

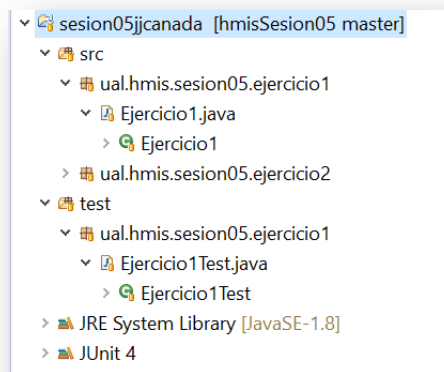
Pruebas unitarias con JUnit

Herramientas y métodos de Ingeniería del Software. Universidad de Almería

Joaquín Cañadas. Curso 2018

Crear de forma individual, un proyecto en Eclipse donde resolver los siguientes ejercicios, denominado **Sesion05abc123** (sustituyendo abc123 por su usuario de la UAL)

Cada ejercicio se debe resolver en un paquete denominado **ual.hmis.sesion05.ejercicioX**, excepto el ejercicio 6, que tendrá su propio proyecto.



REQUISITO: plugin ECLEMMMA para Eclipse (cálculo de la cobertura). Cualquier versión de Eclipse es válida. (ver instrucciones de instalación y uso más adelante en este guión)

Guardar dicho proyecto en un repositorio privado en GitHub en la cuenta personal (La resolución es **individual**). Dar acceso de lectura al profesor (usuario de GitHub: [ualjjcanada](#)).

Contenido

Ejercicios Pruebas unitarias con JUnit.....	2
Información adicional.....	7
Plugin EclEmma para el cálculo de la cobertura de código.....	7
Creación del repositorio Git local y configuración de repo remoto desde Eclipse	9

Ejercicios Pruebas unitarias con JUnit

- 1) Identifique las clases de equivalencia para el parámetro x en el siguiente método Java, e implemente los **casos de prueba** en JUnit necesarios para obtener un **100% de cobertura**:

```
public class Ejercicio1{

    public int transformar (int x) {
        int resultado = 0;
        if (x % 2 == 0) // % Resto de una división entre enteros (mod)
            resultado = transformar (x/2);
        else if (x % 3 == 0)
            resultado = transformar (x/3);
        else if (x % 5 == 0)
            resultado = transformar (x/5);
        else return x;

        return resultado;
    }
}
```

- 2) Escriba el código de un sencillo método en Java, denominado **login**, que reciba como parámetros de entrada dos valores tipo cadena de caracteres, **username** y **password**. El método debe comprobar que:

- los dos valores sean distintos de cadena vacía
- que la longitud de ambos sea menor de 30 caracteres.

En tal caso, debe hacer la llamada a otro método o función denominado "**compruebaLoginEnBD(username, password)**" (que suponemos que existe y no hay que implementar) que devuelve TRUE si **username** y **password** son valores correctos existentes en la BD.

```
public class Ejercicio2 {  
    public boolean login (String username, String password){  
        // comprobar que sean distintos de vacio  
  
        // comprobar que la longitud sea < 30  
  
        // llamar al método de la bbdd  
        return compruebaLoginEnBD(username, password);  
    }  
  
    public boolean compruebaLoginEnBD  
        (String username, String password){  
        // método mock (simulado)  
        if (username == "user" && password == "pass")  
            return true;  
        else  
            return false;  
    }  
}
```

Implemente además los casos de prueba necesarios para obtener un **100% de cobertura** del código del método utilizando JUnit, usando los valores límite adecuados donde proceda.

- 3) Escriba el código en Java de un método que tome un valor numérico entero como parámetro de entrada y devuelva un valor tipo cadena de caracteres cuyo contenido debe ser tantos caracteres '*' (asteriscos) como indica el parámetro de entrada. Por ejemplo, si el valor de entrada es 5 la cadena de salida sería '*****'. Si el valor de entrada es negativo la cadena de salida debe contener el texto 'número erróneo'.

Implemente los **casos de prueba** en JUnit necesarios para obtener un **100% de cobertura**.

- 4) Escriba el código de un método en Java que tome como parámetros de entrada dos cadenas de caracteres, P1, P2, y devuelva otra cadena que contenga los caracteres de P1 que no están en P2.

Implemente los **casos de prueba** en JUnit necesarios para obtener un **100% de cobertura**.

- 5) Escriba el código de un sencillo método en Java que reciba cómo parámetro de entrada una cadena de caracteres:

String subcadenaHastaPunto (String cadena)

Y como salida devuelva la subcadena desde el primer carácter hasta el primer punto '.' (inclusive). Tenga en cuenta que:

a) si la "cadena" de entrada no tiene puntos, debe devolver "Error: cadena sin punto"; y

b) si "cadena" tiene un dígito (0..9) antes del primer punto, debe devolver el texto "Error: cadena con dígito".

Para la implementación, utilice los métodos Java:

`int strlen(String s):` devuelve la longitud de la cadena
`boolean isDigit(char c) :` devuelve true si el carácter es un dígito (0..9)

Ejemplos:

Valor de entrada	Resultado de salida
"Mayor de edad"	"Error: cadena sin punto"
"Menor. De edad 3"	"Menor."
"Edad 3 años"	"Error: cadena con algún número"

Implemente los **casos de prueba** en JUnit necesarios para obtener un **100% de cobertura**, usando los **valores límite** adecuados donde proceda

- 6) A partir del código Java disponible en el repositorio GitHub **notasActividades** (<https://github.com/ualhmis/notasActividades>), que consiste en un ejercicio que ya visteis en la asignatura Ingeniería del Software de 2º, realice un fork en un repositorio en su cuenta GitHub personal. Tras ello, convierta el repositorio en privado: sin embargo, un repositorio GitHub forkeado desde un repo público no se puede convertir a privado, por lo tanto hay que duplicarlo. Para ello:

- cree un nuevo repositorio privado en GitHub
- con Git Bash, cree un clon "bare" del repositorio forkeado

```
git clone --bare https://github.com/exampleuser/forked-repository.git
```

- Mirror-push al nuevo repositorio.

```
cd forked-repository.git  
git push --mirror git@github.com:exampleuser/new-repository.git
```

- Elimine el repositorio local creado en el paso anterior

```
cd ..  
rm -rf forked-repository.git
```

Ya puede trabajar normalmente con el nuevo repositorio privado.

En él, implemente el método de la clase **Alumno** denominado **calcularNotaActividad** que recibe por parámetro el **nombre** de una actividad y que devuelva un **double** con la suma de la puntuación de todos los ejercicios de dicha actividad. En la ejecución de dicho método, se debe actualizar el valor de la propiedad (**puntuacionTotal**) de la clase **Actividad**.

Implemente los **casos de prueba** en JUnit necesarios para obtener un **100% de cobertura** para el nuevo método **calcularNotaActividad**, usando los **valores límite** adecuados donde proceda.

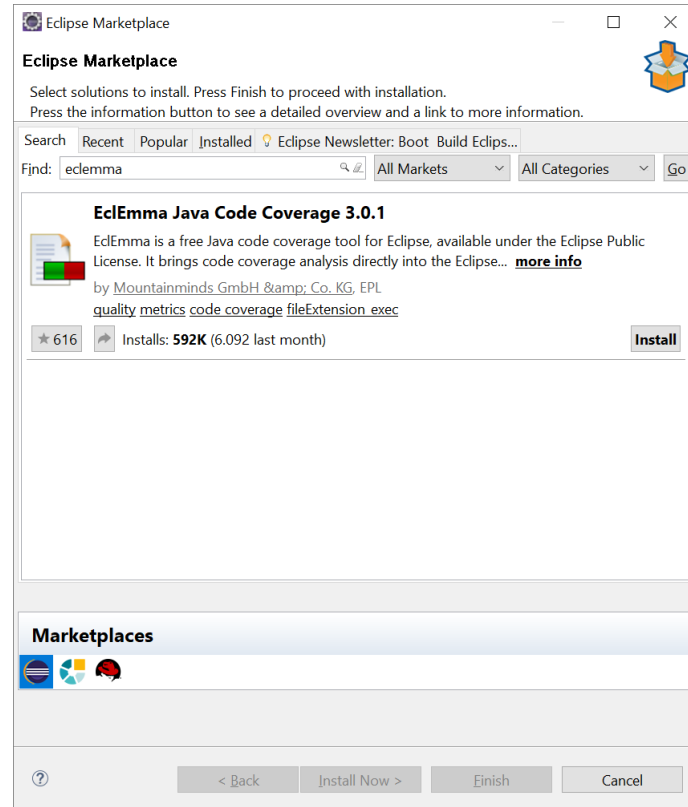
- 7) Para cada uno de los ejercicios del 1 al 6, escriba una nueva clase de test que implemente los mismos casos de prueba, usando en este caso **tests parametrizados**.

Como ejemplo, consultar el repositorio JUnitComplejo con test parametrizados disponible en GitHub: <https://github.com/ualhmis/junitComplejo>

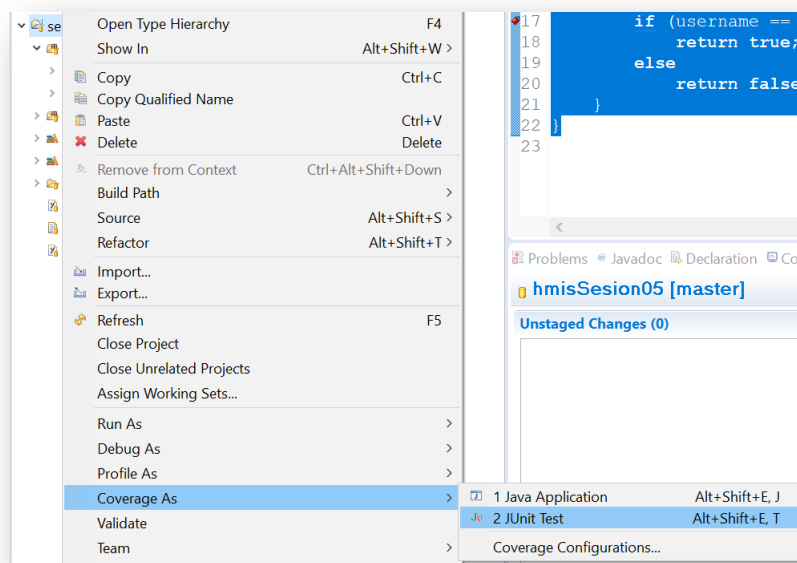
Información adicional

Plugin EclEmma para el cálculo de la cobertura de código

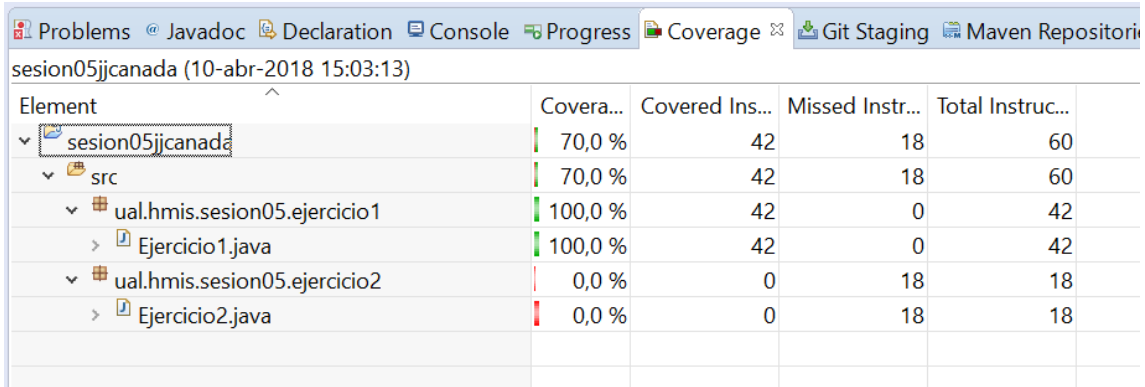
Desde el MarketPlace, instalar plugin EclEmma en cualquier Eclipse que tengas instalado.



Una vez instalado el plugin, calcular cobertura: Coverage as / JUnit test

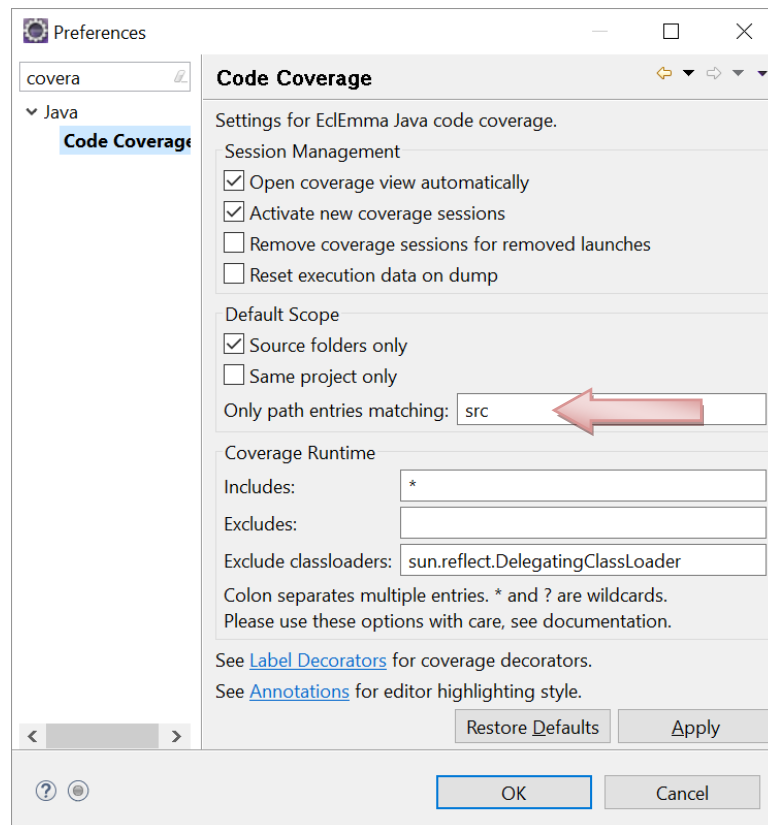


Y se muestra la vista de la métrica de cobertura



Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
sesion05jjcanada	70,0 %	42	18	60
src	70,0 %	42	18	60
ual.hmis.sesion05.ejercicio1	100,0 %	42	0	42
Ejercicio1.java	100,0 %	42	0	42
ual.hmis.sesion05.ejercicio2	0,0 %	0	18	18
Ejercicio2.java	0,0 %	0	18	18

Para calcular la cobertura de únicamente la carpeta "src", y no de la carpeta "test": Window / Preferences / Java / Code Coverage . Y en esta ventana, "Only path entries matching" : src

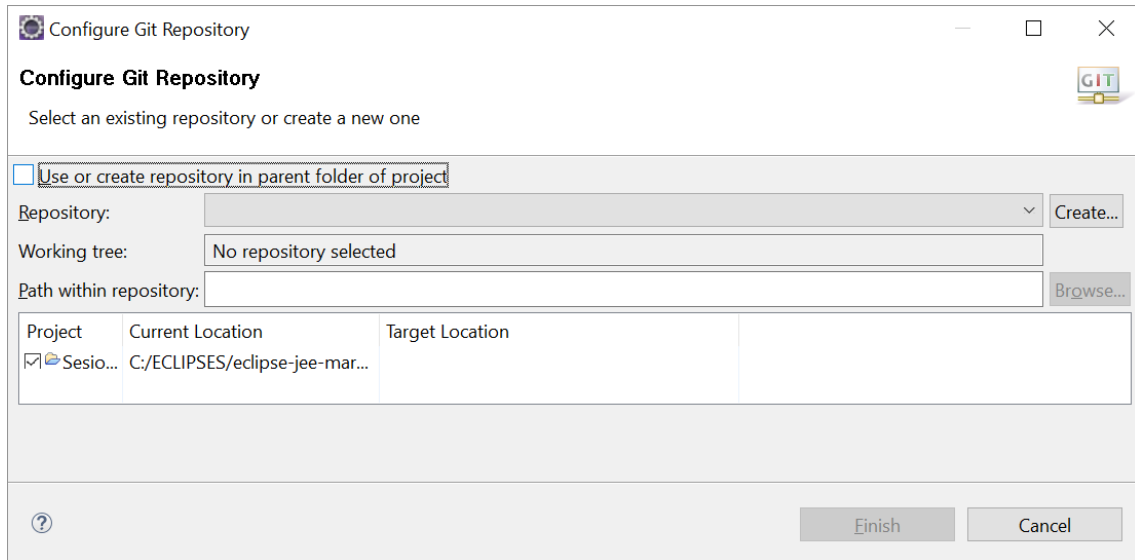


Creación del repositorio Git local y configuración de repo remoto desde Eclipse

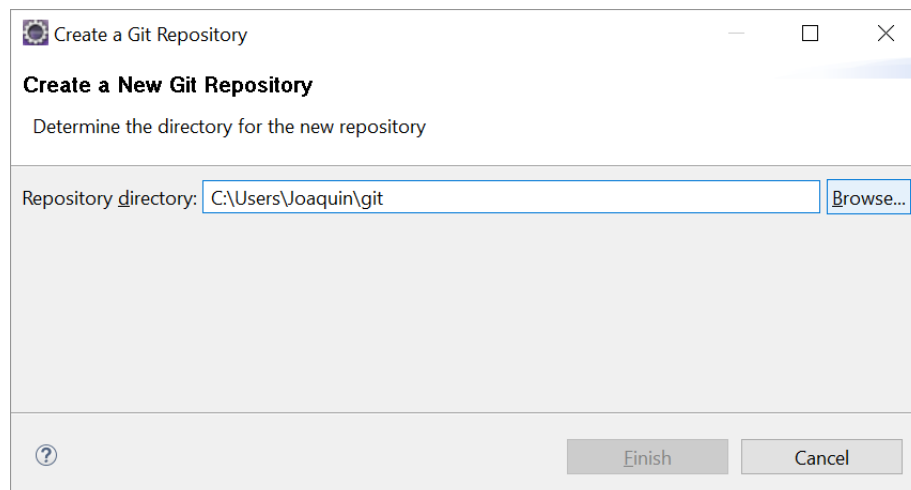
1) Primero creamos el repositorio Git local y hacemos commit de los cambios:

Sobre el proyecto, botón derecho, Team / Share project / Git

Quitamos la marca "Use or create repository in parent folder of project"



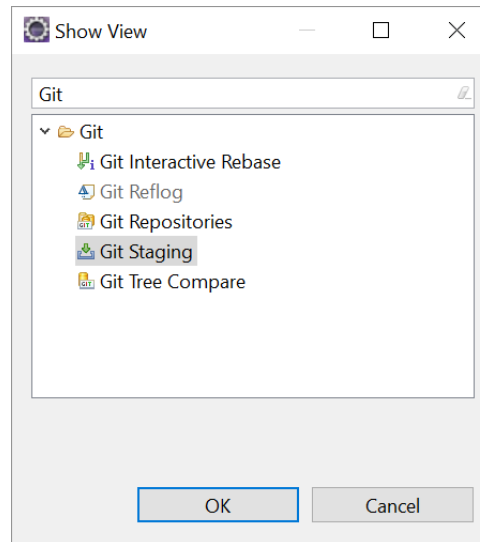
Create...



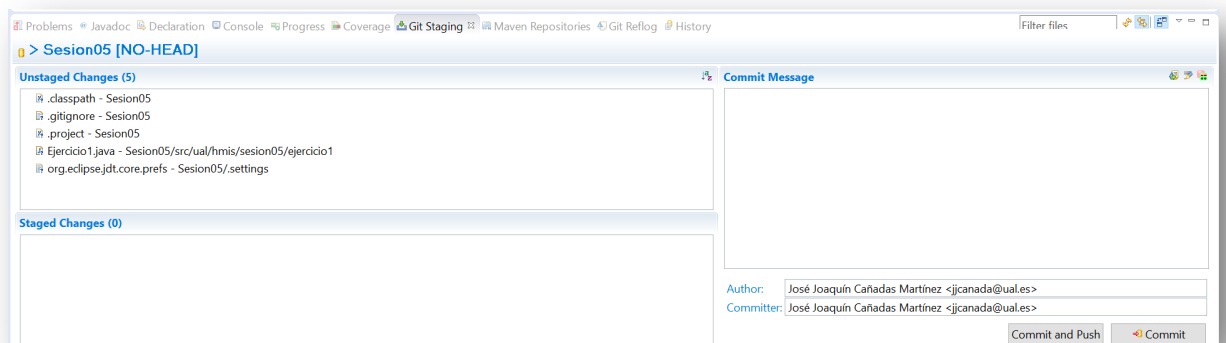
Browse...

Creamos la carpeta local donde queremos que se guarde nuestro repositorio. Finish / Finish

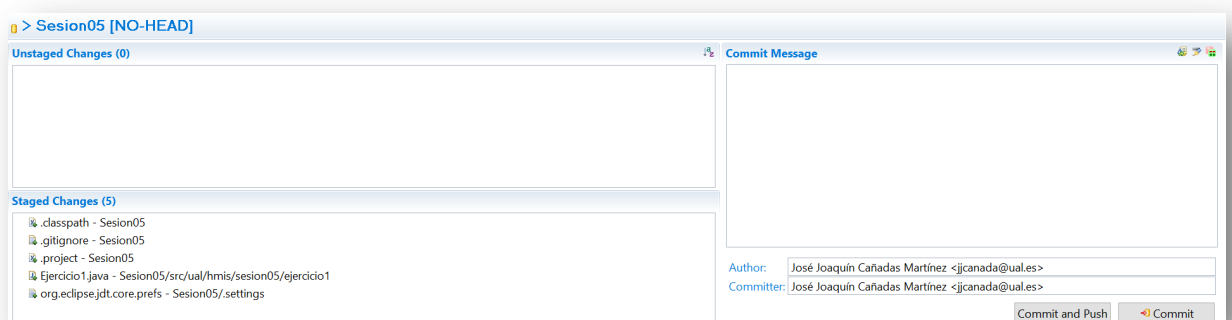
Habilitamos la vista de Git Staging: Window / Show view



Aparecen los archivos en la vista, aun en estado de no preparado:

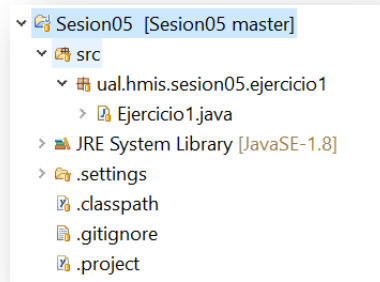


Los pasamos a "preparados" seleccionando los archivos , botón derecho, add to index



Escribimos un mensaje y click en Commit.

Ahora los archivos están en la rama master del repositorio git.

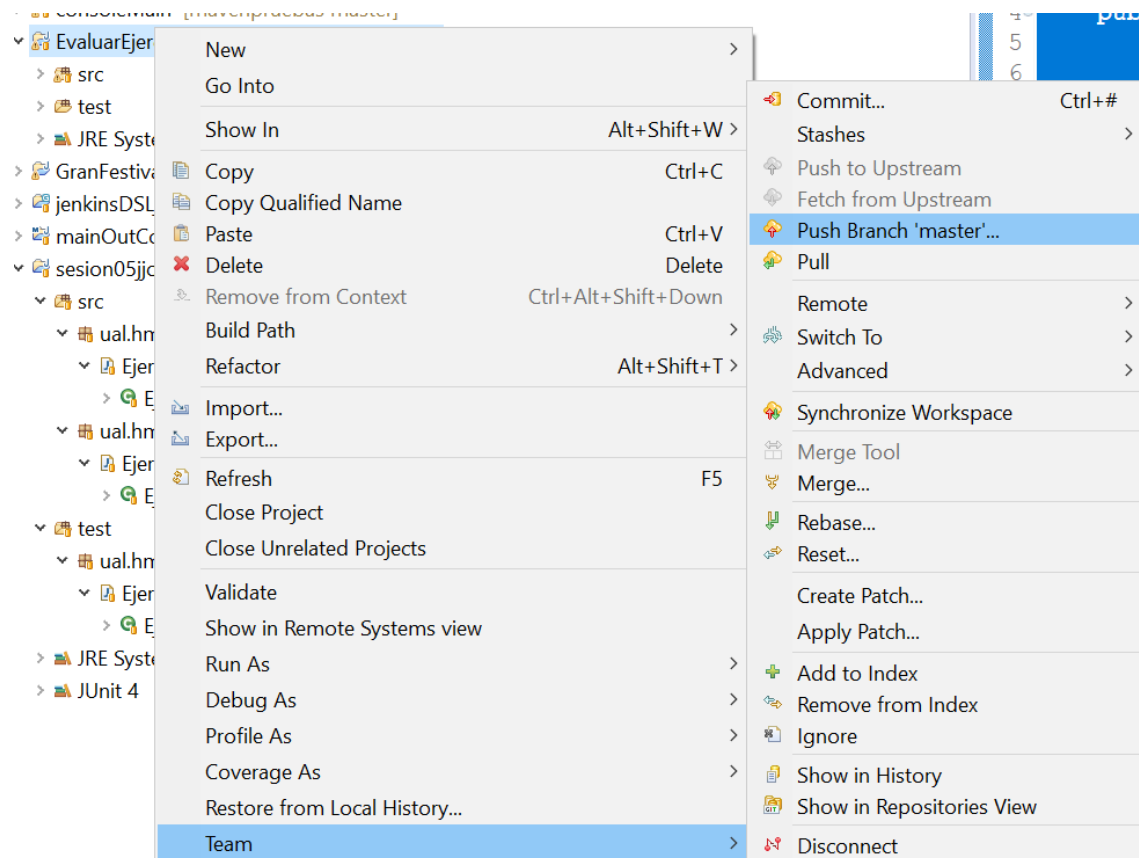


2) Creamos el repositorio privado en GitHub

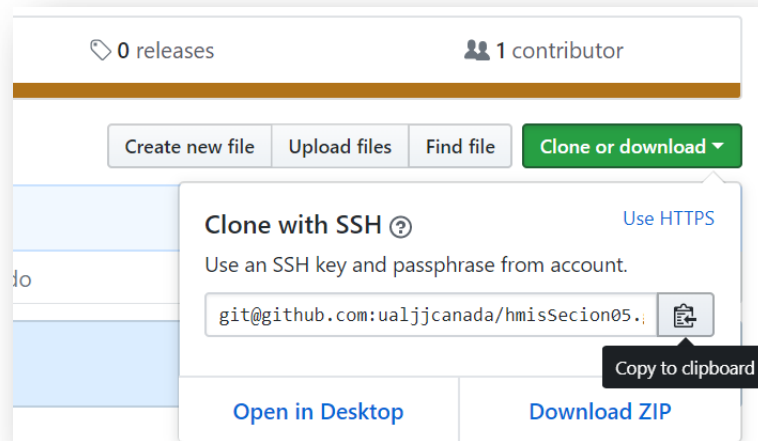
En nuestra cuenta de GitHub, creamos un nuevo repositorio y lo marcamos como privado.

3) Configuramos el repositorio remoto de GitHub y hacemos push

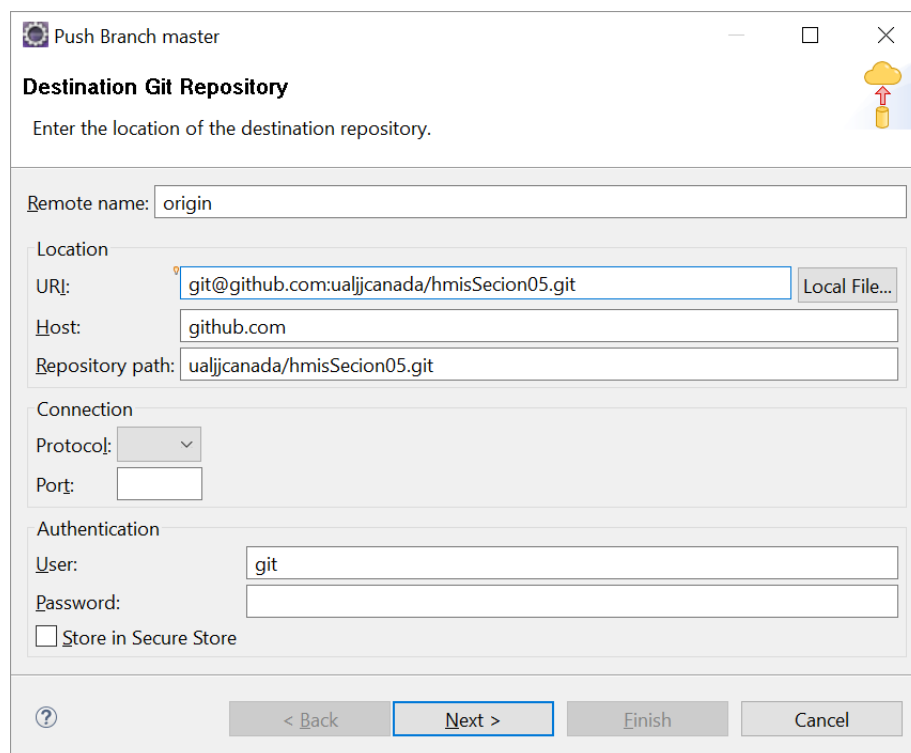
Sobre el proyecto, Team / Push Branch 'master'...



A continuación, copiamos la URL SSH del repositorio privado GitHub



Y pegamos la URL en la ventana de Eclipse de configuración del remoto:



Next / Next / Finish

Y comprobamos que el proyecto Eclipse se ha subido al repositorio GitHub.

A partir de ahora, ya podremos realizar "Commit & Push"