Actividad 08

CURSO 2016-2017

<Actividad08>

<FRANCISCO JAVIER MARQUÉS GAONA>

PROCESOS DE LA INGENIERÍA DEL SOFTWARE II

4º GRADO EN INGENIERÍA INFORMÁTICA

Departamento de Informática
Universidad de Almería

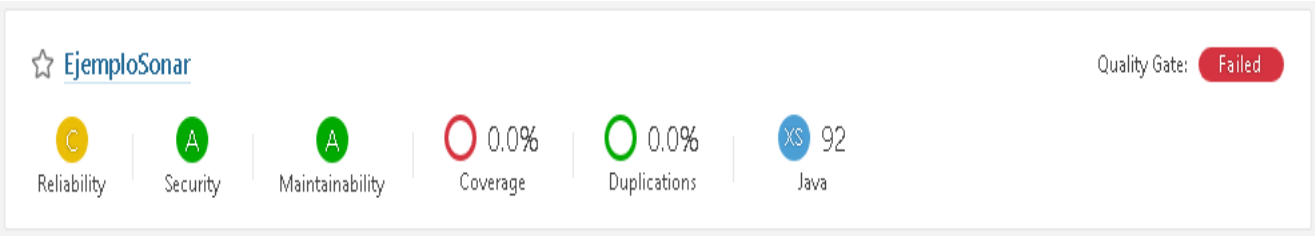
ÍNDICE

APRENDIZAJE REFACTORIZACIÓN

Tal y como se describe en la actividad se seguirán los pasos del primer capítulo del libro de Fowler respecto a un ejemplo de refactorización.

0. Punto inicial

En primer lugar creamos las clases tal y como se describen en el libro de Fowler y a continuación analizamos el proyecto con sonar runner para ver el estado inicial del proyecto.



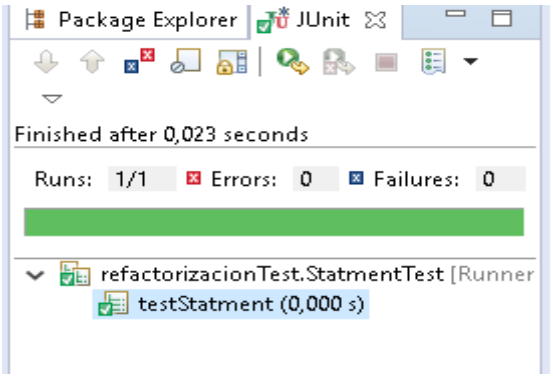
Como podemos apreciar en primer lugar el proyecto no es capaz de pasar nuestra puerta de calidad.

1. Primer paso: Creación de casos de prueba para comprobar el funcionamiento del método "statment()"

El siguiente pasos es crear un caso de prueba que compruebe el funcionamiento del método "statment()", para ello simplemente creamos una serie de clientes que tienen unas cuantas películas en alquiler y debemos comprobar que el string que crea el método "statment()" es igual al que nosotros hemos comprobado de forma manual.

Movie.java | Rental.java | Customer.java | StatmentTest.java

```
14      public void testStatment() {
15          Customer customer = new Customer("Fran");
16          Customer customer2 = new Customer("Pepe");
17          Customer customer3 = new Customer("Lucia");
18
19          Movie pel1 = new Movie("Terminator", 0);
20          Movie pel2 = new Movie("Matrix II", 1);
21          Movie pel3 = new Movie("Intocable", 2);
22
23          Rental rental1 = new Rental(pel1, 5);
24          Rental rental2 = new Rental(pel2, 7);
25          Rental rental3 = new Rental(pel3, 10);
26          customer.addRental(rental1);
27          customer.addRental(rental2);
28          customer.addRental(rental3);
29
30          customer2.addRental(rental1);
31          customer2.addRental(rental2);
32          customer3.addRental(rental3);
33
34          String stat1 = customer.statment();
35          String stat2 = customer2.statment();
36          String stat3 = customer3.statment();
37
38          assertEquals(stat1, "Rental Record for Fran\n\tTerminator\t6.5\n"
39                  + "\tMatrix II\t21.0\n\tIntocable\t12.0\n"
40                  + "Amount owed is 39.5\nYou earned 4 frequent renter points");
41
42          assertEquals(stat2, "Rental Record for Pepe\n\tTerminator\t6.5\n"
43                  + "\tMatrix II\t21.0\n"
44                  + "Amount owed is 27.5\nYou earned 3 frequent renter points");
45
46          assertEquals(stat3, "Rental Record for Lucia\n\tIntocable\t12.0\n"
47                  + "Amount owed is 12.0\nYou earned 1 frequent renter points");
```



2. Segundo Paso: Descomposición y redistribución del método de "statment"

El siguiente paso es la extracción del switch que calcula el precio de cada alquiler de películas en un método externo para mejorar la comprensión y a su vez simplificar el método "statment".

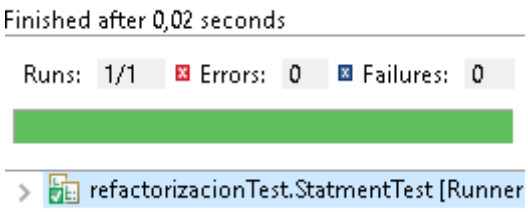
Extracción en el denominado método "amountFor"

```
private double amountFor(Rental aRental) {
    double thisAmount = 0;
    switch (aRental.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount += 2;
            if (aRental.getDaysRented() > 2) {
                thisAmount += (aRental.getDaysRented() - 2) * 1.5;
            break;
            }
        case Movie.NEW_RELEASE:
            thisAmount += aRental.getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (aRental.getDaysRented() > 3) {
                thisAmount += (aRental.getDaysRented() - 3) * 1.5;
            break;
            }
    }
    return thisAmount;
}
```

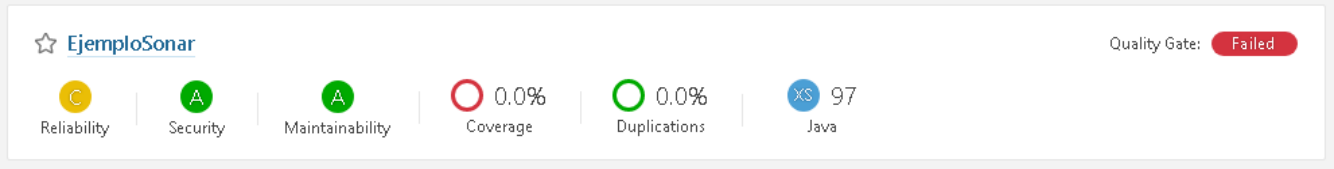
Simplificación de "statment"

```
        while (rentals.hasMoreElements()) {
            double thisAmount = 0;
            Rental each = (Rental) rentals.nextElement();

            //determine amounts for each line
            thisAmount = amountFor(each);
```

Como podemos comprobar, tras la extracción de dicho switch en un método externo el programa sigue pasando el test:



Comprobamos a continuación el estado de nuestro proyecto con sonar runner y el resultado es el siguiente:



3. Tercer Paso: Moviendo el cálculo de la suma

El cuarto paso es mover el cálculo de la suma del alquiler de películas de la clase "Customer" a la clase "Rental", para ello utilizamos "Move Method" y lo denominamos como getCharge(). Obtendríamos un método público que se ubica en la clase "Rental" tal y como deseamos:

```
double getCharge() {
    double result = 0;
    switch (getMovie().getPriceCode()) {
        case Movie.REGULAR:
            result += 2;
            if (getDaysRented() > 2) {
                result += (getDaysRented() - 2) * 1.5;
            break;
            }
        case Movie.NEW_RELEASE:
            result += getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            result += 1.5;
            if (getDaysRented() > 3) {
                result += (getDaysRented() - 3) * 1.5;
            break;
            }
    }
    return result;
}
```

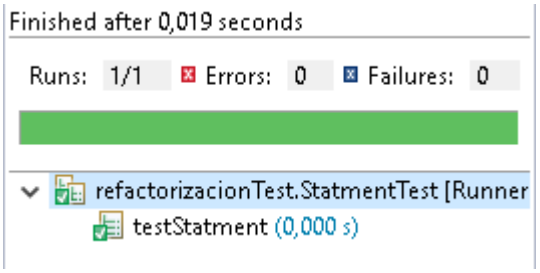
Mientras que en nuestra clase "Customer" se nos quedaría el siguiente método:

```
private double amountFor(Rental aRental) {
    return aRental.getCharge();
}
```

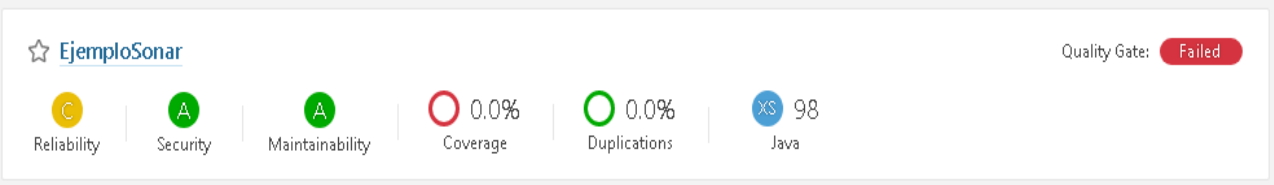
Así mismo en el método “statment” podemos intercambiar thisAmount por el resultado de each.getCharge():

```
        ,
        //show figures for this rental
        result += "\t" + each.getMovie().getTitle()+ "\t" +
                String.valueOf(each.getCharge()) + "\n";
        totalAmount += each.getCharge();
    }
```

Volvemos a comprobar que todo funciona correctamente tras los cambios realizados:



Y también comprobamos el estado del proyecto, que como podemos observar no varía en gran medida:



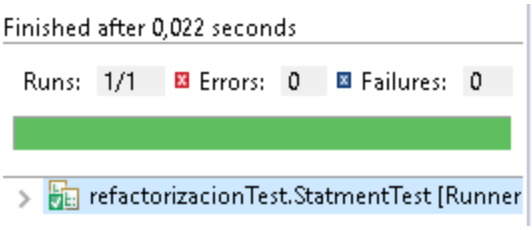
4. Cuarto Paso: Extracción de “frequent renter points”

El siguiente paso sería la extracción del cálculo de “frequent renter points” en un método. Por otra parte dicho método se lleva a la clase “Rental” ya que en el método statment obtendremos el total de “frequent renter points” mediante el uso de each y el método creado como “getFrequentRenterPoints()” tal y como se muestra a continuación:

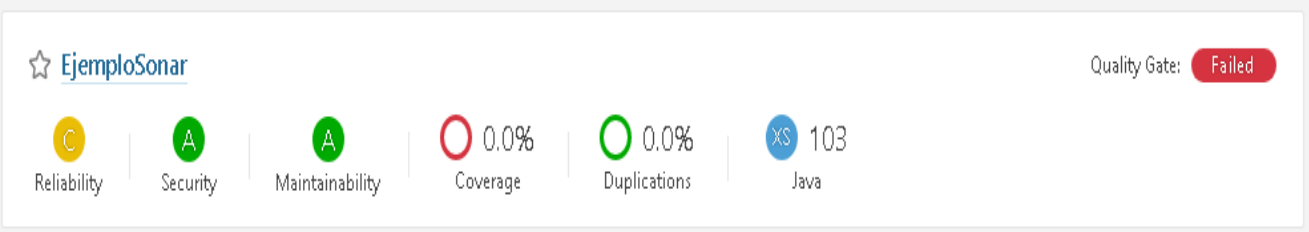
```
    int getFrequentRenterPoints() {
        if ((getMovie().getPriceCode() == Movie.NEW_RELEASE)
                && getDaysRented() > 1) {
            return 2;
        }
        else {
            return 1;
        }
    }
```

```
while (rentals.hasMoreElements()) {
    Rental each = (Rental) rentals.nextElement();
    // add frequent renter points
    frequentRenterPoints += each.getFrequentRenterPoints();

    //show figures for this rental
    result += "\t" + each.getMovie().getTitle()+ "\t" +
            String.valueOf(each.getCharge()) + "\n";
    totalAmount += each.getCharge();
}
```

Todo correcto como siempre:



Y el estado de nuestro proyecto es el siguiente tras realizar los cambios:



5. Quinto Paso: Eliminación de Temps

En este paso eliminamos las variables temporales para que estas se obtengan mediante el uso de un método. Las variables temporales que vamos a eliminar son "totalAmount" y "frequentRenterPoints". Extraemos éstas variables en un método mediante "Extract Method":

```
//add footer lines
result += "Amount owed is " + String.valueOf(getTotalCharge()) +
        "\n";
result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) +
        " frequent renter points";
return result;
```

Los métodos serían los siguientes:

```
private double getTotalCharge() {
    double result = 0;
    Enumeration rentals = _rentals.elements();
    while(rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        result += each.getCharge();
    }
    return result;
}

private int getTotalFrequentRenterPoints() {
    int result = 0;
    Enumeration rentals = _rentals.elements();
    while(rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();
        result += each.getFrequentRenterPoints();
    }
    return result;
}
```

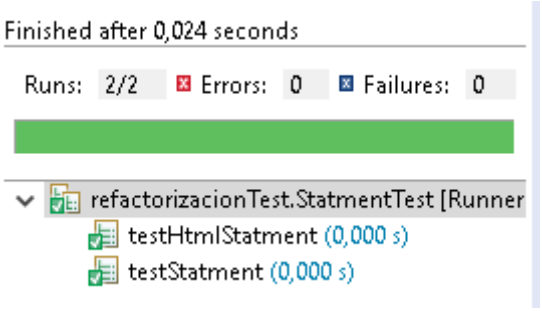
Al realizar este cambio incorporamos un problema fundamental y es que aumentamos la cantidad de código. Antes de realizar la refactorización el bucle "while" se reproducía solo una vez sin embargo ahora re realiza tres veces. Podemos crear un nuevo método denominado "htmlStatment" para solucionar el problema:

```
public String htmlStatment() {
    Enumeration rentals = _rentals.elements();
    String result = "<H1>Rentals for <EM>" + getName() + "</EM></H1><P>\n";
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();

        //show figures for this rental
        result += each.getMovie().getTitle()+ ": " +
                String.valueOf(each.getCharge()) + "<BR>\n";
    }
    //add footer lines
    result += "<P>You owe <EM>" + String.valueOf(getTotalCharge()) +
            "</EM><P>\n";
    result += "On this rental you earned <EM>" + String.valueOf(getTotalFrequentRenterPoints()) +
            "</EM> frequent renter points<P>";
    return result;
}
```

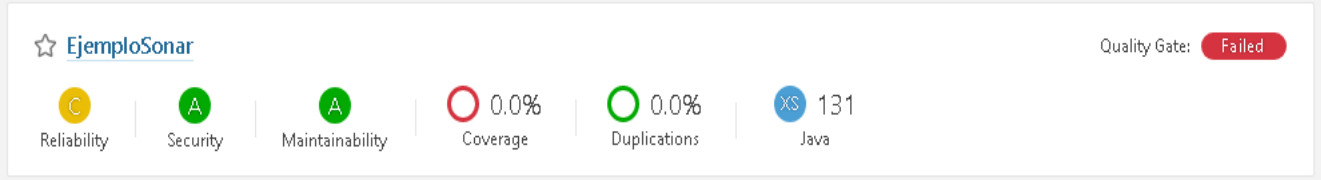
También deberíamos crear un nuevo test para dicho método y comprobar su funcionamiento:

```
public void testHtmlStatment() {
    Customer customer = new Customer("Fran");
    Customer customer2 = new Customer("Pepe");
    Customer customer3 = new Customer("Lucia");

    Movie pel1 = new Movie("Terminator", 0);
    Movie pel2 = new Movie("Matrix II", 1);
    Movie pel3 = new Movie("Intocable", 2);

    Rental rental1 = new Rental(pel1, 5);
    Rental rental2 = new Rental(pel2, 7);
    Rental rental3 = new Rental(pel3, 10);
    customer.addRental(rental1);
    customer.addRental(rental2);
    customer.addRental(rental3);

    customer2.addRental(rental1);
    customer2.addRental(rental2);
    customer3.addRental(rental3);

    String stat1 = customer.htmlStatment();
    String stat2 = customer2.htmlStatment();
    String stat3 = customer3.htmlStatment();
    System.out.println(stat1);
    System.out.println(stat2);
    System.out.println(stat3);

    assertEquals(stat1, "<H1>Rentals for <EM>Fran</EM></H1><P>\nTerminator: 6.5<BR>\n"
            + "Matrix II: 21.0<BR>\nIntocable: 12.0<BR>\n"
            + "<P>You owe <EM>39.5</EM><P>\nOn this rental you earned <EM>4</EM> frequent renter points<P>");

    assertEquals(stat2, "<H1>Rentals for <EM>Pepe</EM></H1><P>\nTerminator: 6.5<BR>\n"
            + "Matrix II: 21.0<BR>\n"
            + "<P>You owe <EM>27.5</EM><P>\nOn this rental you earned <EM>3</EM> frequent renter points<P>");

    assertEquals(stat3, "<H1>Rentals for <EM>Lucia</EM></H1><P>\n"
            + "Intocable: 12.0<BR>\n"
            + "<P>You owe <EM>12.0</EM><P>\nOn this rental you earned <EM>1</EM> frequent renter points<P>");
}
```

Comprobamos si pasa el test correctamente:



Comprobamos el estado de nuestro proyecto y como es lógico hemos aumentado el tamaño del código ya que por el momento tenemos ahora dos métodos **statment**: "statment" y "htmlStatment".



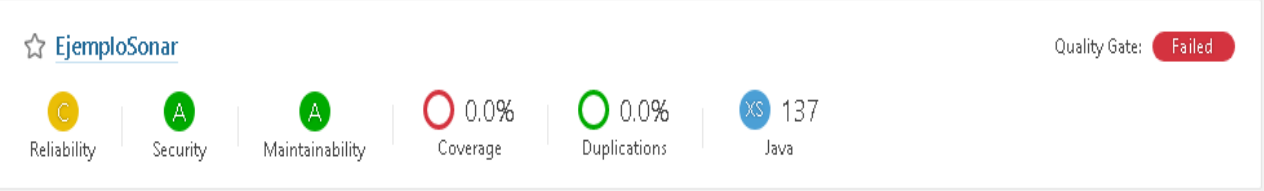
6. Sexto Paso: Sustitución de la lógica condicional de "Price Code" con polimorfismo

En este paso simplemente tenemos que mover los métodos "getCharge" y "getFrequentRenterPoints" de la clase "Rental" a "Movie" ya que si se desea usar una instrucción switch, esta debe usar sus propios datos, no los datos de otra clase. Quedaría de la siguiente forma:

Clase Rental

```
double getCharge() {
    return _movie.getCharge(_daysRented);
}

int getFrequentRenterPoints() {
    return _movie.getFrequentRenterPoints(_daysRented);
}
```

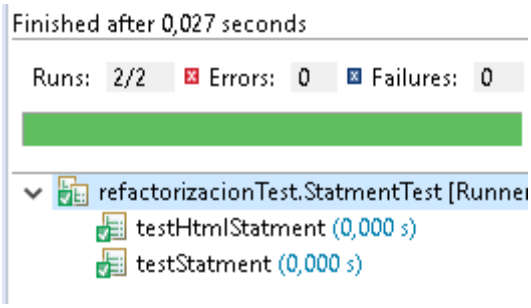
El estado del proyecto se muestra a continuación:



Clase Movie

```
double getCharge(int daysRented) {
    double result = 0;
    switch (getPriceCode()) {
        case Movie.REGULAR:
            result += 2;
            if (daysRented > 2) {
                result += (daysRented - 2) * 1.5;
            }
            break;
        case Movie.NEW_RELEASE:
            result += daysRented * 3;
            break;
        case Movie.CHILDRENS:
            result += 1.5;
            if (daysRented > 3) {
                result += (daysRented - 3) * 1.5;
            break;
            }
    }
    return result;
}

 int getFrequentRenterPoints(int daysRented) {
     if ((getPriceCode() == Movie.NEW_RELEASE)
             && daysRented > 1) {
         return 2;
     }
     else {
         return 1;
     }
 }
```

Comprobación del test:



7. Séptimo Paso: Herencia

Finalmente vamos a crear cuatro nuevas clases: “Price” (que será una clase abstracta que contiene los métodos que deben implementar las demás clases), “ChildrenPrice” (extiende de Price), “RegularPrice” (que extiende de Price) y “NewReleasePrice” (también extiende de Price).

Clase Price

```
package refactorizacion;

public abstract class Price {

    abstract int getPriceCode();

    abstract double getCharge(int daysRented);

    int getFrequentRenterPoints(int daysRented) {
        return 1;
    }
}
```

Clase ChildrenPrice

```
package refactorizacion;

public class ChildrensPrice extends Price {

    @Override
    int getPriceCode() {
        return Movie.CHILDRENS;
    }

    @Override
    double getCharge(int daysRented) {
        double result = 1.5;
        if (daysRented > 3) {
            result += (daysRented - 3) * 1.5;
        }
        return result;
    }
}
```

Clase NewReleasePrice

```
1  package refactorizacion;
2
3  public class NewReleasePrice extends Price {
4
5      @Override
6      int getPriceCode() {
7          return Movie.NEW_RELEASE;
8      }
9
0      @Override
1      double getCharge(int daysRented) {
2          return daysRented * 3;
3      }
4
5      @Override
6      int getFrequentRenterPoints(int daysRented) {
7          return (daysRented > 1) ? 2: 1;
8      }
9  }
```

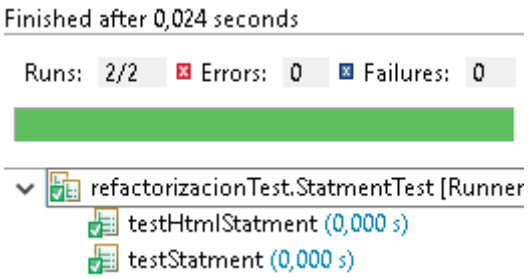
Clase RegularPrice

```
1  package refactorizacion;
2
3  public class RegularPrice extends Price {
4
5      @Override
6      int getPriceCode() {
7          return Movie.REGULAR;
8      }
9
10     @Override
11     double getCharge(int daysRented) {
12         double result = 2;
13         if (daysRented > 2) {
14             result += (daysRented - 2) * 1.5;
15         }
16         return result;
17     }
18 }
```

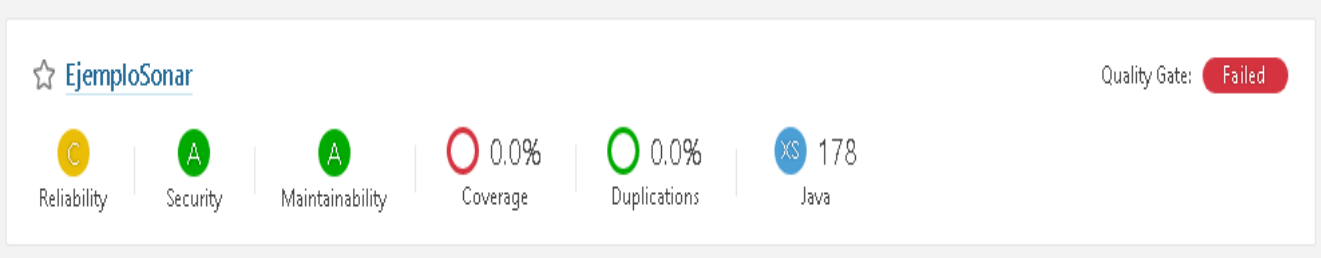
Estos cambios es para simplificar el cálculo de "getCharge" y "getFrequentRenterPoints" derivando éste cálculo a cada uno de los tipos de "Price"

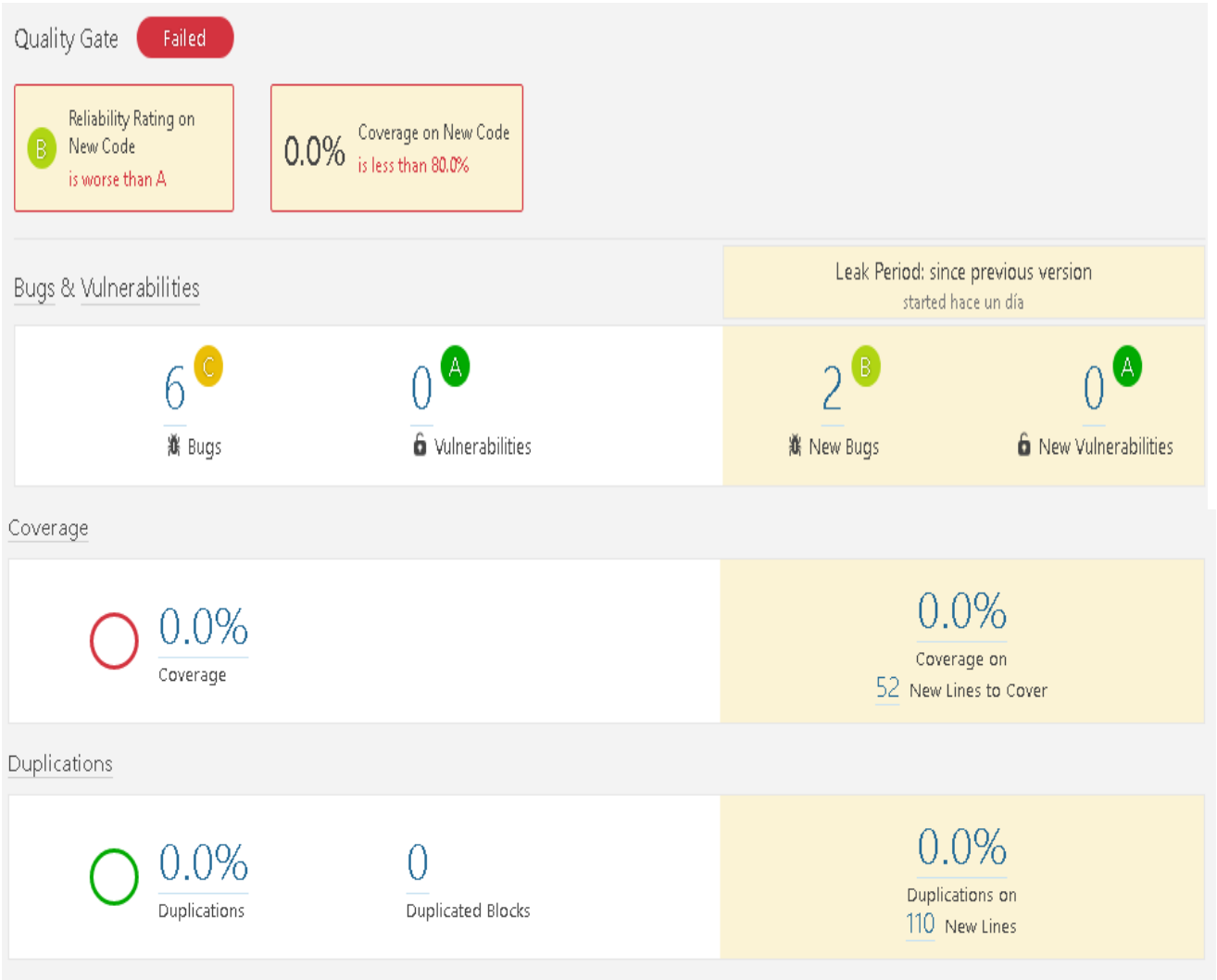
```
    double getCharge(int daysRented) {
        return _price.getCharge(daysRented);
    }

    int getFrequentRenterPoints(int daysRented) {
        return _price.getFrequentRenterPoints(daysRented);
    }
}
```

Comprobamos que pasa los tests correctamente:



Finalmente observamos el estado de nuestro proyecto con las herramientas sonar runner y sonarQube.




Quality Gate
Failed
B
Reliability Rating on New Code
is worse than A
0.0%
Coverage on New Code
is less than 80.0%
Bugs & Vulnerabilities
Leak Period: since previous version
started hace un día
6
C
Bugs
0
A
Vulnerabilities
2
B
New Bugs
0
A
New Vulnerabilities
Coverage
0.0%
Coverage
0.0%
Coverage on
52
New Lines to Cover
Duplications
0.0%
Duplications
0
Duplicated Blocks
0.0%
Duplications on
110
New Lines