Pruebas unitarias con JUnit

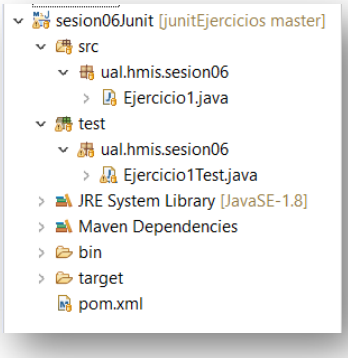
Herramientas y métodos de Ingeniería del Software. Universidad de Almería

Joaquín Cañadas

Crear de forma **individual**, un proyecto en Eclipse donde resolver los siguientes ejercicios, denominado **Sesion06abc123** (sustituyendo abc123 por su usuario de la UAL)

Cada ejercicio se debe resolver en un paquete denominado **`ual.hmis.sesion06.ejercicioX`**, excepto el ejercicio 5, que tendrá su propio proyecto.

Utilice en todos ellos JUnit 5 y tests parametrizados. Como ejemplo, consultar el repositorio JUnitEjercicios con test parametrizados disponible en GitHub: https://github.com/ualhmis/junitEjercicios



REQUISITO: plugin ECLEMMA para Eclipse (cálculo de la cobertura). Cualquier versión de Eclipse es válida, las últimas versiones ya lo traen instalado de forma predeterminada. En caso de usar una versión de Eclipse que no lo tenga, ver instrucciones de instalación y uso más adelante en este guión.

Guardar dicho proyecto en un repositorio en GitHub en la cuenta personal (La resolución es **individual**). En caso de ser un repositorio privado, dar acceso de lectura al profesor (usuario de GitHub: ualjjcanada).

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 cómo 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.
- ***El password debe tener al menos una letra mayúscula, una minúscula y un dígito***

En tal caso, debe hacer la llamada a otro método denominado "**`compruebaLoginEnBD(username, password)`**" (que para este ejercicio estamos simulando su funcionamiento) 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 cómo parámetro de entrada y devuelva una 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'. S***i el valor de entrada es menor que 5, mostrará 5 asteriscos, y si es mayor que 12, mostrará 12 asteriscos.***

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 cómo parámetros de entrada dos cadenas de caracteres, P1, P2, y devuelva otra cadena que contenga los caracteres de P1 que ***están*** en P2.

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

5) 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.

6) Sobre el mismo proyecto del ejercicio 5, se desea añadir el método "`calificación`" de la clase "`Alumno`" que recibe un número real entre 0 y 10 y devuelve una cadena con la calificación, de acuerdo con la siguiente lógica:

Valor de nota	**Calificación**
Entre 0 y 4,9	Suspenso
Entre 5 y 6,9	Aprobado
Entre 7 y 8,9	Notable
Entre 9 y 9,9	Sobresaliente
10	Matrícula
Menor que 0 ó mayor a 10	Error en la nota

```
public class Alumno{
      ...
      public String calificacion (double nota){
            ...
      }
}
```

Utilizando JUnit 5 implemente los casos de prueba necesarios con **tests parametrizados** para obtener un **100% de cobertura** del método anterior, usando los **valores límite** adecuados donde proceda.

7) Sobre el mismo proyecto del ejercicio 5, escriba el método "**`cursoSegunEdad`**" de la clase "`Alumno`" que reciba cómo parámetro de entrada el año de nacimiento y devuelva una cadena de caracteres que contenga el curso y ciclo en el que se debe matricular un niño para el próximo curso 2020-2021, según el sistema educativo español detallado en la siguiente tabla:

`String cursoSegunEdad (int anyoNacimiento)`

Año Nacim.	Curso
2017	1º Educación Infantil
2016	2º Educación Infantil
2015	3º Educación Infantil
2014	1º Educación Primaria
2013	2º Educación Primaria
2012	3º Educación Primaria
2011	4º Educación Primaria
2010	5º Educación Primaria
2009	6º Educación Primaria
2008	1º Educación Secundaria
2007	2º Educación Secundaria
2006	3º Educación Secundaria
2005	4º Educación Secundaria

Utilizando JUnit 5 implemente los casos de prueba necesarios con **tests parametrizados** para obtener un **100% de cobertura** del método anterior.

8) Se quiere implementar el control de un Ferry.

Deberá implementar la clase **Vehículo**, que gestione aspectos comunes a todos los vehículos como: número de pasajeros, número de ruedas, peso del vehículo con carga (Kg) (sin pasajeros). El Ferry transporta un conjunto de vehículos, y sus pasajeros. La clase **Ferry** tendrá atributos como: máximo número de pasajeros, peso máximo de vehículos, número total de pasajeros, número total de vehículos, peso total de vehículos. Además, el Ferry guardará la lista de vehículos que transporta.

En la clase Ferry, implemente los siguientes métodos:

```
boolean embarcarVehiculo (Vehiculo v)
   // Embarca un vehículo añadiéndolo a la lista

int totalVehiculos()
   // devuelve el número total de vehículos embarcados

boolean vacio()
   // verdadero si no hay ningún vehículo

boolean superadoMaximoVehiculos()
  // verdadero si el número total de los vehículos supera el máximo

boolean superadoMaximoPeso()
  // verdadero si el peso total de los vehículos supera el máximo
```

Utilizando JUnit 5 implemente los casos de prueba necesarios con **tests parametrizados** para obtener un **100% de cobertura** de las clases Vehiculo y Ferry, usando los **valores límite** adecuados donde proceda.

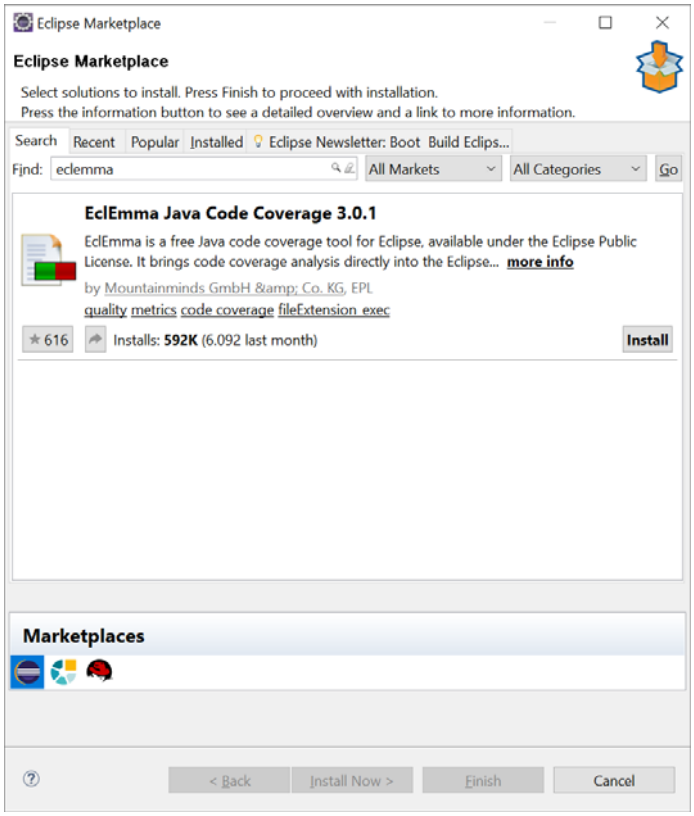
Entrega

En la tarea correspondiente a la sesio06 debe entregarse un pequeño informe con: las URLs de los repositorios utilizados (recuerda dar acceso al profesor si un repositorio es privado), y una captura de los resultados de cobertura en Eclipse.

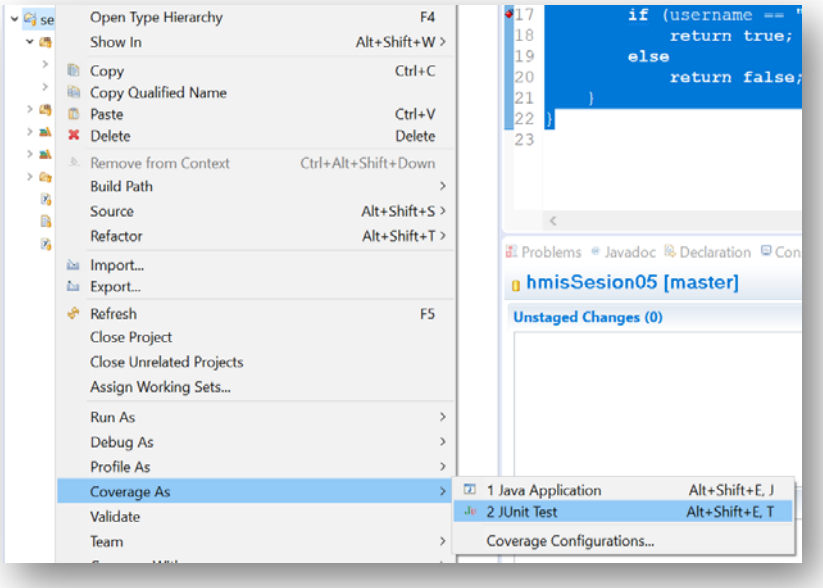
Información adicional

Plugin EclEmma para el cálculo de la cobertura de código

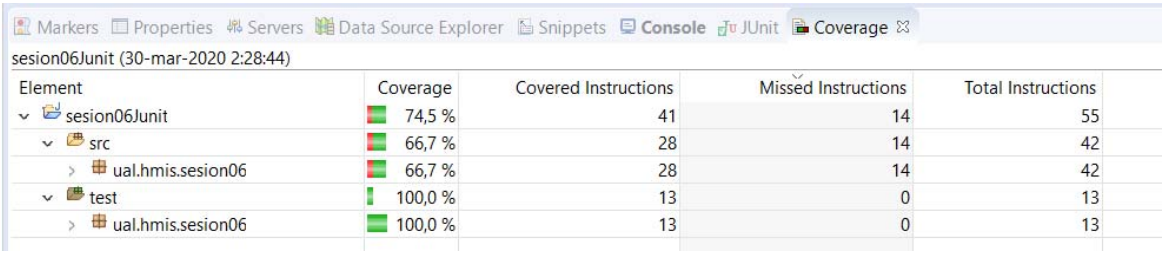
Desde el MarketPlace, instalar plugin EclEmma en cualquier Eclipse que tengas instalado.



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

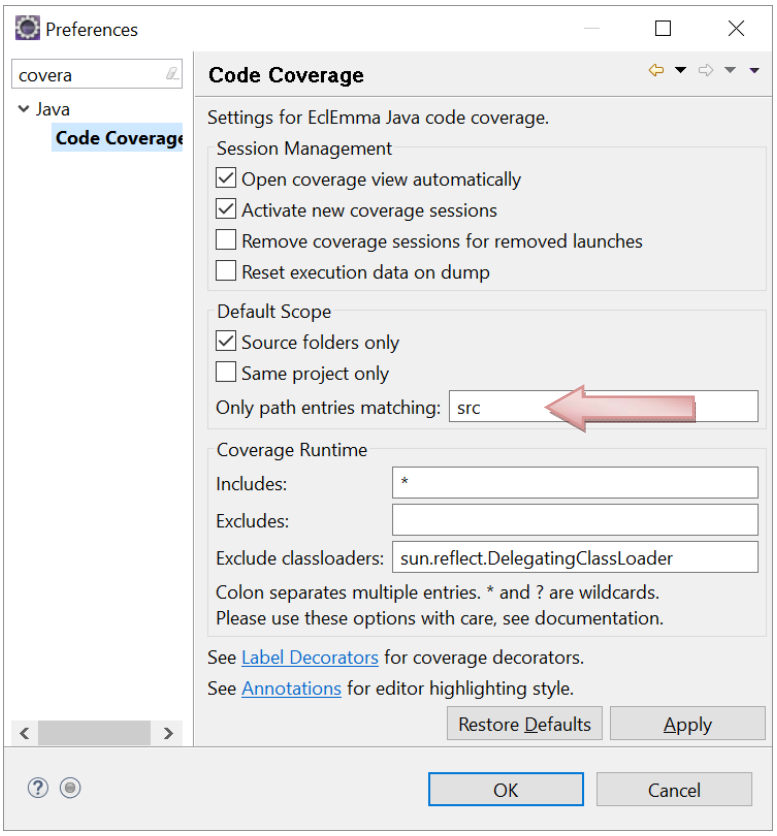


Y se muestra la vista de la métrica de cobertura



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
sesion06Junit	74,5 %	41	14	55
src	66,7 %	28	14	42
ual.hmis.sesion06	66,7 %	28	14	42
test	100,0 %	13	0	13
ual.hmis.sesion06	100,0 %	13	0	13

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

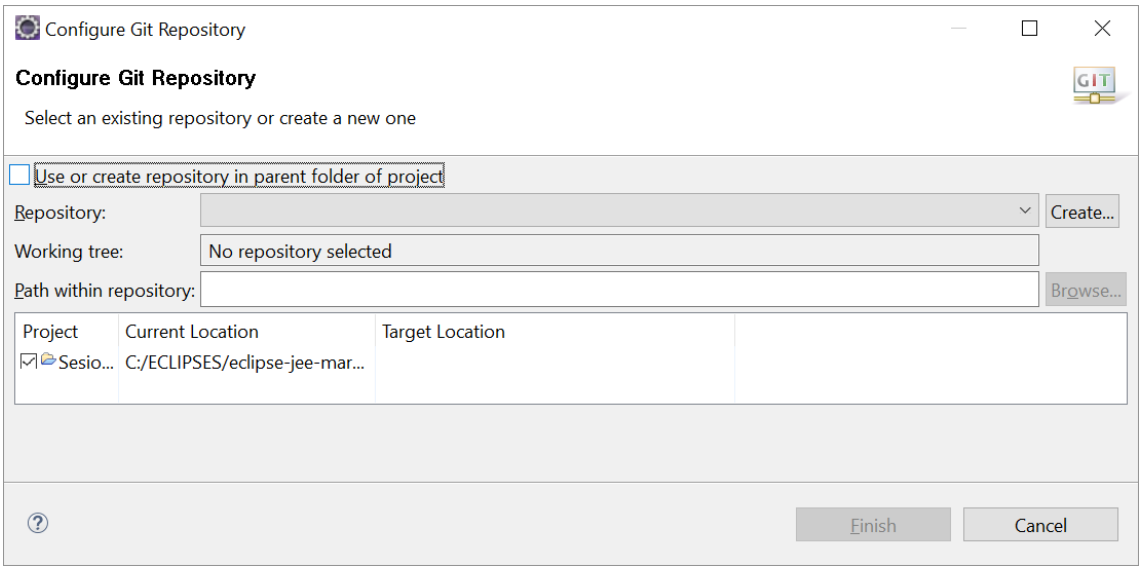


Creación del repositorio Git local y configuración de repo remoto desde Eclipse

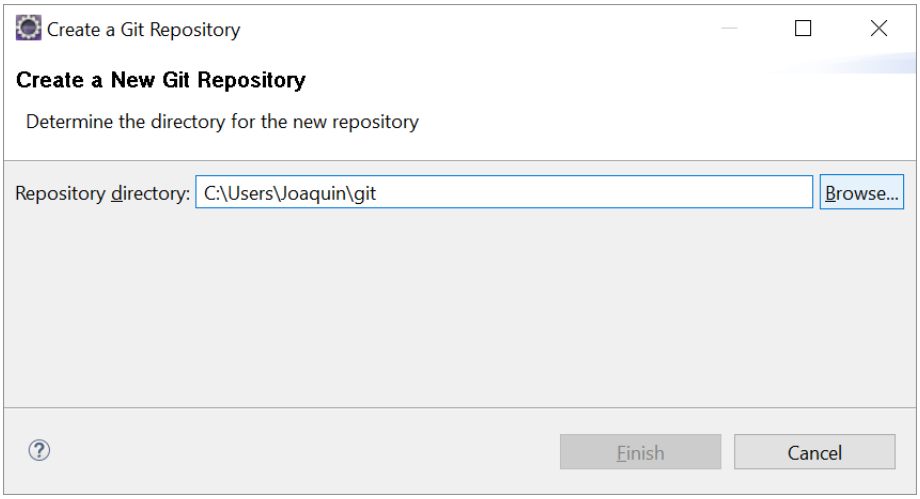
1) Creación del repositorio Git local y commits de los cambios:

Si queremos gestionar con Git un proyecto creado en Eclipse: sobre el proyecto, botón derecho, Team / Share project / Git

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



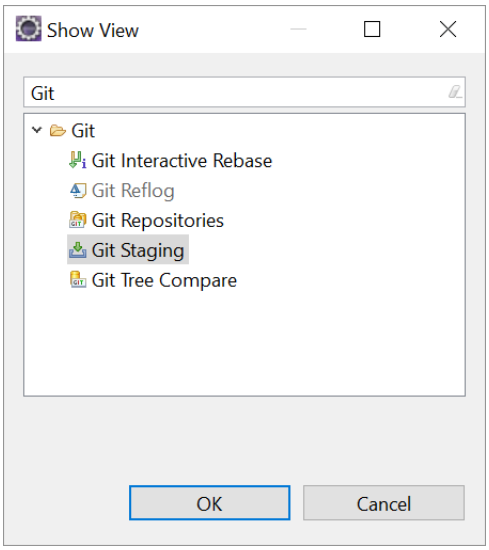
Create...



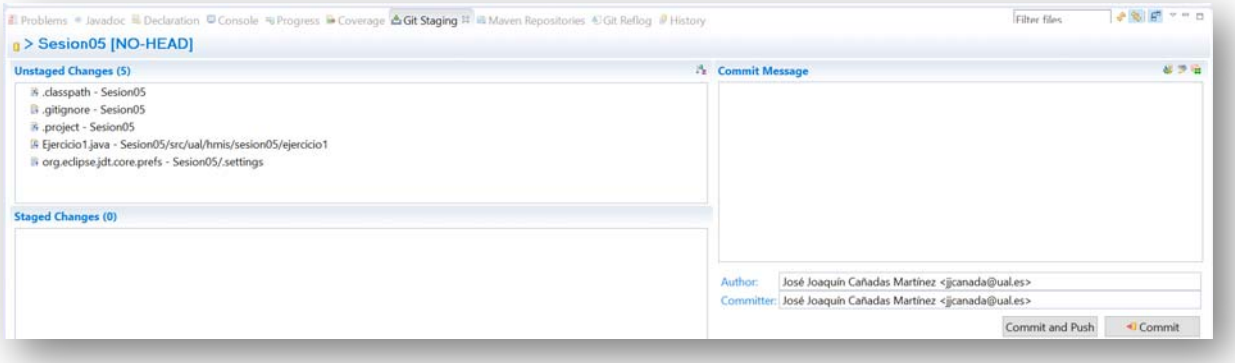
Browse...

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

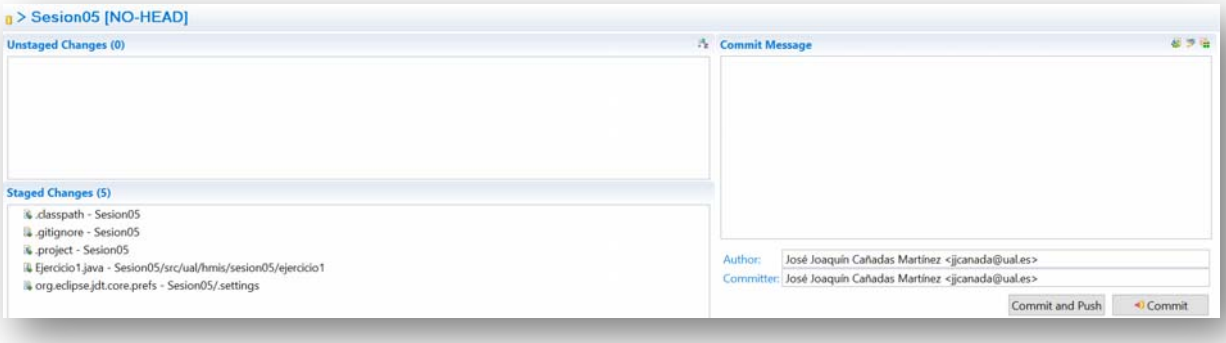
Habilitamos la vista de Git Staging: Window / Show view



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

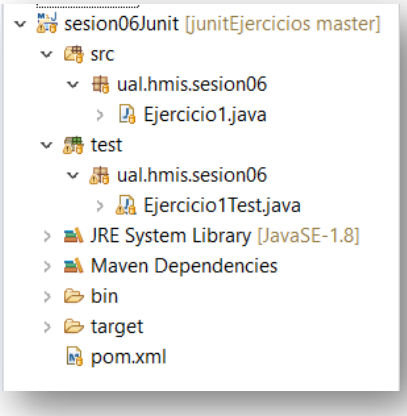


Los pasamos a "preparados" seleccionando los archivos, botón derecho, add to index



Escribimos un mensaje y click en Commit.

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

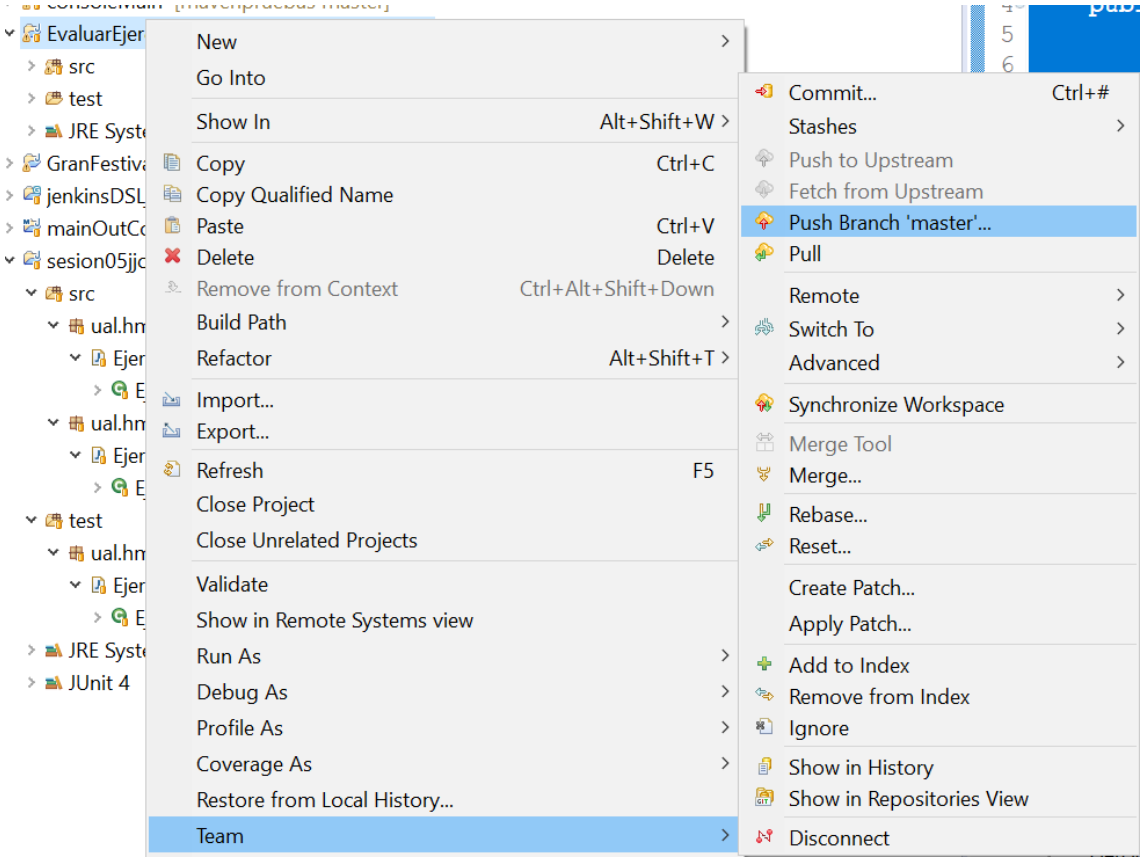


2) Creación el repositorio en GitHub

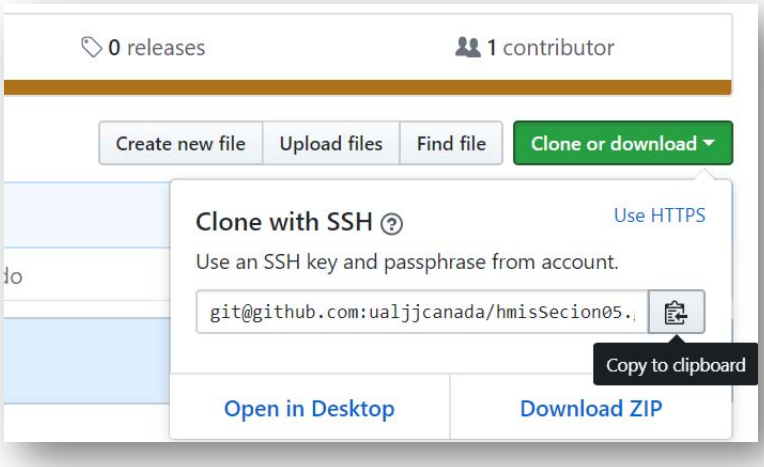
En nuestra cuenta de GitHub, creamos un nuevo repositorio y alternativamente lo marcamos como privado.

3) Configuración del repositorio remoto de GitHub y operación de push

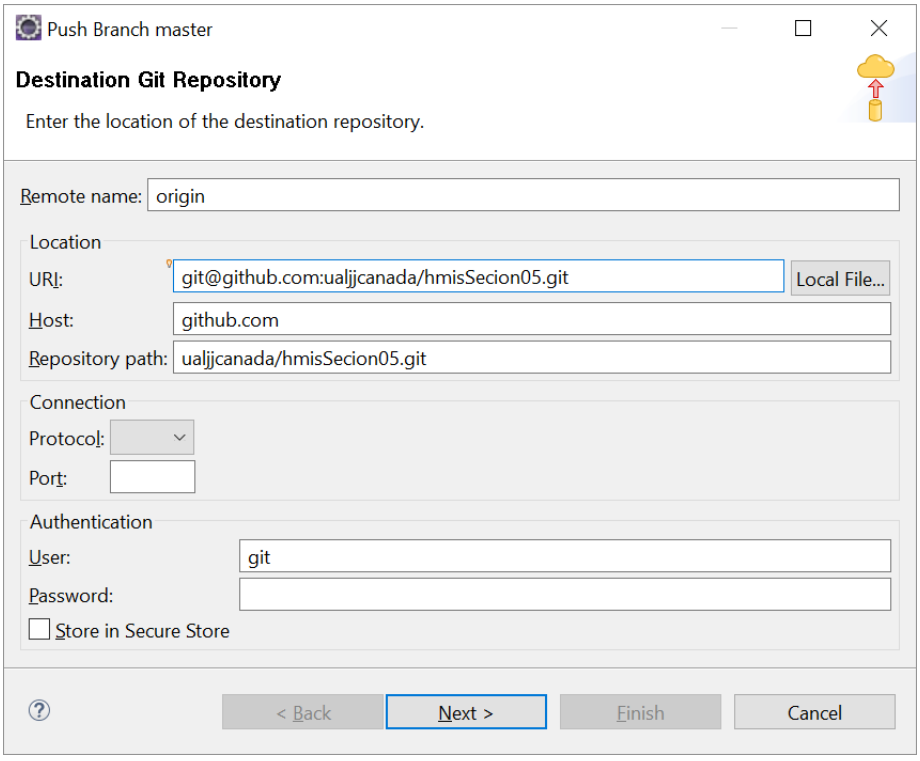
Sobre el proyecto, Team / Push Branch 'master'...



A continuación, copiamos la URL **SSH** del repositorio privado GitHub



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



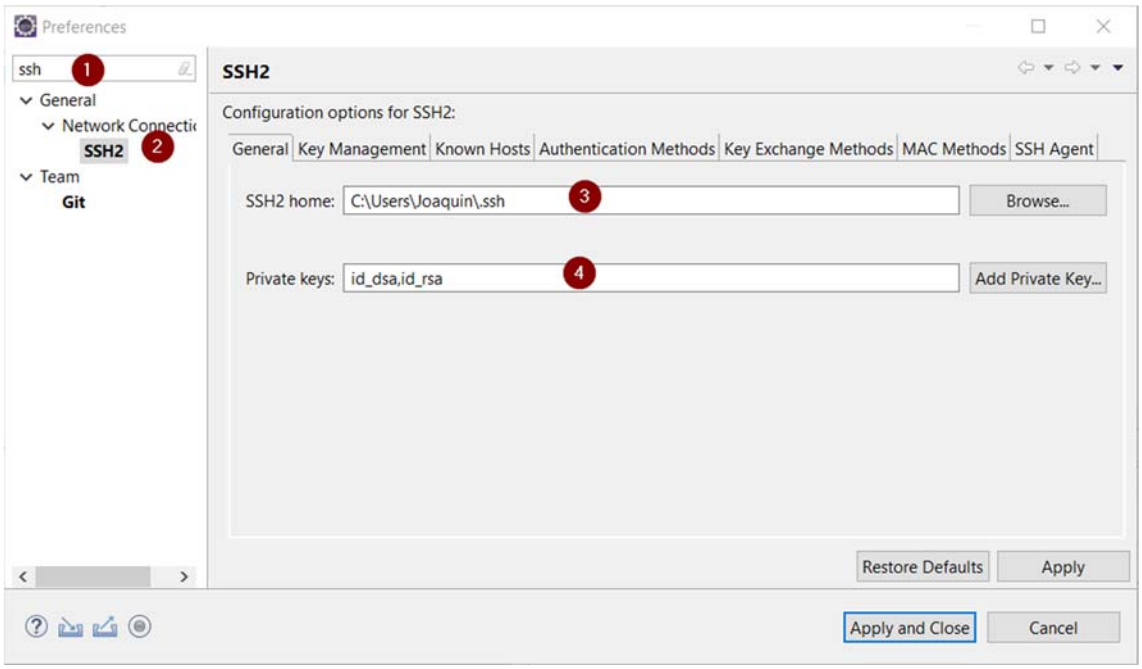
Next / Next / Finish

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

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

NOTA: si tienes algún **problema para hacer *push*** usando SSH y recibes un mensaje de error, debes revisar la configuración del SSH. Para ello, accede al menú **Window** / **Preferences**. Escribe **SSH.**

Comprueba que la carpeta SSH2 home, donde busca la clave privada, es correcta. Igualmente con los posibles nombres de archivo de la clave privada.


Preferences
ssh
1
General
Network Connecti
SSH2
2
Team
Git
SSH2
Configuration options for SSH2:
General
Key Management
Known Hosts
Authentication Methods
Key Exchange Methods
MAC Methods
SSH Agent
SSH2 home:
C:\Users\Joaquin\.ssh
3
Browse...
Private keys:
id_dsa,id_rsa
4
Add Private Key...
Restore Defaults
Apply
Apply and Close
Cancel