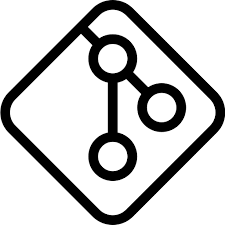
|  |
| --- |
|  |
| **Control de versiones con GIT** |
| TR13\_PaulaFernandezSuarez |

|  |
| --- |
| Paula Fernández Suárez |



[Configuración de GIT 2](#_Toc101113267)

[Creación del repositorio remoto 3](#_Toc101113268)

[Primera versión 4](#_Toc101113269)

[Código: 4](#_Toc101113270)

[Introducción del código en GIT local 6](#_Toc101113271)

[Añadir el proyecto al repositorio remoto 6](#_Toc101113272)

[Segunda versión 7](#_Toc101113273)

[Crear rama Develop en GIT 7](#_Toc101113274)

[Código tras añadir comentarios: 7](#_Toc101113275)

[Crear Javadoc 10](#_Toc101113276)

[Añadir cambios al repositorio local 11](#_Toc101113277)

[Optimización del código 11](#_Toc101113278)

[Pruebas unitarias 15](#_Toc101113279)

[Método ProductoEntero: 15](#_Toc101113280)

[Método CocienteEntero 18](#_Toc101113281)

[Método RestoEntero 21](#_Toc101113282)

[Método PotenciaEntera 24](#_Toc101113283)

[Método Factorial 27](#_Toc101113284)

[Método ProductoRuso 30](#_Toc101113285)

[Suite de pruebas 33](#_Toc101113286)

[Añadir los archivos al repositorio remoto 34](#_Toc101113287)

# Configuración de GIT

Antes de realizar el código y de comenzar a crear las versiones debemos configurar la aplicación que controla estas, en nuestro caso “GIT”.

Para ello debemos seguir los siguientes pasos:

1. Introducimos el comando ‘git’ para saber si el intérprete de comandos está operativo. Si nos sale el siguiente mensaje es que GIT está listo para funcionar y podemos continuar con la configuración.

$ git

usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

[--super-prefix=<path>] [--config-env=<name>=<envvar>]

<command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.

1. Para establecer en nombre de usuario y el correo electrónico debemos introducir los siguientes dos comandos:

$ git config --global user.name 'paula'

$ git config --global user.email 'paulafs35@educastur.es'

# Creación del repositorio remoto

Para esta práctica he decidido crear un repositorio remoto. Para ello debemos ir al apartado de repositorio en la página web de git (si no tienes una cuenta debes crearla previamente). A continuación, hacemos click sobre “New” y lo configuramos a según nuestras necesidades.

Imagen que contiene Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Primera versión

Para la primera versión creamos un proyecto en Eclipse y le introducimos únicamente el código de este, sin comentarios ni las pruebas unitarias de Junit.

## Código:

**package** OperacionesMatematicas;

**public** **class** OperacionesMatematicas

{

**public** **int** ProductoEntero(**int** n1, **int** n2)

{

**int** producto;

**if**((n1 < 0) || (n2 <= 0))

producto = -1;

**else**

{

producto = 0;

**if**(n2 > 0)

{

**while**(n1 > 0)

{

producto = producto + n2;

n1--;

}

}

}

**return** producto;

}

**public** **int** CocienteEntero(**int** dividendo, **int** divisor)

{

**int** cociente;

**if**((dividendo < 0) || (divisor <= 0))

cociente = -1;

**else**

{

cociente = 0;

**while**(dividendo >= divisor)

{

cociente++;

dividendo = dividendo - divisor;

}

}

**return** cociente;

}

**public** **int** RestoEntero(**int** dividendo, **int** divisor)

{

**if**((dividendo < 0) || (divisor <= 0))

dividendo = -1;

**else**

{

**while**(dividendo >= divisor)

{

dividendo = dividendo - divisor;

}

}

**return** dividendo;

}

**public** **int** PotenciaEntera(**int** base, **int** exponente)

{

**int** potencia;

**if**((base < 0) || (exponente < 0))

potencia = -1;

**else**

{

potencia = 1;

**while**(exponente > 0)

{

potencia = potencia \* base;

exponente = exponente -1;

}

}

**return** potencia;

}

**public** **int** Factorial(**int** numero)

{

**int** factorial;

**if**(numero < 0)

factorial = -1;

**else**

{

factorial = 1;

**while**(numero > 1)

{

factorial = factorial \* numero;

numero--;

}

}

**return** factorial;

}

**public** **int** ProductoRuso(**int** factor1, **int** factor2)

{

**int** solucion;

**if**((factor1 < 0) || (factor2 < 2))

solucion = -1;

**else**

{

solucion = 0;

**if**(factor1 > 0)

{

**while**(factor2 > 0)

{

**if**(factor2 % 2 != 0)

solucion = solucion + factor1;

factor1 = factor1 \* 2;

factor2 = factor2 / 2;

}

}

}

**return** solucion;

}

}

## Introducción del código en GIT local

Para introducir el código en GIT primero debemos guardar los cambios realizados. A continuación, nos situamos en la carpeta que nos corresponde dentro del intérprete de comandos utilizando la instrucción:

$ cd /c/Users/Paula/OneDrive/Escritorio/TR13\_PaulaFernandezSuarez

Después inicializamos GIT dentro de la carpeta.

$ git init

Initialized empty Git repository in C:/Users/Paula/OneDrive/Escritorio/TR13\_PaulaFernandezSuarez/.git/

Por último, añadimos los cambios a la zona de índices y, finalmente al repositorio local, junto con un comentario indicando que se ha realizado.

$ git add --all

$ git commit -m 'Añadimos el código inicial, sin comentarios ni pruebas unitarias'

Al realizar esto debemos obtener el siguiente mensaje:

[master (root-commit) 246cd4d] Añadimos el código inicial, sin comentarios ni pruebas unitarias

7 files changed, 154 insertions(+)

create mode 100644 .classpath

create mode 100644 .project

create mode 100644 .settings/org.eclipse.jdt.core.prefs

create mode 100644 bin/OperacionesMatematicas/OperacionesMatematicas.class

create mode 100644 bin/module-info.class

create mode 100644 src/OperacionesMatematicas/OperacionesMatematicas.java

create mode 100644 src/module-info.java

## Añadir el proyecto al repositorio remoto

Para añadir el proyecto que acabamos de crear al repositorio remoto, debemos importar el proyecto a este. Para ello utilizaremos los siguientes comandos:

$ git remote add origin https://github.com/paulafs35/TR13\_PaulaFernandezSuarez.git

$ git branch -M main

$ git push -u origin main

Tras esto, GIT nos mostrará un mensaje indicando los cambios que se han realizado en nuestro repositorio remoto y en este aparecerá en qué versión nos encontramos.

Texto

Descripción generada automáticamente con confianza baja

# Segunda versión

Para esta versión añadiremos comentarios utilizando javadoc al código fuente, facilitando así reconocer qué se realiza en cada parte y qué es cada una de ellas.

## Crear rama Develop en GIT

Antes de modificar el código debemos crear una nueva rama dentro del proyecto en la que se almacenen los cambios sin que esto afecte a la rama principal. Para ello creamos una bifurcación de la misma llamada Develop, que nos permitirá trabajar sobre ella sin tocar el trabajo original.

Para crearla y cambiarnos a esta debemos introducir los siguientes comandos:

$ git branch Develop

$ git checkout Develop

Si se ha cambiado correctamente GIT nos mostrará un comentario y cambiará las letras finales, que se encuentran en color azul y entre paréntesis, al nombre de la nueva rama en la que nos encontramos.

Switched to branch 'Develop'

Paula@DESKTOP-3LCIL78 MINGW64 ~/OneDrive/Escritorio/TR13\_PaulaFernandezSuarez (Develop)

## Código tras añadir comentarios:

**package** OperacionesMatematicas;

/\*\*

\* Realiza diferentes operaciones y, si no se pueden realizar, se obtiene -1.

\* **@author** Paula Fernández Suárez

\* **@version** 1.2

\* **@since** 13/04/2022

\*/

**public** **class** OperacionesMatematicas

{

/\*\*

\* Método que calcula el producto obtenido tras multiplicar dos números enteros.

\* **@param** n1

\* **@param** n2

\* **@return** producto

\*/

**public** **int** ProductoEntero(**int** n1, **int** n2)

{// Declaramos las variables que vamos a utilizar

**int** producto;

// Comprobamos que los números sean válidos

**if**((n1 < 0) || (n2 <= 0))

producto = -1;

**else**

{

// Calculamos el producto

producto = 0;

**if**(n2 > 0)

{

**while**(n1 > 0)

{

producto = producto + n2;

n1--;

}

}

}

// Devolvemos el resultado obtenido

**return** producto;

}

/\*\*

\* Método que calcula el cociente obtenido tras dividir dos números enteros.

\* **@param** dividendo

\* **@param** divisor

\* **@return** cociente

\*/

**public** **int** CocienteEntero(**int** dividendo, **int** divisor)

{

// Declaramos las variables que vamos a utilizar

**int** cociente;

// Comprobamos si los valores son válidos

**if**((dividendo < 0) || (divisor <= 0))

cociente = -1;

**else**

{

// Calculamos el cociente

cociente = 0;

**while**(dividendo >= divisor)

{

cociente++;

dividendo = dividendo - divisor;

}

}

// Devolvemos el resultado obtenido

**return** cociente;

}

/\*\*

\* Método que calcula el resto obtenido tras dividir dos números enteros.

\* **@param** dividendo

\* **@param** divisor

\* **@return** dividendo

\*/

**public** **int** RestoEntero(**int** dividendo, **int** divisor)

{

// Comprobamos que los valores sean válidos

**if**((dividendo < 0) || (divisor <= 0))

dividendo = -1;

**else**

{

// Calculaos el resto

**while**(dividendo >= divisor)

{

dividendo = dividendo - divisor;

}

}

// Devolvemos el resultado obtenido.

**return** dividendo;

}

/\*\*

\* Método que calcula la potencia de dos números enteros.

\* **@param** base

\* **@param** exponente

\* **@return** potencia

\*/

**public** **int** PotenciaEntera(**int** base, **int** exponente)

{

// Declaramos las variables que vamos a utilizar

**int** potencia;

// Comprobamos si los valores son válidos

**if**((base < 0) || (exponente < 0))

potencia = -1;

**else**

{

// Calculamos la potencia

potencia = 1;

**while**(exponente > 0)

{

potencia = potencia \* base;

exponente = exponente -1;

}

}

// Devolvemos el resultado obtenido

**return** potencia;

}

/\*\*

\* Método que calcula el factorial de un número.

\* **@param** numero

\* **@return** factorial

\*/

**public** **int** Factorial(**int** numero)

{

// Declaramos las variables que vamos a utilizar

**int** factorial;

// Comprobamos si los valores son válidos

**if**(numero < 0)

factorial = -1;

**else**

{

// Calculamos el factorial

factorial = 1;

**while**(numero > 1)

{

factorial = factorial \* numero;

numero--;

}

}

// Devolvemos el resultado obtenido

**return** factorial;

}

/\*\*

\* Método que calcula el producto obtenido tras multiplicar dos números enteros utilizando el método ruso.

\* **@param** factor1

\* **@param** factor2

\* **@return** solucion

\*/

**public** **int** ProductoRuso(**int** factor1, **int** factor2)

{

// Declaramos las variables que vamos a utilizar

**int** solucion;

// Comprobamos si los valores son válidos

**if**((factor1 < 0) || (factor2 < 2))

solucion = -1;

**else**

{

// Calculamos el producto ruso

solucion = 0;

**if**(factor1 > 0)

{

**while**(factor2 > 0)

{

**if**(factor2 % 2 != 0)

solucion = solucion + factor1;

factor1 = factor1 \* 2;

factor2 = factor2 / 2;

}

}

}

// Devolvemos el resultado obtenido

**return** solucion;

}

}

## Crear Javadoc

Para crear el documento HTML de javadoc debemos hacer click en ‘Project’ y, dentro del menú que se despliega, ‘Generate javadoc’.

Texto

Descripción generada automáticamente

## Añadir cambios al repositorio local

Para añadir únicamente lo que ha sido modificado se utilizará el comando add junto con el path completo donde se encuentra el archivo y el nombre del archivo en sí.

$ git add /c/Users/Paula/OneDrive/Escritorio/TR13\_PaulaFernandezSuarez/src/OperacionesMatematicas/OperacionesMatematicas.java

$ git add /c/Users/Paula/OneDrive/Escritorio/TR13\_PaulaFernandezSuarez/doc

$ git commit -m 'Añadimos comentarios y generamos javadoc'

# Optimización del código

Antes de comenzar a realizar las pruebas consideré que el código se podía optimizar para que fuese más sencillo realizar las pruebas por lo que creé dos constructores para la clase, refactoricé las variables que se utilizaban en los métodos y las extraje como dos variables del mismo nombre.

El código quedaría así:

**package** OperacionesMatematicas;

/\*\*

\* Realiza diferentes operaciones y, si no se pueden realizar, se obtiene -1.

\* **@author** Paula Fernández Suárez

\* **@version** 1.2

\* **@since** 13/04/2022

\*/

**public** **class** OperacionesMatematicas

{

**int** n1, n2;

/\*\*

\* Constructor con dos parámetros enteros

\* **@param** n1

\* **@param** n2

\*/

**public** OperacionesMatematicas(**int** n1, **int** n2)

{

**this**.n1 = n1;

**this**.n2 = n2;

}

/\*\*

\* Constructor con un parámetro entero

\* **@param** n1

\*/

**public** OperacionesMatematicas(**int** n1)

{

**this**.n1 = n1;

}

/\*\*

\* Método que calcula el producto obtenido tras multiplicar dos números enteros.

\* <br> n1 = multiplicando

\* <br> n2 = multiplicador

\* **@return** producto

\*/

**public** **int** ProductoEntero()

{

// Declaramos las variables que vamos a utilizar

**int** producto;

// Comprobamos que los números sean válidos

**if**((n1 < 0) || (n2 <= 0))

producto = -1;

**else**

{

// Calculamos el producto

producto = 0;

**if**(n2 > 0)

{

**while**(n1 > 0)

{

producto = producto + n2;

n1--;

}

}

}

// Devolvemos el resultado obtenido

**return** producto;

}

/\*\*

\* Método que calcula el cociente obtenido tras dividir dos números enteros.

\* <br> n1 = dividendo

\* <br> n2 = divisor

\* **@return** cociente

\*/

**public** **int** CocienteEntero()

{

// Declaramos las variables que vams a utilizar

**int** cociente;

// Comprobamos si los valores son válidos

**if**((n1 < 0) || (n2 <= 0))

cociente = -1;

**else**

{

// Calculamos el cociente

cociente = 0;

**while**(n1 >= n2)

{

cociente++;

n1 = n1 - n2;

}

}

// Devolvemos el resultado obtenido

**return** cociente;

}

/\*\*

\* Método que calcula el resto obtenido tras dividir dos números enteros.

\* **@return** resto

\*/

**public** **int** RestoEntero()

{

// Declaramos las variables que vamos a utilizar

**int** resto;

// Comprobamos que los valores sean válidos

**if**((n1 < 0) || (n2 <= 0))

resto = -1;

**else**

{

resto = n1;

// Calculamos el resto

**while**(resto >= n2)

{

resto = resto - n2;

}

}

// Devolvemos el resultado obtenido.

**return** resto;

}

/\*\*

\* Método que calcula la potencia de dos números enteros.

\* <br> n1 = base

\* <br> n2 = exponente

\* **@return** potencia

\*/

**public** **int** PotenciaEntera()

{

// Declaramos las variables que vamos a utilizar

**int** potencia;

// Comprobamos si los valores son válidos

**if**((n1 < 0) || (n2 < 0))

potencia = -1;

**else**

{

// Calculamos la potencia

potencia = 1;

**while**(n2 > 0)

{

potencia = potencia \* n1;

n2 = n2 -1;

}

}

// Devolvemos el resultado obtenido

**return** potencia;

}

/\*\*

\* Método que calcula el factorial de un número.

\* <br> n1 = numero

\* **@return** factorial

\*/

**public** **int** Factorial()

{

// Declaramos las variables que vamos a utilizar

**int** factorial;

// Comprobamos si los valores son válidos

**if**(n1 < 0)

factorial = -1;

**else**

{

// Calculamos el factorial

factorial = 1;

**while**(n1 > 1)

{

factorial = factorial \* n1;

n1--;

}

}

// Devolvemos el resultado obtenido

**return** factorial;

}

/\*\*

\* Método que calcula el producto obtenido tras multiplicar dos números enteros utilizando el método ruso.

\* <br> n1 = multiplicando

\* <br> n2 = multiplicador

\* **@return** solucion

\*/

**public** **int** ProductoRuso()

{

// Declaramos las variables que vamos a utilizar

**int** solucion;

// Comprobamos si los valores son válidos

**if**((n1 < 0) || (n2 < 2))

solucion = -1;

**else**

{

// Calculamos el producto ruso

solucion = 0;

**if**(n1 > 0)

{

**while**(n2 > 0)

{

**if**(n2 % 2 != 0)

solucion = solucion + n1;

n1 = n1 \* 2;

n2 = n2 / 2;

}

}

}

// Devolvemos el resultado obtenido

**return** solucion;

}

}

# Pruebas unitarias

Para realizar las pruebas realicé caja blanca y caja negra de los métodos de manera individual y las fui añadiendo a git cada vez que terminaba una.

## Método ProductoEntero:

### Caja Blanca

#### Nodos

**public** **int** ProductoEntero()

{

1

**int** producto;

2

3

**if**((n1 < 0) || (n2 <= 0))

4

producto = -1;

**else**

{

5

producto = 0;

6

**if**(n2 > 0)

{

7

**while**(n1 > 0)

{

producto = producto + n2;

8

n1--;

}

}

}

9

**return** producto;

}

Diagrama

Descripción generada automáticamente

V1 = 5

V2 = 12 – 9 + 2 = 5

V3 = 4 + 1 = 5

#### Caminos:

1. 1 – 2 – 4 – 9
2. 1 – 3 – 4 – 9
3. 1 – 2 – 3 – 5 – 6 – 9
4. 1 – 2 – 3 – 5 – 6 – 7 – 9
5. 1 – 2 – 3 – 5 – 6 – 7 – 8 – 7 – … – 9

#### Pruebas:

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | n1 | n2 | producto |
| 1 | -1 | 8 | -1 |
| 2 | 3 | -1 | -1 |
| 3 | 3 | 0 | 0 |
| 4 | 0 | 3 | 0 |
| 5 | ~~3~~  ~~2~~  ~~1~~  0 | 3 | ~~0~~  ~~3~~  ~~6~~  9 |

### Caja Negra

#### Nodos

|  |  |  |  |
| --- | --- | --- | --- |
| Condición | Clase válida | Clase no válida | Nodo |
| Nº de parámetros | N = 2 | N < 2  N > 2 | 1 |
| 2.1 |
| 2.2 |
| Tipo de dato de n1 |  |  | 3 |
| 4 |
| Tipo de dato de n2 |  |  | 5 |
| 6 |
| Valor de n1 | n1 >= 0  n1 < 0 |  | 7 |
| 8 |
| Valor de n2 | n2 >= 0  n2 < 0 |  | 9 |
| 10 |

#### Pruebas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de prueba | n1 | n2 | Salida | Camino |
| Clases válidas | {3, 3} | | 9 | 1 – 3 – 5 – 7 – 9 |
| {-3, -4} | | -1 | 1 – 3 – 5 – 8 – 10 |
| Clases no válidas | {1} | | Error | 2.1 |
| {1, 3, 5} | | Error | 2.2 |
| {‘Hola’, 3} | | Error | ~~4~~ |
| {3, ‘Hola’} | | Error | 6 |

### JUnit

**import** **static** org.junit.Assert.\*;

**import** java.util.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

/\*\*

\* Clase que comprueba si el método ProductoEntero funciona correctamente

\* **@author** Paula

\* **@since** 14/04/2022

\*/

@RunWith (value = Parameterized.**class**)

**public** **class** OperacionesMatematicasTestProducto

{

**private** **int** n1;

**private** **int** n2;

**private** **int** resultado;

**public** OperacionesMatematicasTestProducto(**int** n1, **int** n2, **int** resultado)

{

**this**.n1 = n1;

**this**.n2 = n2;

**this**.resultado = resultado;

}

@Parameters

**public** **static** Iterable<Object[]> numeros()

{

**return** Arrays.*asList*(**new** Object[][] {{-1, 8, -1}, {3, -1, -1}, {0, 3, 0},

{3, 3, 9}, {-3, -4, -1}, {"Hola", 3, 0}, {3, "Hola", 0}});

}

@Test

**public** **void** testProductoEntero()

{

OperacionesMatematicas operacion = **new** OperacionesMatematicas (n1, n2);

**int** resultado = operacion.ProductoEntero();

*assertEquals*(resultado, **this**.resultado);

}

}

Gracias a estas pruebas detecté un error en el código, por lo que tuve que modificarlo y volver a ejecutarlas, siendo esta vez todas las pruebas exitosas.

## Método CocienteEntero

### Caja Blanca

#### Nodos

**public** **int** CocienteEntero()

{

1

**int** cociente;

2

3

**if**((n1 < 0) || (n2 <= 0))

4

cociente = -1;

**else**

{

5

cociente = 0;

6

**while**(n1 >= n2)

{

cociente++;

7

n1 = n1 - n2;

}

}

8

**return** cociente;

}

Diagrama

Descripción generada automáticamente

V1 = 4

V2 = 10 – 8 + 2 = 4

V3 = 3 + 1 = 4

#### Caminos:

1. 1 – 2 – 4 – 8
2. 1 – 3 – 4 – 8
3. 1 – 2 – 3 – 5 – 6 – 8
4. 1 – 2 – 3 – 5 – 6 – 7 – 6 – … – 8

#### Pruebas:

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | n1 | n2 | cociente |
| 1 | -1 | 8 | -1 |
| 2 | 3 | -1 | -1 |
| 3 | 0 | 3 | 0 |
| 4 | ~~9~~  ~~6~~  ~~3~~  0 | 3 | ~~0~~  ~~1~~  ~~2~~  3 |

### Caja Negra

#### Nodos

|  |  |  |  |
| --- | --- | --- | --- |
| Condición | Clase válida | Clase no válida | Nodo |
| Nº de parámetros | N = 2 | N < 2  N > 2 | 1 |
| 2.1 |
| 2.2 |
| Tipo de dato de n1 |  |  | 3 |
| 4 |
| Tipo de dato de n2 |  |  | 5 |
| 6 |
| Valor de n1 | n1 >= 0  n1 < 0 |  | 7 |
| 8 |
| Valor de n2 | n2 > 0  n2 <= 0 |  | 9 |
| 10 |

#### Pruebas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de prueba | n1 | n2 | Salida | Camino |
| Clases válidas | {9, 3} | | 3 | 1 – 3 – 5 – 7 – 9 |
| {-3, -4} | | -1 | 1 – 3 – 5 – 8 – 10 |
| Clases no válidas | {1} | | Error | 2.1 |
| {1, 3, 5} | | Error | 2.2 |
| {‘Hola’, 3} | | Error | ~~4~~ |
| {3, ‘Hola’} | | Error | 6 |

### JUnit

**import** **static** org.junit.Assert.\*;

**import** java.util.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

/\*\*

\* Clase que comprueba si el método CocienteEntero funciona correctamente

\* **@author** Paula

\* **@since** 14/04/2022

\*/

@RunWith (value = Parameterized.**class**)

**public** **class** OperacionesMatematicasTestCociente

{

**private** **int** n1;

**private** **int** n2;

**private** **int** resultado;

**public** OperacionesMatematicasTestCociente(**int** n1, **int** n2, **int** resultado)

{

**this**.n1 = n1;

**this**.n2 = n2;

**this**.resultado = resultado;

}

@Parameters

**public** **static** Iterable<Object[]> numeros()

{

**return** Arrays.*asList*(**new** Object[][] {{-1, 8, -1}, {3, -1, -1}, {0, 3, 0},

{9, 3, 3}, {-3, -4, -1}, {"Hola", 3, 0}, {3, "Hola", 0}});

}

@Test

**public** **void** testCocienteEntero()

{

OperacionesMatematicas operacion = **new** OperacionesMatematicas (n1, n2);

**int** resultado = operacion.CocienteEntero();

*assertEquals*(resultado, **this**.resultado);

}

}

Al realizar las pruebas pude comprobar que este método no presentaba errores.

## Método RestoEntero

### Caja Blanca

#### Nodos

**public** **int** RestoEntero()

{

1

**int** resto;

2

3

**if**((n1 < 0) || (n2 <= 0))

4

cociente = -1;

**else**

{

5

resto = n2;

6

**while**(resto >= n2)

{

7

resto = resto - n2;

}

}

8

**return** cociente;

}

Diagrama

Descripción generada automáticamente

V1 = 4

V2 = 10 – 8 + 2 = 4

V3 = 3 + 1 = 4

#### Caminos:

1. 1 – 2 – 4 – 8
2. 1 – 3 – 4 – 8
3. 1 – 2 – 3 – 5 – 6 – 8
4. 1 – 2 – 3 – 5 – 6 – 7 – 6 – … – 8

#### Pruebas:

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | n1 | n2 | resto |
| 1 | -1 | 8 | -1 |
| 2 | 3 | -1 | -1 |
| 3 | 0 | 3 | 0 |
| 4 | 9 | 3 | ~~9~~  ~~6~~  ~~3~~  0 |

### Caja Negra

#### Nodos

|  |  |  |  |
| --- | --- | --- | --- |
| Condición | Clase válida | Clase no válida | Nodo |
| Nº de parámetros | N = 2 | N < 2  N > 2 | 1 |
| 2.1 |
| 2.2 |
| Tipo de dato de n1 |  |  | 3 |
| 4 |
| Tipo de dato de n2 |  |  | 5 |
| 6 |
| Valor de n1 | n1 >= 0  n1 < 0 |  | 7 |
| 8 |
| Valor de n2 | n2 > 0  n2 <= 0 |  | 9 |
| 10 |

#### Pruebas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de prueba | n1 | n2 | Salida | Camino |
| Clases válidas | {9, 3} | | 0 | 1 – 3 – 5 – 7 – 9 |
| {-3, -4} | | -1 | 1 – 3 – 5 – 8 – 10 |
| Clases no válidas | {1} | | Error | 2.1 |
| {1, 3, 5} | | Error | 2.2 |
| {‘Hola’, 3} | | Error | ~~4~~ |
| {3, ‘Hola’} | | Error | 6 |

### JUnit

**import** **static** org.junit.Assert.\*;

**import** java.util.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

/\*\*

\* Clase que comprueba si el método RestoEntero funciona correctamente

\* **@author** Paula

\* **@since** 14/04/2022

\*/

@RunWith (value = Parameterized.**class**)

**public** **class** OperacionesMatematicasTestResto

{

**private** **int** n1;

**private** **int** n2;

**private** **int** resultado;

**public** OperacionesMatematicasTestResto(**int** n1, **int** n2, **int** resultado)

{

**this**.n1 = n1;

**this**.n2 = n2;

**this**.resultado = resultado;

}

@Parameters

**public** **static** Iterable<Object[]> numeros()

{

**return** Arrays.*asList*(**new** Object[][] {{-1, 8, -1}, {3, -1, -1}, {0, 3, 0},

{9, 3, 0}, {-3, -4, -1}, {"Hola", 3, 0}, {3, "Hola", 0}});

}

@Test

**public** **void** testRestoEntero()

{

OperacionesMatematicas operacion = **new** OperacionesMatematicas (n1, n2);

**int** resultado = operacion.RestoEntero();

*assertEquals*(resultado, **this**.resultado);

}

}

Tras ejecutar el código no encontré errores en el código a parte de los que ya es habían especificado en el análisis de Caja Negra realizado previamente.

## Método PotenciaEntera

### Caja Blanca

#### Nodos

**public** **int** PotenciaEntera()

{

1

**int** potencia;

2

3

**if**((n1 < 0) || (n2 < 0))

4

potencia = -1;

**else**

{

5

potencia = 1;

6

**while**(n2 > 0)

{

7

potencia = potencia \* n1;

n2 = n2 -1;

}

}

8

**return** potencia;

}

Diagrama

Descripción generada automáticamente

V1 = 4

V2 = 10 – 8 + 2 = 4

V3 = 3 + 1 = 4

#### Caminos:

1. 1 – 2 – 4 – 8
2. 1 – 3 – 4 – 8
3. 1 – 2 – 3 – 5 – 6 – 8
4. 1 – 2 – 3 – 5 – 6 – 7 – 6 – … – 8

#### Pruebas:

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | n1 | n2 | potencia |
| 1 | -1 | 8 | -1 |
| 2 | 3 | -1 | -1 |
| 3 | 0 | 3 | 0 |
| 4 | 6 | ~~2~~  ~~1~~  0 | ~~1~~  ~~6~~  36 |

### Caja Negra

#### Nodos

|  |  |  |  |
| --- | --- | --- | --- |
| Condición | Clase válida | Clase no válida | Nodo |
| Nº de parámetros | N = 2 | N < 2  N > 2 | 1 |
| 2.1 |
| 2.2 |
| Tipo de dato de n1 |  |  | 3 |
| 4 |
| Tipo de dato de n2 |  |  | 5 |
| 6 |
| Valor de n1 | n1 >= 0  n1 < 0 |  | 7 |
| 8 |
| Valor de n2 | n2 >= 0  n2 < 0 |  | 9 |
| 10 |

#### Pruebas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de prueba | n1 | n2 | Salida | Camino |
| Clases válidas | {6, 2} | | 36 | 1 – 3 – 5 – 7 – 9 |
| {-3, -4} | | -1 | 1 – 3 – 5 – 8 – 10 |
| Clases no válidas | {1} | | Error | 2.1 |
| {1, 3, 5} | | Error | 2.2 |
| {‘Hola’, 3} | | Error | ~~4~~ |
| {3, ‘Hola’} | | Error | 6 |

### JUnit

/\*\*

\* Clase que comprueba si el método PotenciaEntera funciona correctamente

\* **@author** Paula

\* **@since** 14/04/2022

\*/

@RunWith (value = Parameterized.**class**)

**public** **class** OperacionesMatematicasTestPotencia

{

**private** **int** n1;

**private** **int** n2;

**private** **int** resultado;

**public** OperacionesMatematicasTestPotencia(**int** n1, **int** n2, **int** resultado)

{

**this**.n1 = n1;

**this**.n2 = n2;

**this**.resultado = resultado;

}

@Parameters

**public** **static** Iterable<Object[]> numeros()

{

**return** Arrays.*asList*(**new** Object[][] {{-1, 8, -1}, {3, -1, -1}, {0, 3, 0},

{6, 2, 36}, {-3, -4, -1}, {"Hola", 3, 0}, {3, "Hola", 0}});

}

@Test

**public** **void** testPotenciaEntera()

{

OperacionesMatematicas operacion = **new** OperacionesMatematicas (n1, n2);

**int** resultado = operacion.PotenciaEntera();

*assertEquals*(resultado, **this**.resultado);

}

}

Tras ejecutar el código no se encontraron errores en el código.

## Método Factorial

### Caja Blanca

#### Nodos

**public** **int** Factorial()

{

1

**int** factorial;

2

**if**((n1 < 0)))

3

factorial = -1;

**else**

{

4

factorial = 1;

5

**while**(n1 > 1)

{

6

factorial = factorial \* n1;

n1--;

}

}

7

**return** factorial;

}

Diagrama

Descripción generada automáticamente

V1 = 3

V2 = 8 – 7 + 2 = 3

V3 = 2 + 1 = 3

#### Caminos:

1. 1 – 2 – 3 – 7
2. 1 – 2 – 4 – 5 – 7
3. 1 – 2 – 4 – 5 – 6 – 5 – … – 7

#### Pruebas:

|  |  |  |
| --- | --- | --- |
| Camino | n1 | factorial |
| 1 | -1 | -1 |
| 2 | 1 | 1 |
| 3 | ~~5~~  ~~4~~  ~~3~~  ~~2~~  1 | ~~1~~  ~~5~~  ~~20~~  ~~60~~  120 |

### Caja Negra

#### Nodos

|  |  |  |  |
| --- | --- | --- | --- |
| Condición | Clase válida | Clase no válida | Nodo |
| Nº de parámetros | N = 1 | N < 1  N > 1 | 1 |
| 2.1 |
| 2.2 |
| Tipo de dato de n1 |  |  | 3 |
| 4 |
| Valor de n1 | n1 >= 0  n1 < 0 |  | 5 |
| 6 |

#### Pruebas

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de prueba | n1 | Salida | Camino |
| Clases válidas | {5} | 120 | 1 – 3 – 5 |
| {-1} | -1 | 1 – 3 – 5 – 6 |
| Clases no válidas | {} | Error | 2.1 |
| {1, 3} | Error | 2.2 |
| {‘Hola’} | Error | 4 |

### JUnit

**import** **static** org.junit.Assert.\*;

**import** java.util.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

/\*\*

\* Clase que comprueba si el método Factorial funciona correctamente

\* **@author** Paula

\* **@since** 16/04/2022

\*/

@RunWith (value = Parameterized.**class**)

**public** **class** OperacionesMatematicasTestFactorial

{

**private** **int** n1;

**private** **int** resultado;

**public** OperacionesMatematicasTestFactorial(**int** n1, **int** resultado)

{

**this**.n1 = n1;

**this**.resultado = resultado;

}

@Parameters

**public** **static** Iterable<Object[]> numeros()

{

**return** Arrays.*asList*(**new** Object[][] {{-1, -1}, {1, 1}, {5, 120}, {"Hola", 0}});

}

@Test

**public** **void** testFactorial()

{

OperacionesMatematicas operacion = **new** OperacionesMatematicas (n1);

**int** resultado = operacion.Factorial();

*assertEquals*(resultado, **this**.resultado);

}

}

Tras realizar las pruebas no se encontraron errores en el código, sin tener en cuenta los previstos en caja negra.

## Método ProductoRuso

### Caja Blanca

#### Nodos

**public** **int** ProductoRuso()

{

1

**int** solucion;

3

2

**if**((n1 < 0) || (n2 < 2))

4

solucion = -1;

**else**

{

5

solucion = 0;

6

**if**(n1 > 0)

{

7

**while**(n2 > 0)

{

8

**if**(n2 % 2 != 0)

9

solucion = solucion + n1;

n1 = n1 \* 2;

10

n2 = n2 / 2;

}

}

}

11

**return** solucion;

}

Diagrama

Descripción generada automáticamente

V1 = 6

V2 = 15 – 11 + 2 = 6

V3 = 5 + 1 = 6

#### Caminos:

1. 1 – 2 – 4 –11
2. 1 – 2 – 3 – 4 – 11
3. 1 – 2 – 3 – 5 – 6 – 11
4. 1 – 2 – 3 – 5 – 6 – 7 –… – 11
5. 1 – 2 – 3 – 5 – 6 – 7 – 8 – 10 – 7 – … – 11
6. 1 – 2 – 3 – 5 – 6 – 7 – 8 – 9 – 10 – 7 – … – 11

***Pruebas:***

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | n1 | n2 | solucion |
| 1 | -1 | 8 | -1 |
| 2 | 3 | -1 | -1 |
| 3 | 0 | 3 | 0 |
| 4 | 3 | 0 | 0 |
| 5 | ~~2~~  ~~4~~  8 | ~~2~~  ~~1~~  0 | ~~0~~  4 |
| 6 | ~~2~~  ~~4~~  8 | ~~3~~  ~~1~~  0 | ~~0~~  ~~2~~  6 |

### Caja Negra

#### Nodos

|  |  |  |  |
| --- | --- | --- | --- |
| Condición | Clase válida | Clase no válida | Nodo |
| Nº de parámetros | N = 2 | N < 2  N > 2 | 1 |
| 2.1 |
| 2.2 |
| Tipo de dato de n1 |  |  | 3 |
| 4 |
| Tipo de dato de n2 |  |  | 5 |
| 6 |
| Valor de n1 | n1 >= 0  n1 < 0 |  | 7 |
| 8 |
| Valor de n2 | n2 >= 0  n2 < 0 |  | 9 |
| 10 |

#### Pruebas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de prueba | n1 | n2 | Salida | Camino |
| Clases válidas | {2, 2} | | 4 | 1 – 3 – 5 – 7 – 9 |
| {-3, -4} | | -1 | 1 – 3 – 5 – 8 – 10 |
| Clases no válidas | {1} | | Error | 2.1 |
| {1, 3, 5} | | Error | 2.2 |
| {‘Hola’, 3} | | Error | ~~4~~ |
| {3, ‘Hola’} | | Error | 6 |

### JUnit

**import** **static** org.junit.Assert.\*;

**import** java.util.\*;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

/\*\*

\* Clase que comprueba si el método ProductoRuso funciona correctamente

\* **@author** Paula

\* **@since** 16/04/2022

\*/

@RunWith (value = Parameterized.**class**)

**public** **class** OperacionesMatematicasTestProductoRuso

{

**private** **int** n1;

**private** **int** n2;

**private** **int** resultado;

**public** OperacionesMatematicasTestProductoRuso(**int** n1, **int** n2, **int** resultado)

{

**this**.n1 = n1;

**this**.n2 = n2;

**this**.resultado = resultado;

}

@Parameters

**public** **static** Iterable<Object[]> numeros()

{

**return** Arrays.*asList*(**new** Object[][] {{-1, 8, -1}, {3, -1, -1}, {0, 3, 0}, {3, 0, 0},

{2, 2, 4}, {2, 3, 6}, {-3, -4, -1}, {"Hola", 3, 0}, {3, "Hola", 0}});

}

@Test

**public** **void** testProductoRuso()

{

OperacionesMatematicas operacion = **new** OperacionesMatematicas (n1, n2);

**int** resultado = operacion.ProductoRuso();

*assertEquals*(resultado, **this**.resultado);

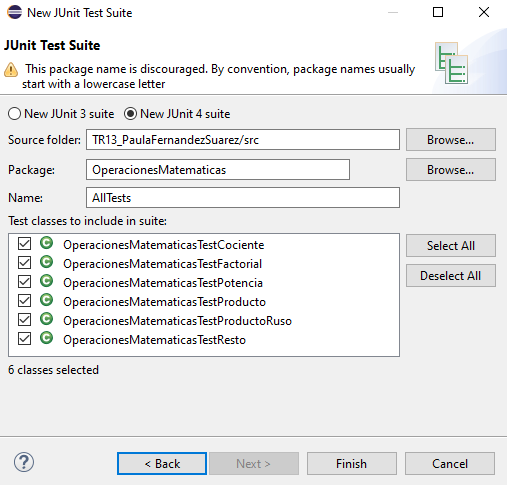
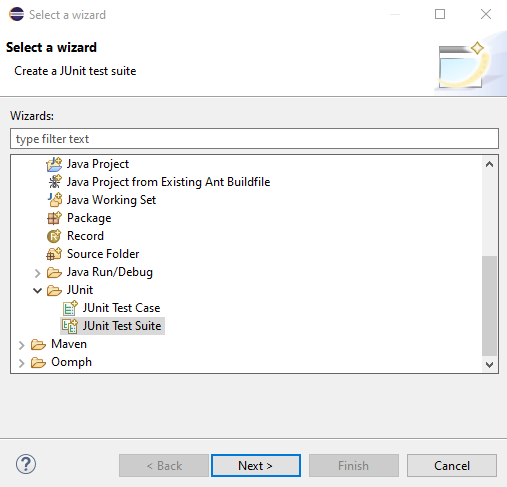
}

}

Al realizar las pruebas detecté que no comprobaba correctamente qué números eran válidos, ya que en vez de 0 había escrito 2 en una de las condiciones. Corregí el problema y añadí a git los archivos modificados. También me di cuenta de que había olvidado realizar la comprobación de {3, 0, 0} en el test de ProductoEntero, por lo que la añadí al conjunto de pruebas.

## Suite de pruebas

Una vez creadas todas la pruebas de cada método hice una suite de pruebas con todas las pruebas unitarias en ella, de tal forma que se ejecutasen todas a la vez. Para ello vamos al menú y hacemos click en new. Se nos abrirá una ventana con opciones y seleccionaremos la opción de “JUnit Test Suite”. A continuación, seleccionaremos los casos de prueba que queremos que contenga y le daremos a “Finish”.



Por último, la añadiremos a git, de tal forma que ya habríamos terminado con las modificaciones del código.

Esto significa que antes de realizar el último commit debemos actualizar el javadoc que habíamos creado con anterioridad y añadirlo también a git.

Una vez hecho esto podeos mirar todas las modificaciones que hemos realizado con ‘git log –oneline’.

$ git log --oneline

57efe63 (HEAD -> Develop) Actualizado javadoc

57847ad Realizada la suite de pruebas

f1c3e12 Realizadas las pruebas de ProductoRuso y modificado el código para que funcione correctamente, tanto las pruebas de ProductoEntero como el método ProductoRuso

13cafb1 Realizadas las pruebas de Factorial

1cbd660 Realizadas las pruebas de PotenciaEntera

d08d090 Realizadas las pruebas de RestoEntero

641d819 Realizadas las pruebas de CocienteEntero

20905bd Realizadas las pruebas de ProductoEntero y corregido el código

baea790 Refactorizamos el código y actualizamos javadoc

dc6d828 Añadimos comentarios y generamos javadoc

246cd4d (origin/main, main) Añadimos el código inicial, sin comentarios ni pruebas unitarias

# Añadir los archivos al repositorio remoto

Para añadir los archivos generados debemos añadir todos los cambios a la rama maste. Para ello se utiliza la instrucción ‘git merge’ que une una rama con la rama hija especificada.

Antes de realizar la instrucción debemos colocarnos en la rama a la que le queremos añadir los cambios. Como me mostraba un error al cambiar las ramas tuve que forzar el cambio utilizando -f antes del nombre de la rama.

$ git checkout -f main

$ git merge 'Develop'

Por último, para añadir los cambios al repositorio remoto debemos utilizar la instrucción push.

$ git push

Si a continuación entramos en nuestro repositorio remoto veremos algo de este estilo.

Texto, Aplicación

Descripción generada automáticamente

Es decir, indica en qué modificación es la última que se ha realizado en cada parte.

Al día siguiente, al revisar las modificaciones que había hecho, me di cuenta de que había añadido la suite de pruebas y el test de ProductoRuso a la carpeta equivocada, por lo que añadí de nuevo toda la carpeta con los cambios correspondientes y el documento en Word que había estado realizando, de tal forma que no solo tendría las versiones del proyecto de eclipse, sino las de las modificaciones que realizase en el documento y en la presentación que estaba realizando.

Al finalizar quedó así: