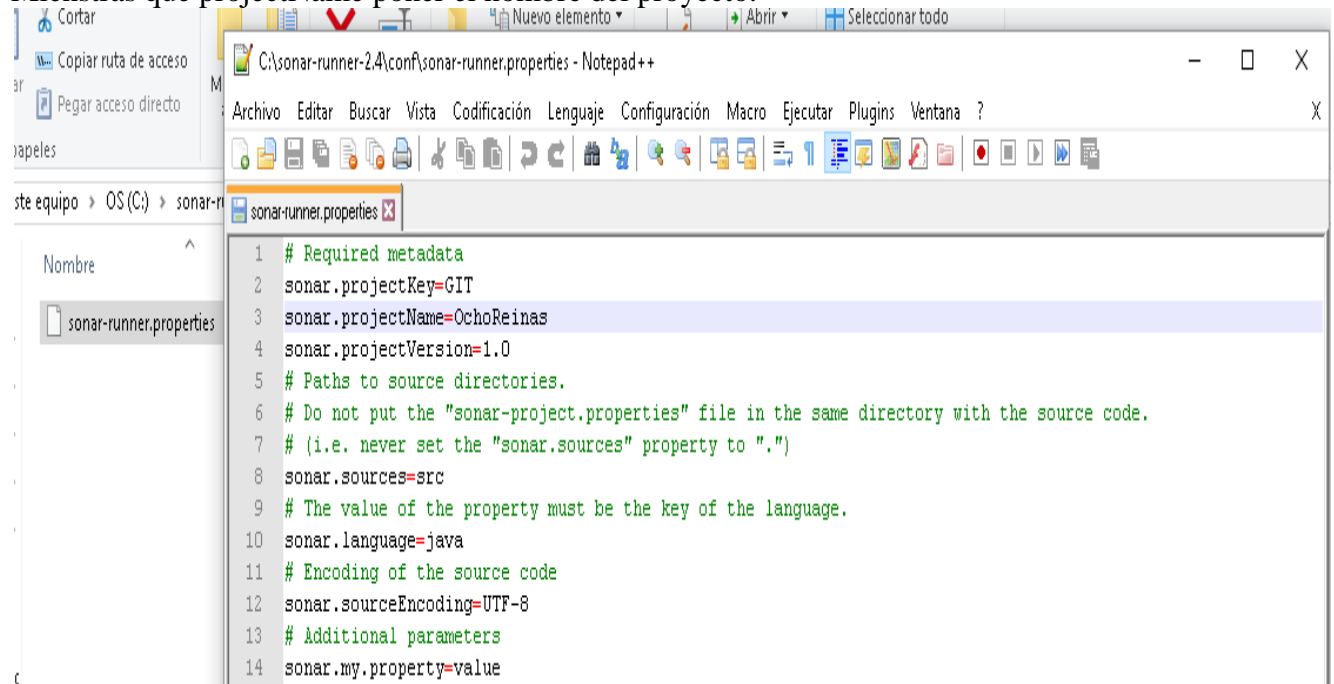
Tras los siguientes pasos podremos ver el análisis de nuestro proyecto de forma local:

Ejecutar SonarQube



Cambiar la configuración de Sonar-Runner

Añadir en projectKey el nombre de la carpeta donde se ubica el proyecto
Mientras que projectName poner el nombre del proyecto.



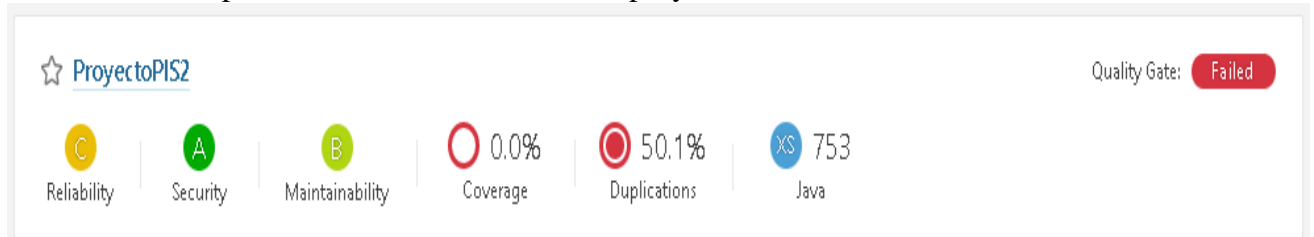
Ejecutar Sonar-Runner

Debemos ubicarnos en el directorio donde se encuentra el proyecto

```
C:\Users\Fran\Desktop\WORKSPACES\GIT\ProyectoPIS2>sonar-runner
```

```
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
Total time: 14.498s
Final Memory: 18M/407M
INFO: -----
```

Finalmente nos aparecerá en local el análisis del proyecto:



Aunque se ha conseguido instalar el plugin de eCobertura añadiendo carpetas de junit de versiones de eclipse más antiguas no se ha conseguido mostrar en SonarQube información referente a la cobertura del código:

org.junit_4.10.0.v4_10_0_v20120426-0900	21/05/2017 15:20	Carpeta de archivos
org.junit_4.12.0.v201504281640	24/09/2015 6:40	Carpeta de archivos
org.junit4_4.8.1.v20120523-1257	21/05/2017 15:20	Carpeta de archivos

Navigation: eCobertura > Coverage Session View

%	Branches	Total

Buscando información, este problema sucede ya que este plugin solo es compatible con versiones de SonarQube menores a la 6.0, y en la realización de las actividades usamos la versión 6.3.1 de SonarQube

SQ <6.0 ONLY

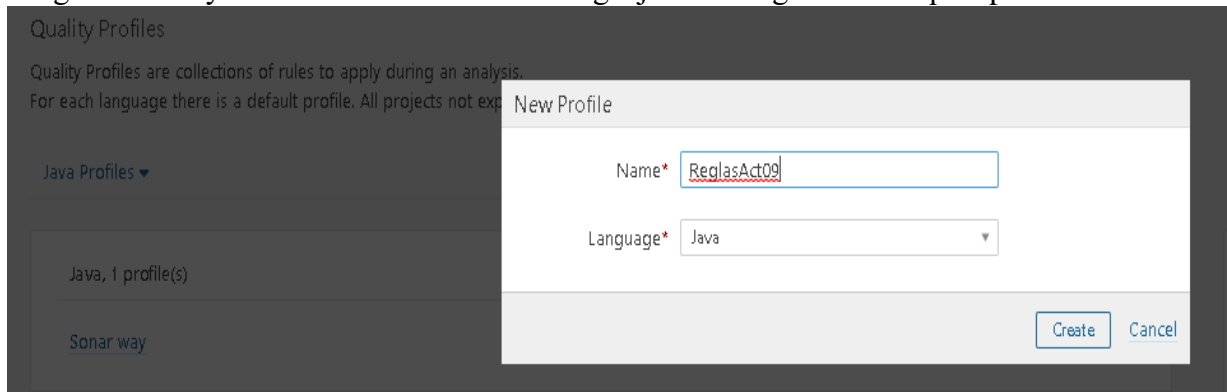
Import Cobertura code coverage report.

Set the property to the path of the Cobertura .xml report.

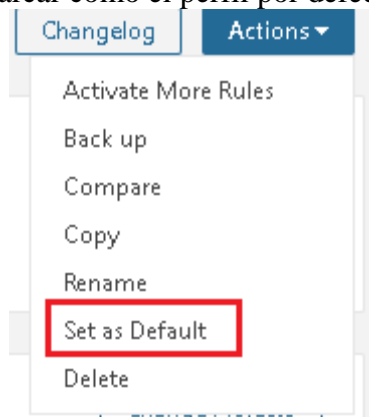
Note that the [Cobertura Plugin](#) is incompatible with SonarQube > 6.0.

2. Creación de un perfil de calidad

En primer lugar crearemos el perfil de calidad en la ventana “Quality Profiles” pulsando el botón Create, posteriormente indicamos el nombre del nuevo perfil en nuestro caso “ReglasAct09” y seleccionando además el lenguaje del código sobre el que aplicar.






Posteriormente lo debemos marcar como el perfil por defecto:


























Java, 2 profile(s)	Projects	Rules	Updated	Used
ReglasAct09	Default	15	hace 5 minutos	Never ▼
Sonar way	0	277	Never	hace una hora ▼

3. Añadiendo reglas

Para añadir reglas sobre nuestro perfil pulsamos sobre él y posteriormente sobre “Activate More”

Rules	Active	Inactive
Total	0	397
 Bugs	0	149
 Vulnerabilities	0	31
 Code Smells	0	217
Activate More		

A continuación nos aparecen todas las reglas que se pueden añadir:

"equals()" should not be used to test the values of "Atomic" classes	Java  Bug  multi-threading ▼	Activate
"@Deprecated" code should not be used	Java  Code Smell  cert, cwe, obsolete ▼	Activate
"@NonNull" values should not be set to null	Java  Bug ▼	Activate
"@Override" should be used on overriding and implementing methods	Java  Code Smell  bad-practice ▼	Activate
"action" mappings should not have too many "forward" entries	Java  Code Smell  brain-overload, struts ▼	Activate
"Arrays.stream" should be used for primitive arrays	Java  Bug  performance ▼	Activate
"assert" should only be used with boolean variables	Java  Bug  cert, suspicious ▼	Activate
"BigDecimal(double)" should not be used	Java  Bug  cert ▼	Activate
"catch" clauses should do more than rethrow	Java  Code Smell  cert, clumsy, finding, unused ▼	Activate
"clone" should not be overridden	Java  Code Smell  suspicious ▼	Activate
"Cloneables" should implement "clone"	Java  Bug ▼	Activate
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	Java  Code Smell  obsolete, pitfall ▼	Activate
"compareTo" results should not be checked for specific values	Java  Bug  unpredictable ▼	Activate

Una herramienta útil para filtrar las reglas y disminuir el total de ellas nos aparece en el margen izquierdo donde podemos buscar reglas, seleccionar por lenguaje, tipo, gravedad, etc. Con este modo podemos seleccionar reglas que nos parezcan más importantes y además que se puedan aplicar al lenguaje de nuestro proyecto (no tendría lógica añadir una regla de otro lenguaje):

☒ Language

Java	382
Python	238
C#	189
JavaScript	184
PHP	126
Flex	79

☒ Type

Bug	144
Vulnerability	31
Code Smell	207

☐ Tag

☐ Repository

☒ Default Severity





























































Blocker	30	Minor	139
Critical	48	Info	5
Major	160		

Seleccionamos un total de quince reglas que son las que se muestran a continuación:

Rules	Active	Inactive
Total	15	382
Bugs	6	143
Vulnerabilities	0	31
Code Smells	9	208

[Activate More](#)

Actividad09



	A field should not duplicate the name of its containing class	Java  Code Smell  brain-overload 	Deactivate
	Abstract classes without fields should be converted to interfaces	Java  Code Smell  java8 	Deactivate
	Anonymous inner classes containing only one method should become lambdas	Java  Code Smell  java8 	Deactivate
	Boolean literals should not be redundant	Java  Code Smell  clumsy 	Deactivate
	Constructors should not be used to instantiate "String" and primitive-wrapper classes	Java  Bug  performance 	Deactivate
	Dead stores should be removed	Java  Bug  cert, cwe, suspicious, unused 	Deactivate
	Local Variables should not be declared and then immediately returned or thrown	Java  Code Smell  clumsy 	Deactivate
	Method parameters, caught exceptions and foreach variables should not be reassigned	Java  Bug  misra, pitfall 	Deactivate
	Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	Java  Bug  clumsy 	Deactivate
	Return values should not be ignored when function calls don't have any side effects	Java  Bug  cert, misra 	Deactivate
	Standard outputs should not be used directly to log anything	Java  Code Smell  bad-practice, cert 	Deactivate
	String literals should not be duplicated	Java  Code Smell  design 	Deactivate
	Strings should not be concatenated using '+' in a loop	Java  Bug  performance 	Deactivate
	The diamond operator ("<>") should be used	Java  Code Smell  clumsy 	Deactivate
	Utility classes should not have public constructors	Java  Code Smell  design 	Deactivate

4. Refactorización del proyecto

A continuación se muestran todos los pasos realizados sobre cada una de las reglas aplicadas a nuestro perfil:

1ª Regla

A field should not duplicate the name of its containing class (squid:S1700)

 Code smell  Major



```
private ArrayList<Monomio> polinomio;
```

Este método indica que un campo no debe tener el mismo nombre que su clase contenedora, por lo que simplemente cambiando el nombre del atributo por un nombre distinto a la clase contenedora se soluciona el problema.

```
private ArrayList<Monomio> pol;
```

2ª Regla

The diamond operator ("<>") should be used (squid:S2293)

 Code smell  Minor



```
pol = new ArrayList<Monomio>();
```

En lugar de tener que declarar el tipo de una lista tanto en su declaración como en su constructor, ahora puede simplificar la declaración del constructor con <> y el compilador deducirá el tipo.

```
pol = new ArrayList<>();
```

3ª Regla

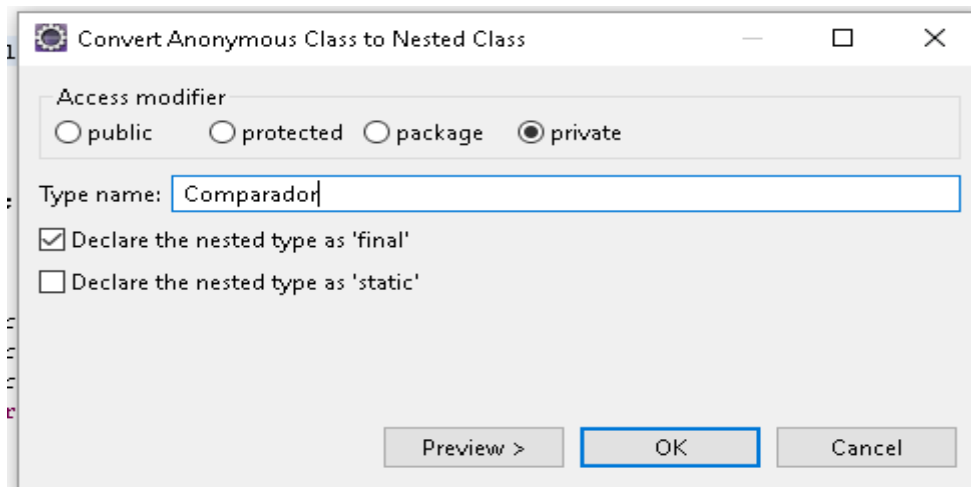
Anonymous inner classes containing only one method should become lambdas (squid:S1604)

 Code smell  Major

```
public void ordenar() {
    Collections.sort(pol, new Comparator<Monomio>() {
        @Override
        public int compare(Monomio m1, Monomio m2) {
```

La mayoría de los usos de clases internas anónimas deben ser reemplazados por lambdas para aumentar la legibilidad del código fuente.

Para ello nos ubicamos sobre Comparator botón derecho => Refactor => Convert “Anonymous Class to Nested Class”, nos aparecerá el siguiente menú:



Indicamos el nombre y los declaramos como final y se nos creará una final class que contiene el método compare entre monomios.

```
private final class Comparador implements Comparator<Monomio> {
    @Override
    public int compare(Monomio m1, Monomio m2) {
        return new Integer(m2.getExponente()).compareTo(new Integer(m1.getExponente()));
    }
}
```

El método ordenar quedaría de la siguiente forma:

```
public void ordenar() {
    Collections.sort(pol, new Comparador());
}
```

4ª Regla

Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls (squid:S1158)

Bug Minor

```
return new Integer(m2.getExponente()).compareTo(new Integer(m1.getExponente()));
```

La creación de objetos de encapsulamiento primitivos temporales sólo para la conversión de cadenas o el uso del método compareTo es ineficiente.

Por lo tanto en este caso, se debe utilizar el método comparar de la clase primitiva.

```
private final class Comparador implements Comparator<Monomio> {
    @Override
    public int compare(Monomio m1, Monomio m2) {
        return Integer.compare(m2.getExponente(), m1.getExponente());
    }
}
```

5ª Regla

Boolean literals should not be redundant (squid:S1125)

Code smell Minor

```
if(encontrado == true) {
    Monomio mSimplificado = new Monomio(suma, i);
    simplificado.add(mSimplificado);
    encontrado = false;
}
```

Los literales booleanos redundantes deben eliminarse de las expresiones para mejorar la legibilidad.

```
if(encontrado) {
    Monomio mSimplificado = new Monomio(suma, i);
    simplificado.add(mSimplificado);
    encontrado = false;
}
```

6ª Regla

Strings should not be concatenated using '+' in a loop (squid:S1643)

Code smell Minor

```
public String toString() {
    String cadena = "";
    for(int i = 0; i < pol.size(); i++) {
        if(i == pol.size() - 1 && pol.get(i).getExponente() == 0) {
            cadena = cadena + "(" + pol.get(i).getCoeficiente() + ")";
        }
        else if(i == pol.size() - 1 && pol.get(i).getCoeficiente() > 0) {
            cadena = cadena + "(" + pol.get(i).getCoeficiente() + ")x^" + pol.get(i).getExponente();
        }
        else {
            cadena = cadena + "(" + pol.get(i).getCoeficiente() + ")x^" + pol.get(i).getExponente() + " + ";
        }
    }
    return cadena;
}
```

Las cadenas son objetos inmutables, por lo que la concatenación no agrega simplemente el nuevo String al final de la cadena existente. En cambio, en cada iteración de bucle, la primera String se convierte en un tipo de objeto intermedio, se añade la segunda cadena y, a continuación, se convierte el objeto intermedio en String. Además, el rendimiento de estas operaciones intermedias se degrada a medida que el String se hace más largo. Por lo tanto, se prefiere el uso de StringBuilder tal y como se muestra a continuación:

```

public String toString() {
    StringBuilder cadena = new StringBuilder();
    for(int i = 0; i < pol.size(); i++) {
        if(i == pol.size() - 1 && pol.get(i).getExponente() == 0) {
            cadena.append("(" + pol.get(i).getCoeficiente() + ")");
        }
        else if(i == pol.size() - 1 && pol.get(i).getCoeficiente() > 0) {
            cadena.append("(" + pol.get(i).getCoeficiente() + ")x^" + pol.get(i).getExponente());
        }
        else {
            cadena.append("(" + pol.get(i).getCoeficiente()
                + ")x^" + pol.get(i).getExponente() + " + ");
        }
    }
    return cadena.toString();
}

```

7ª Regla

Local Variables should not be declared and then immediately returned or thrown (squid:S1488)

☠ Code smell ⬇ Minor

```

Polinomio pol = new Polinomio(dividendo);
return pol;

```

Declarar una variable sólo para devolverla inmediatamente después o realizar una excepción es una mala práctica.

```

    }
    return new Polinomio(dividendo);
}

```

8ª Regla

Utility classes should not have public constructors (squid:S1118)

☠ Code smell ⬆ Major

```

package actividad04.OperacionesPolinomio;

```

```

import java.util.ArrayList;

```

```

public class User {

```

Las clases de utilidad, que son una colección de miembros estáticos, no están destinadas a ser instanciadas. Incluso las clases de utilidad abstractas, que pueden ser extendidas, no deben tener constructores públicos. Java añade un constructor público implícito a cada clase que no define al menos uno explícitamente.

Por lo tanto como se trata de una clase de utilidad debemos definir un constructor aunque no sea utilizado.

```



public class User {

    private User() {
        throw new IllegalAccessException("Utility class");
    }
}

```

9ª Regla

Standard outputs should not be used directly to log anything (squid:S106)

 Code smell  Major

```
System.out.println(";Bienvenido a nuestro programa de operaciones con Polinomios!");
System.out.println("Introduce el coeficiente y el exponente de varios monomios para fo
System.out.println("Introduzca (0) en el coeficiente y (0) en el exponente para termin
```

Cuando se registra un mensaje hay varios requisitos importantes que deben cumplirse:

- El usuario debe poder recuperar fácilmente los registros
- El formato de todos los mensajes registrados debe ser uniforme para permitir al usuario leer fácilmente el registro
- Los datos registrados deben ser grabados
- Los datos confidenciales sólo se deben registrar de forma segura

Si un programa escribe directamente en las salidas estándar, no hay absolutamente ninguna manera de cumplir con esos requisitos. Es por eso que la definición y el uso de un registrador dedicado es muy recomendable.



Para la resolución de esta regla usamos `PrintStream` que añade la funcionalidad de `System.out`: ***PrintStream out = System.out***

`PrintStream out`

```
out.println(";Bienvenido a nuestro programa de operaciones con Polinomios!");
out.println("Introduce el coeficiente y el exponente de varios monomios para formar v
out.println("Introduzca (0) en el coeficiente y (0) en el exponente para terminar");
```

10ª Regla

String literals should not be duplicated (squid:S1192)

 Code smell  Critical

```
out.println("Introduzca el coeficiente: ");
int c = scan.nextInt();
out.println("Introduzca el exponente: ");
int e = scan.nextInt();
m = new Monomio(c, e);
```

Los literales de cadenas duplicados hacen que el proceso de refactorización sea propenso a errores, ya que debe estar seguro de actualizar todas las ocurrencias.

Por otro lado, las constantes pueden ser referenciadas desde muchos lugares, pero sólo necesitan ser actualizadas en un solo lugar, por lo que es aconsejable usar constantes.

Definimos las constantes que se usarán:

```
static final String coe = "Introduzca el coeficiente: ";
static final String exp = "Introduzca el exponente: ";
static final String err = "ERROR";
```

Y usamos dichas constantes:

```
//Polinomio 1
Polinomio p = new Polinomio();
Monomio m = null;
out.println(coe);
int c = scan.nextInt();
out.println(exp);
int e = scan.nextInt();
m = new Monomio(c, e);
p.add(m);

while (c != 0 || e != 0) {
    out.println(coe);
    c = scan.nextInt();
    out.println(exp);
    e = scan.nextInt();
    m = new Monomio(c, e);
    p.add(m);
    if (c == 0 && e == 0) {
        p.remove(m);
    }
}

//Polinomio 2
Polinomio p2 = new Polinomio();
Monomio m2 = null;
out.println(coe);
int c2 = scan.nextInt();
out.println(exp);
int e2 = scan.nextInt();
m2 = new Monomio(c2, e2);
p2.add(m2);

while (c2 != 0 || e2 != 0) {
    out.println(coe);
    c2 = scan.nextInt();
    out.println(exp);
    e2 = scan.nextInt();
    m2 = new Monomio(c2, e2);
    p2.add(m2);
    if (c2 == 0 && e2 == 0) {
        p2.remove(m2);
    }
}
```

11ª Regla

Dead stores should be removed (squid:S1854)

 Bug  Major

```
ArrayList<Polinomio> lista = new ArrayList<Polinomio>();
```

Un almacén inactivo ocurre cuando se asigna un valor a una variable local, incluyendo null, que no se lee por ninguna instrucción siguiente, por lo que debe ser eliminada.

Realmente lista la usamos para guardar el valor del coeficiente y del resto tras realizar la operación dividir, pero luego no la retornamos en ningún lado por lo que se trata de un error:

```
ArrayList<Polinomio> lista = new ArrayList<Polinomio>();
int s = scan.nextInt();
if (s == 1) {
    out.println("Dividendo (en ruffini): " + p.convertirRuffini().toString());
    out.println("Divisor: " + p3.toString());
    lista = p.dividir(p3);
    out.println("El resultado es: ");
    out.println("Cociente: " + lista.get(0));
    out.println("Resto: " + lista.get(1));
}
if (s == 2) {
    out.println("Dividendo (en ruffini): " + p2.convertirRuffini().toString());
    out.println("Divisor: " + p3.toString());
    lista = p2.dividir(p3);
    out.println("El resultado es: ");
    out.println("Cociente: " + lista.get(0));
    out.println("Resto: " + lista.get(1));
}
```

Para resolver este problemas podríamos crear dos métodos uno que devuelva el coeficiente (tan sencillo como devolver la posición 0 del polinomio), y otro para devolver el resto (devolver la posición 1 del polinomio), esto se debe a que el Polinomio tras realizar la división tal y como se ha realizado el método sería de la siguiente forma {Coeficiente, Resto}. Sin embargo en nuestra resolución realizamos la operación dividir y después retornamos el coeficiente o el resto. Claramente de esta forma realizamos la operación más veces por lo que resulta un gasto de recursos.

```

-----
if(s == 1) {
    out.println("Dividendo (en ruffini): " + p.converti
    out.println("Divisor: " + p3.toString());
    out.println("El resultado es: ");
    out.println("Cociente: " + p.dividir(p3).get(0));
    out.println("Resto: " + p.dividir(p3).get(1));
}
if(s == 2) {
    out.println("Dividendo (en ruffini): " + p2.convert:
    out.println("Divisor: " + p3.toString());
    out.println("El resultado es: ");
    out.println("Cociente: " + p2.dividir(p3).get(0));
    out.println("Resto: " + p2.dividir(p3));
}

```

Lo más conveniente sería implementar esos métodos y usarlos para imprimir el coeficiente o el resto:

```

public Monomio devolverCoeficiente() {
    return pol.get(0);
}

public Monomio devolverResto() {
    return pol.get(1);
}

```

El problema de dichos métodos es que solo deben usarse cuando se ha realizado la operación de dividir, si usamos “devolverCoeficiente” o “devolverResto” sobre cualquier Polinomio, no estaríamos devolviendo ni el coeficiente ni el resto si no el Monomio que se encuentra en dicha posición.

12ª Regla

Method parameters, caught exceptions and foreach variables should not be reassigned (squid:S1226)

Code smell Minor

```
while(Math.abs(f.eval(x)) > error && k < n) {
    x = (x0*f.eval(x1) - x1*f.eval(x0)) / (f.eval(x1) - f.eval(x0));
    cadena = "\n" + x0 + "\t\t" + x1 + "\t\t" + x + "\t\t" + f.eval(x);
    iteraciones.add(cadena);
    if(f.eval(x0)*f.eval(x) < 0) {
        x1 = x;
    }
    else {
        x0 = x;
    }
    k++;
}
```

Aunque es técnicamente correcto asignar parámetros desde dentro del cuerpo del método, reduce la legibilidad del código porque los desarrolladores no podrán saber si se está accediendo al parámetro original o a alguna variable temporal sin pasar por todo el método. Por otra parte también se podrían esperar asignaciones de parámetros del método para ser visible, por lo que puede inducir a confusión. En su lugar, todos los parámetros, excepciones captadas y parámetros “foreach” deben ser tratados como finales.



Aunque puede resultar redundante y menos óptimo debería crear dos variables locales `r0` y `r1` que sustituyan los valores de `x0` y `x1`:

```
public double calcularRaiz(Funcion f, double x0, double x1, int n, double error) {
    double r = Double.NaN;
    double x = x0;
    int k = 0;
    double r0 = x0;
    double r1 = x1;
    String cadena = "Xizq\t\t\tXder\t\t\tX\t\t\tF(x)";
    iteraciones.add(cadena);

    while(Math.abs(f.eval(x)) > error && k < n) {
        x = (r1*f.eval(r1) - r1*f.eval(r0)) / (f.eval(r1) - f.eval(r0));
        cadena = "\n" + r0 + "\t\t" + r1 + "\t\t" + x + "\t\t" + f.eval(x);
        iteraciones.add(cadena);
        if(f.eval(r0)*f.eval(x) < 0) {
            r1 = x;
        }
        else {
            r0 = x;
        }
        k++;
    }
    if(k < n) {
        r = x;
    }
}
```

13ª Regla

Abstract classes without fields should be converted to interfaces (squid:S1610)

 Code smell  Minor

```
package actividad04.RaizPolinomio;

public abstract class Funcion {

    public abstract double eval(double x);

}
```

Cualquier clase abstracta sin campo directo o heredado debe convertirse en una interfaz.

```
package actividad04.RaizPolinomio;

interface Funcion {

    public abstract double eval(double x);

}
```

Además la clase Polinomio debe implementar dicha interfaz y sobrescribir el método “eval”:

```
@Override
public double eval(double x) {
    double valor = 0;
    for (Monomio m: pol) {
        valor = valor + m.getCoficiente() * Math.pow(x, m.getExponente());
    }
    return valor;
}
```

14ª Regla

Printf-style format strings should not lead to unexpected behavior at runtime (squid:S2275)

 Bug  Major

```
out.printf("Iteraciones\n" + fP.getIteraciones());
```

Debido a que las cadenas de formato de estilo printf se interpretan en tiempo de ejecución, en lugar de validadas por el compilador de Java, pueden contener errores que provocan un comportamiento inesperado o errores de tiempo de ejecución. Esta regla valida de forma estática el buen comportamiento de los formatos printf al llamar al método `format(...)` de las clases `java.util.Formatter`, `java.lang.String`, `java.io.PrintStream`, `MessageFormat` y `java.io.PrintWriter`. Y los métodos `printf (...)` de las clases `java.io.PrintStream` o `java.io.PrintWriter`.

Debemos utilizar formato

```
String.format("Iteraciones\n", fP.getIteraciones());
```


15ª Regla

Return values should not be ignored when function calls don't have any side effects (squid:S2201)

 Bug  Major

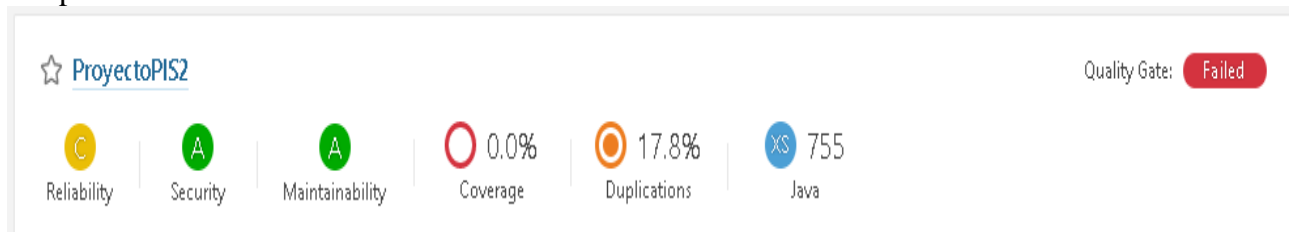
```
String.format("Iteraciones%n", fP.getIteraciones());
```

Cuando la llamada a una función no tiene efectos secundarios, ¿Para que hacer la llamada si se ignoran los resultados? En tal caso, o bien la llamada a la función es inútil y debe ser eliminado o el código fuente no se comporta como se esperaba.

La mejor forma de solucionar este problema es mediante el uso de StringBuilder:

```
StringBuilder s = new StringBuilder();
s.append(fP.getIteraciones());
String cadena = String.format("%s", s.toString());
out.println(cadena);
```

Tras la resolución de la factorización de estas reglas el proyecto mejora, los comprobamos con sonar-runner:



Aun así el proyecto sigue sin pasar las puertas de calidad, por lo que habrá que inspeccionar cada apartado con la intención de mejorarlo.

Los principales bugs se encontraban a la hora de definir los Monomios como “null”, al cambiar este problema adquirimos un grado de A en Reliability.

```
//Polinomio 1
Polinomio p = new Polinomio();
Monomio m;
//Polinomio 2
Polinomio p2 = new Polinomio();
Monomio m2;
```

