***Método 2. Conversión de números romanos a base diez.***

1. Parámetros del SUT:

Como parámetro recibimos un String s que se corresponde con un número romano.

2. Caracterizaciones

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Caracterización** | **b1** | **b2** | **b3** | **b4** | **b5** |
| Orden de los símbolos | Símbolos ordenados de mayor a menor valor | Símbolos no repetidos desordenados | Símbolos repetidos desordenados | Símbolos correctos, pero no cumplen las reglas de numeración | símbolos incorrectos |
| Contiene “I” | True | False | Inválido |  |  |
| Contiene “V” | True | False | Inválido |  |  |
| Contiene “X” | True | False | Inválido |  |  |
| Contiene “L” | True | False | Inválido |  |  |
| Contiene “C” | True | False | Inválido |  |  |
| Contiene “D” | True | False | Inválido |  |  |
| Contiene “M” | True | False | Inválido |  |  |

3. Combinación de los bloques

Como tenemos 7 caracterizaciones con 3 bloques y 1 caracterización con 5 bloques, tenemos que llegar a una solución de compromiso entre el número de tests a desarrollar y el porcentaje de funcionalidad testeada.

En vez de utilizar el criterio de cobertura *base-choice,* vamos a utilizar el *multiple* *base choice.* Esto se debe a que la cantidad de situaciones de test que tenemos que simular van más allá de pequeñas variaciones sobre un caso base. Tenemos especial interés en testear que se cumplen todas las posibles reglas de la numeración romana.

4. Identificación de restricciones

Existen combinaciones de bloques que carecen de sentido o que son imposibles:

Símbolos no repetidos desordenados que solo contenga 1 símbolo.

Símbolos repetidos desordenados que solo contenga 1 símbolo.

No puede haber casos de test donde no se contenga ningún símbolo (todos *false*).

Etc.

5. Creando los casos de prueba:

Haciendo las combinaciones entre bloques, incluyendo las restricciones del problema y eliminando casos de test (ya sea porque son redundantes o por compromiso con el tiempo de dedicación de la práctica), implementaremos los siguientes tests.

Para escoger cada caso no solo hemos combinado los bloques, sino que además hemos revisando una por una las reglas de la numeración romana, asegurando que hay al menos un caso de test relacionado con cada regla.

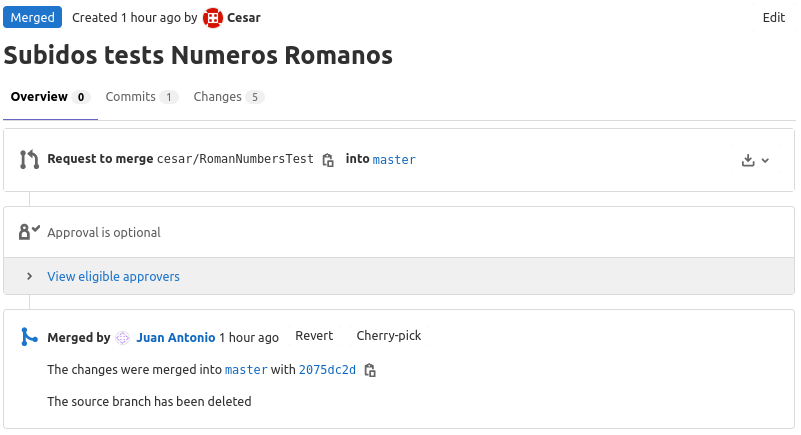
|  |  |  |
| --- | --- | --- |
| N.º test | Combinación de bloques | Test |
| 1 | Símbolos ordenados de mayor a menor valor + Contiene (I,V,X,L,C,D,M) | MDCLXVI |
| 2 | Símbolos no repetidos desordenados + Contiene (I,V,X) | XIV |
| 3 | Símbolos no repetidos desordenados + Contiene (I,X) | IX |
| 4 | Símbolos no repetidos desordenados + Contiene (I,X,L) | XLI |
| 5 | Símbolos no repetidos desordenados + Contiene (C,M) | CM |
| 6 | Símbolos no repetidos desordenados + Contiene (I, C, D) | CDI |
| 7 | Símbolos repetidos desordenados + Contiene (I,V) | XXIV |
| 8 | Símbolos repetidos desordenados + Contiene (I,X) | XIX |
| 9 | Símbolos repetidos desordenados + Contiene (C,D,I,V) | CDIV |
| 10 | Símbolos repetidos desordenados + Contiene (D,C,X,V) | DCXXV |
| 11 | Símbolos repetidos desordenados + Contiene (M,C) | MCM |
| 12 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (I) | IIII |
| 13 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (X) | XXXX |
| 14 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (C,X,I) | CCCCXXI |
| 15 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (C, M, D) | CMDC |
| 16 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (I, V) | VVII |
| 17 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (L,V) | LLV |
| 18 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (D, I) | DDDIII |
| 19 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (I,M) | IM |
| 20 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (X,D) | XD |
| 21 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (C,L) | LC |
| 22 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (V,D) | VD |
| 23 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (L,X) | LXL |
| 24 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (C,M) | CCM |
| 25 | Símbolos correctos, pero no cumplen las reglas de numeración + Contiene (X,C,D) | XCD |
| 26 | Símbolos incorrectos | AUYB |
| 27 | Símbolos incorrectos + Contiene V | VOP |

6. Escribir los tests en Java

Primero hacemos un *git pull origin master* del repositorio de nuestro proyecto. Creamos una nueva rama local para trabajar: *cesar/RomanNumbersTest.*

Programamos los test en Java en un fichero *RomanNumbersTest.* Luego, metemos este fichero en una carpeta denominada *RomanNumbers,* hacemos un commit y un *push origin cesar/RomanNumbersTest* y solicitamos un *merge request* a *master* para subir la carpeta con el fichero de tests. Le asignamos específicamente el merge request al compañero que va a programar el SUT del método 2. (<https://gitlab.etsit.urjc.es/grupo-isi/practica-isi/-/merge_requests/9>).

Cuando nos acepta el merge request, la rama *cesar/RomanNumbersTest* se elimina.



7. Ejecución de los tests:

En este apartado al ejecutar los test nos damos cuenta de que no se han cumplido todas las normas de los números romanos, pero gracias a los tests nos hemos dado cuenta de estos errores que hemos conseguido solucionar.

El hecho de habernos dado cuenta de que no cumplíamos todas las especificaciones de los números romanos viene motivado por el hecho de haber realizado los tests antes de realizar la implementación, ya que de haberlo realizado al revés es probable que nos hubiésemos visto influenciados por los happy paths y no hubiésemos detectado algunos.

***Método 3. Embotelladora***

1. Parámetros del SUT:

Los parámetros del SUT son el número de botellas pequeñas, el número de botellas grandes y el número total de litros que hay que embotellar, todos ellos como números enteros.

2. Caracterizaciones

|  |  |  |  |
| --- | --- | --- | --- |
| Caracterización | b1 | b2 | b3 |
| Se puede embotellar todo con N botellas grandes | A1 = True | A2 = False | A3 = Inválido |
| Se puede embotellar todo con M botellas pequeñas | B1 = True | B2 = False | B3 = Inválido |
| Se puede embotellar por una combinación de N botellas grandes y M botellas pequeñas | C1 = True | C2 = False | C3 = Inválido |
| No se puede embotellar porque no tenemos suficientes botellas | D1 = True | D2 = False | D3 = Inválido |
| No se puede embotellar porque el número de litros sea inválido (número negativo) | E1 = True | E2 = False | E3 = Inválido |

3. Combinación de los bloques

Para el caso de la embotelladora recurriremos al algoritmo base-choice puro, en el cual estableceremos un único caso base, a diferencia de como hacíamos en el apartado de los números romanos. Para nosotros el caso base va a ser aquel en el que se pueda embotellar por una combinación de botellas grandes y pequeñas (A2, B2, C1, D2, E2); hemos escogido este caso base porque es en el que salen más casos de tests que sean interesantes salen, aunque también hemos probado a realizar otros casos bases como que sólo se embotelle con botellas pequeñas o solo con botellas grandes.

4. Identificación de restricciones

Hay restricciones en los siguientes dos casos que nos salen con el algoritmo base-choice:ç

* A2, B2, C1, D1, E2: Porque si se puede embotellar con una combinación de N botellas grandes y M botellas pequeñas no es posible que no se pueda embotellar.
* A2, B2, C2, D2, E2: No puede darse el caso de que no se pueda embotellar por no tener ni las suficientes grandes ni las suficientes pequeñas ni una combinación entre ambas ni tampoco sea posible el caso de no ser embotelladas.

5. Creando los casos de prueba:

Para crear los casos de prueba en este caso nos hemos ayudado de una tabla con las combinaciones que nos han salido al realizar el algoritmo base-choice:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A2, B2, C1, D2, E2 | A2, B2, C1, D2, E1 | A2, B2, C1, D1, E2 | A2, B2, C2, D2, E2 | A2, B1, C1, D2, E2 | A1, B2, C1, D2, E2 |
|  | A2, B2, C1, D2, E3 | A2, B2, C1, D3, E2 | A2, B2, C3, D2, E2 | A2, B3, C1, D2, E2 | A3, B2, C1, D2, E2 |

Y cada uno de los bloques de esta tabla se corresponde con los siguientes casos de prueba:

|  |  |
| --- | --- |
| N.º test | Combinación de bloques |
| 1 | Se puede embotellar con una combinación de botellas grandes y pequeñas. |
| 2 | Se puede embotellar con una combinación de botellas, pero el número de litros es inválido. |
| 3 | Se puede llenar todo con botellas grandes y con una combinación de botellas. |
| 4 | Se puede llenar todo con botellas pequeñas y con una combinación de botellas. |
| 5 | Se podría llenar con una combinación de botellas, pero el número de litros es inválido. |
| 6 | El número de botellas que tenemos es inválido. |
| 7 | La combinación de botellas es inválida. |
| 8 | Se embotella con una combinación de botellas, pero el número de botellas grandes es un número invalido. |
| 9 | Se puede embotellar con una combinación de botellas, pero el número de botellas pequeñas es invalido. |

6. Escribir los tests en java:

Al igual que en el apartado anterior hacemos un git pull origin master del repositorio de nuestro proyecto. Creamos una nueva rama local: ja.ortega.2017/ramaEmbotelladora

Programamos los tests en un fichero con nombre EmbotelladoraTest dentro del directorio Embotelladora, donde además se implementará el código de este ejercicio. Hacemos un commit y un push origin ja.ortega.2017/Embotelladora y solicitamos un merge request para poder subir el fichero con los tests como hicimos en los apartado asnteriores. La URL del merge request es : https://gitlab.etsit.urjc.es/grupo-isi/practica-isi/-/merge\_requests/8.

Una vez se acepte el merge request, la rama creada se eliminará.

7. Ejecución de los tests:

Aceptamos el *merge request de* la rama local que está utilizando el compañero a *master (*[*https://gitlab.etsit.urjc.es/grupo-isi/practica-isi/-/merge\_requests/8*](https://gitlab.etsit.urjc.es/grupo-isi/practica-isi/-/merge_requests/8)*).*

Lo que quiere subir el compañero son los tests para testear el método 3 (Embotelladora). Para ello, ha creado en una rama local una carpeta *Embotelladora* donde está todo el código de los tests.

Un vez aceptado el merge, ya tenemos la carpeta *Embotelladora* en *origin/master*. Ahora lo que hacemos es un *pull origin master* en nuestro repositorio local y luego creamos una nueva rama local *cesar/EmbotelladoraTest.*

Es en esta ramadonde vamos a meter el código de la Embotelladora que hemos programado. Añadiremos a la carpeta *Embotelladora* el código de nuestro método (*NoSolution.java y NoSolution.java)* y los scripts necesarios para ejecutar los tests de junit desde la linea de comandos.

Antes de ejecutar los tests, lo primero que hacemos es comprobar que *EmbotelladoraTest.java* está correcto para poder ejecutarlo sin error.

Vemos que para que pueda ejecutarse correctamente, hemos tenido que cambiar algunas cosas.

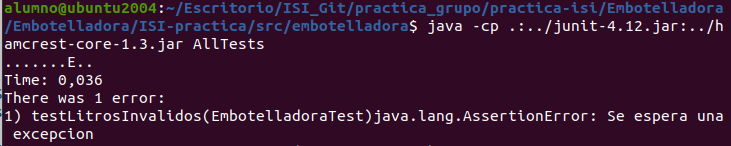
1) Hemos eliminado los comentarios que llevaban tilde o ñ porque al compilar los detecta como carácteres inválidos.

2) Hemos añadido alguna librería que hacía falta para ejecutar los tests.

3) Hemos eliminado el *package*, puesto que tanto el programador del SUT como el de los tests han utilizado el IDE de Eclipse, nombrando el paquete de forma diferente (por lo que había conflictos).

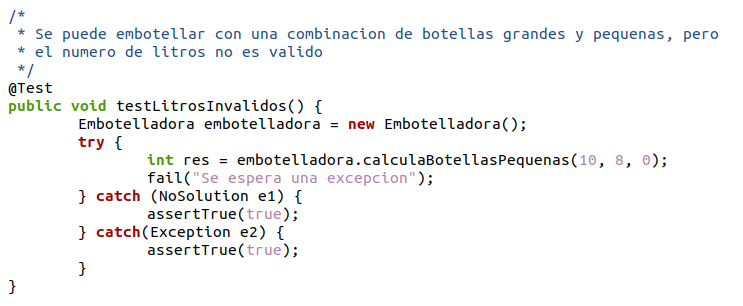
Una vez hecho esto, los tests ya pueden correr.

En el primer intento, nuestro SUT de la Embotelladora no pasa todos los tests. Vemos que tenemos un error en el antepenúltimo test, donde el mensaje de error nos dice que “se esperaba una excepción”.



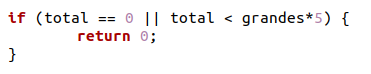
Vamos a observar cuál es el test que da error y por qué. Posteriormente veremos qué hay que modificar en el código para que no de error el test.

Este es el test que da error:



Vemos que, según el test, si la entrada del total = 0, debería saltar una excepción.

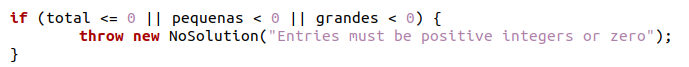
Vamos a ver cómo ejecuta esto nuestro código de la Embotelladora:



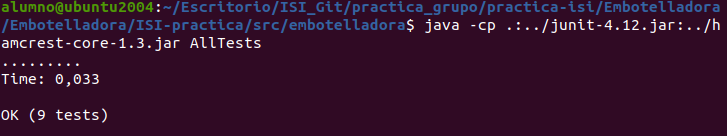
Vemos que en nuestro código, si total = 0, se retorna 0, en vez de hacer saltar una excepción.

¿Es realmente esto un error? El enunciado NO ESPECIFICA cómo se considera la entrada cuando total = 0. No sabemos si se trata de una entrada correcta y tiene que retornar 0, o por el contrario es incorrecta y hace saltar una excepción.

El programador de los test ha considerado que se tendría que hacer saltar una excepción, por lo que modificaremos nuestro SUT para que se comporte de esta manera.



Una vez modificado, corremos los test de nuevo para ver si esta modificación provoca el fallo de otros test o el código pasa de forma exitosa.



Como vemos, el SUT pasa todos los tests de forma exitosa. Hacemos un nuevo commit donde se refleje el cierre de la *issue #27.* Por último hacemos un *push origin cesar/EmbotelladoraTest.*

Hacemos un merge request de la rama *cesar/EmbotelladoraTest* a *master. (*[*https://gitlab.etsit.urjc.es/grupo-isi/practica-isi/-/merge\_requests/10*](https://gitlab.etsit.urjc.es/grupo-isi/practica-isi/-/merge_requests/10)*).* Vemos que quien acepta el merge request es el compañero que ha programado los tests de ese método.

Vemos que, cuando se hace el merge request, la rama *cesar/EmbotelladoraTest* desaparece. Ya he cumplido su misión.

