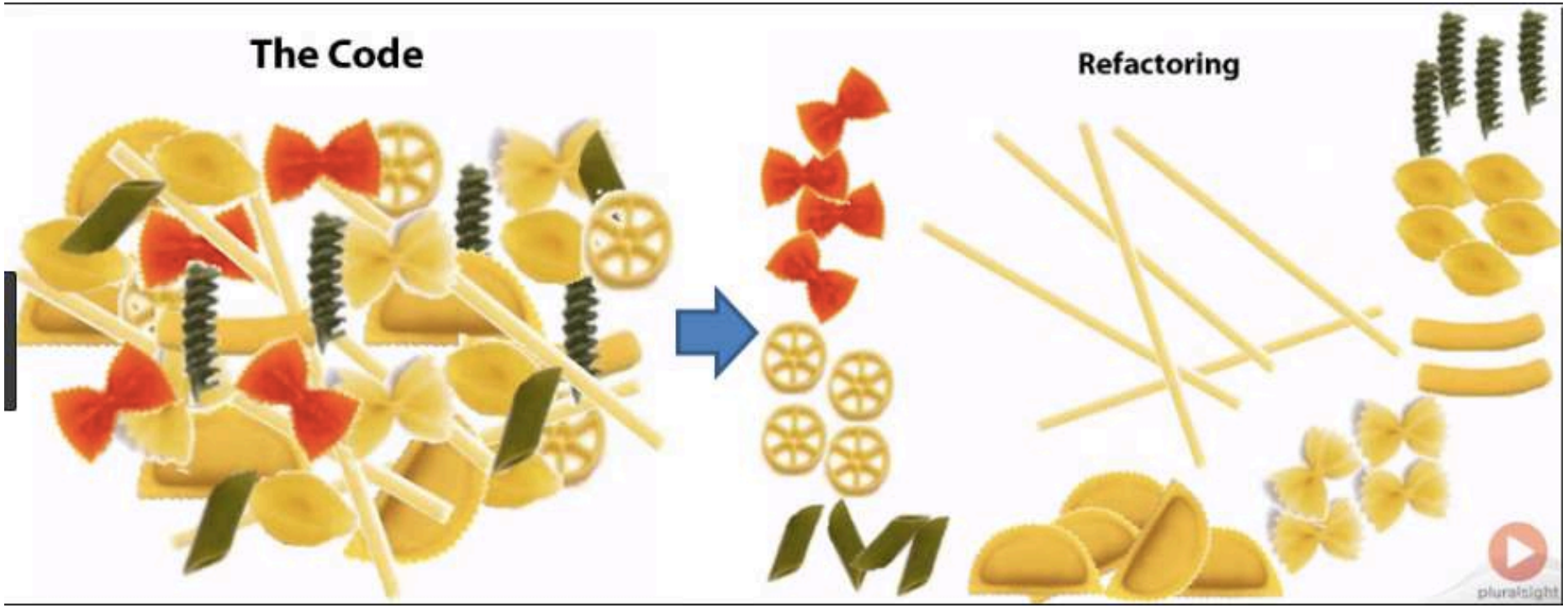
5. REFACTORIAZACIÓN OPTIMIZACIÓN Y DOCUMENTACIÓN

1. REFACTORIZACIÓN

- La refactorización consiste en realizar una transformación al software preservando su comportamiento, modificando su estructura interna para mejorarlo.
- En múltiples ocasiones necesitaremos revaluar y modificar código creado anteriormente, o trabajar con código de otra persona.
- La refactorización (informalmente llamada limpieza de código) ayuda a tener un código que es más sencillo de comprender, más compacto, más limpio y, por supuesto, más fácil de modificar.
- Su principal objetivo es que sea más fácil de entender y de modificar, permitiendo una mejor lectura para comprender qué es lo que se está realizando.
- Después de refactorizar el proyecto seguirá ejecutándose igual y obteniendo los mismos resultados

- ¿Qué hace la refactorización?
  - Limpia el código, mejorando la consistencia y la claridad
  - Mantiene el código, no corrige errores ni añade funciones nuevas
  - Va a permitir facilitar la realización de cambios en el código
  - Se obtiene un código limpio y altamente modularizado
- Por tanto, no cambia la funcionalidad del código ni el comportamiento del programa, el programa deberá comportarse de la misma forma antes y después de efectuar las refactorizaciones.


The Code
Refactoring
pluralsight

1.1 CUÁNDO REFACTORIZAR. MALOS OLORES

- La refactorización se debe ir haciendo mientras se va realizando el desarrollo de al aplicación. Hay ciertos síntomas que indican la necesidad de refactorizar que comúnmente se suelen llamar bad smells (malos olores). Algunos de estos síntomas son los siguientes:
- 1. Código duplicado (duplicated code).
  - Es la principal razón para refactorizar.
  - Si se detecta el mismo código en más de un lugar, se debe buscar la forma de extraerlo y unificarlo.
- 2. Métodos muy largos (Long method).
  - Cuanto más largo es un método más difícil es de entender.
  - Un método muy largo normalmente está realizando tareas que deberían ser responsabilidad de otros.
  - Se debe identificar y descomponer el métodos en otros más pequeños.
  - En POO cuanto más corto es un método más fácil es reutilizarlo.

- 3. Clases muy grandes (Large class).
  - Si una clase intenta resolver muchos problemas, tendremos una clase con demasiados métodos, atributos o incluso instancias (la clase está asumiendo demasiadas responsabilidades).
  - Hay que intentar hacer clases más pequeñas, de forma que cada una trate con un conjunto pequeño de responsabilidades bien limitadas.
- 4. Lista de parámetros extensa (Long parameter list).
  - En POO no se suelen pasar muchos parámetros a los métodos.
  - Tener demasiados parámetros puede estar indicando un problema de encapsulación de datos o la necesidad de crear una clase de objetos a partir de varios de esos parámetros y pasar ese objeto como argumento en vez de todos los parámetros (especialmente si esos parámetros tienen que ver unos con otros y suelen ir juntos siempre). Ejemplo: En lugar de pasar DNI, Nombre, Apellidos,… es mejor crear la clase persona y pasar como parámetro un Objeto de esta clase.

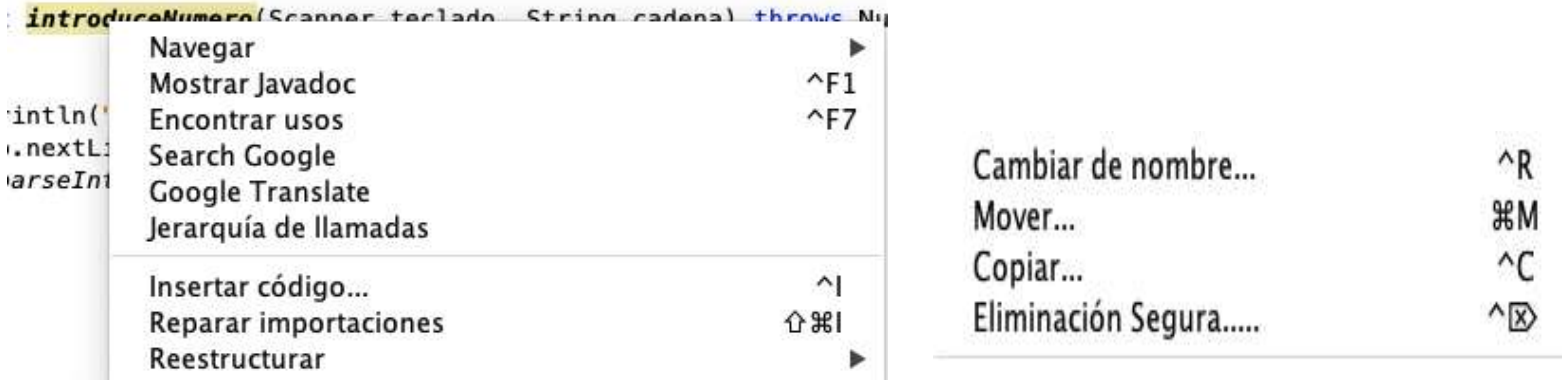
- 5. Cambio divergente (Divergent change).
  - Una clase es frecuentemente modificada por diversos motivos, los cuales no suelen estar relacionados entre sí. Quizás debemos plantearnos la necesidad de la clase y/o eliminarla.
- 6. Cirugía a tiro de pistola (Shotgun surgery).
  - Este síntoma se presenta cuando después de un cambio en una determinada clase, se deben realizar varias modificaciones adicionales en diversos lugares para compatibilizar dicho cambio.
- 7. Envidia de funcionalidad (Feature envy)
  - Se observa este síntoma cuando tenemos un método que utiliza más cantidad de elementos de otra clase que de la suya propia. Se suele resolver el problema pasando el método a la clase cuyos elementos utiliza más.

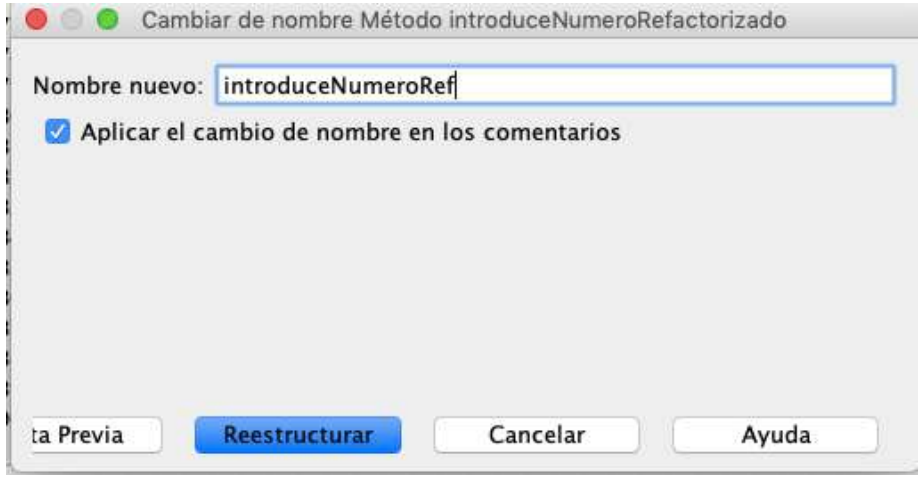
- 8. Clase de solo datos (Data class).
  - Clases que solo tienen atributos y cuyos métodos son solo para acceder a estos atributos (generalmente “get” y “set”). Estas clases deberían cuestionarse dado que no suelen tener comportamiento alguno.
- 9. Legado rechazado (Refused bequest).
  - Este síntoma lo encontramos en subclases que utilizan solo unas pocas características de sus superclases.
  - Si las subclases no necesitan o no requieren todo lo que sus superclases les proveen por herencia, esto suele indicar que como fue pensada la jerarquía de clases no es correcto.
  - La delegación suele ser la solución a este tipo de inconvenientes.
- Podemos ver que el proceso de refactorización presenta algunas ventajas, entre las que se encuentran el mantenimiento del diseño del sistema, incremento de la facilidad de lectura y comprensión del código fuente, detección temprana de fallos o aumento en la velocidad en la que se programa.

1.2 MÉTODOS O PATRONES DE REFACTORIZACIÓN

- Se tratan de diversas prácticas concretas que nos ayudan a refactorizar nuestro código.
- Algunos de los métodos más comunes:
- **1. Rename**.
  - Es una de las opciones más utilizadas. Cambia el nombre de variables, clases, métodos, paquetes y casi cualquier identificador Java.
  - Tras la refactorización, se modifican las referencias a ese identificador, por lo que nos libra de tener que hacer eso nosotros de manera manual (modifica todo el código del proyecto para referenciar al nuevo nombre).
  - Para utilizar el rename en Netbeans, seleccionamos el texto que queremos renombrar, pulsamos botón derecho y elegimos la opción Reestructurar (la opción de menú que indica refactorizar) y elegimos la opción Cambiar de nombre. También se puede hacer directamente seleccionando y pulsando Ctrl+R

- Para utilizar el rename en Netbeans, seleccionamos el texto que queremos renombrar, pulsamos botón derecho y elegimos la opción Reestructurar (la opción de menú que indica refactorizar) y elegimos la opción Cambiar de nombre. También se puede hacer directamente seleccionando y pulsando Ctrl+R



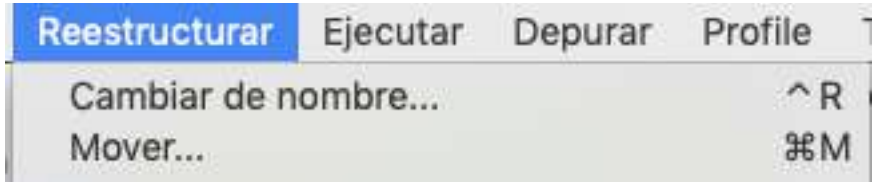


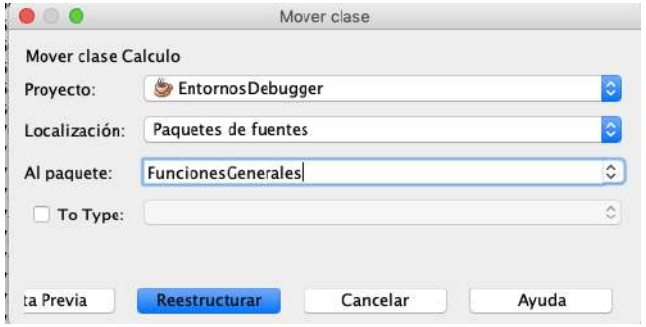

```java
public class Ejercicio1_calculoMayor_Dos_Numeros {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner teclado=new Scanner(System.in);

        int primerNumero;
        primerNumero=Calculo.introduceNumeroRef(teclado,"primer");
```

- **2.Mover clase.** Mueve una clase de un paquete a otro, se mueve el archivo .java a la carpeta, y se cambian todas las referencias. También se puede arrastrar y soltar una clase a un nuevo paquete, se realiza una refactorización automática.



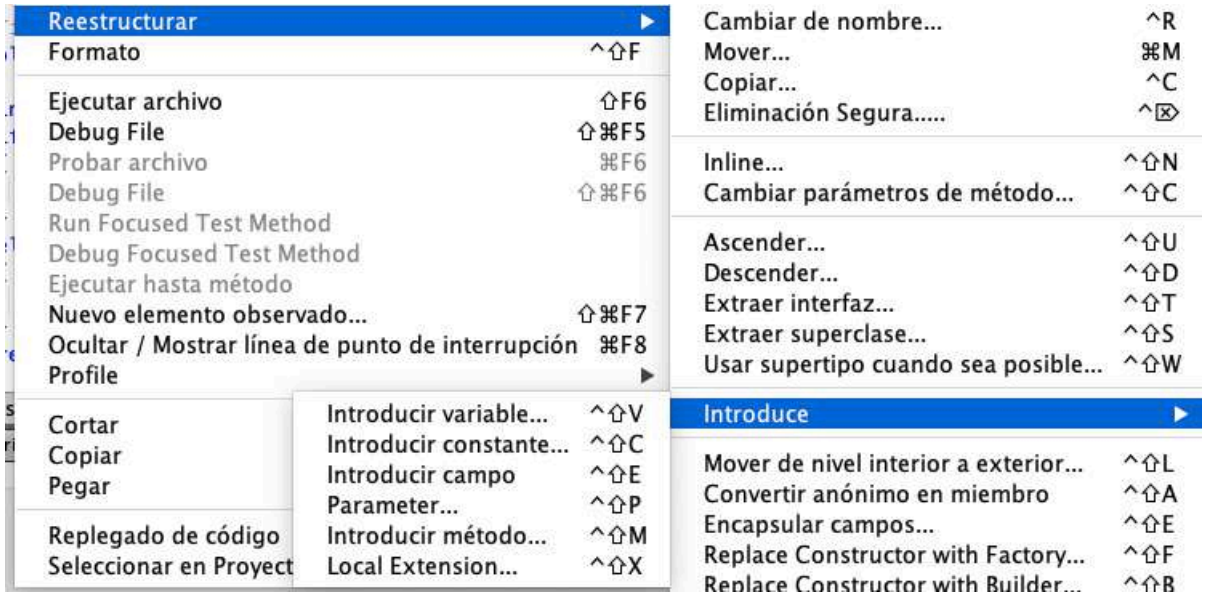


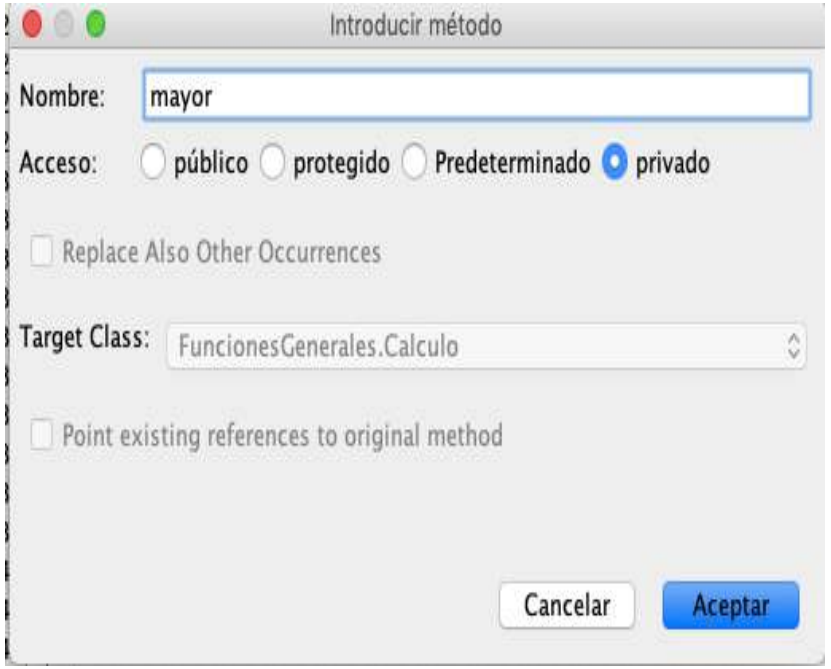
- En clases donde se hace referencia se puede ver la refactorización

```
import FuncionesGenerales.Calculo;
```

- **3. Extraer método (Extract Method).** Nos permite seleccionar un bloque de código y convertirlo en un método. El bloque de código no debe dejar llaves abiertas. Es muy útil cuando se crean métodos muy largos, que se podrán dividir en varios métodos o cuando se tiene un grupo de instrucciones que se repite varias veces.
- Seleccionamos el código, botón derecho -> Reestructurar -> Introduce -> Introducir método. Habrá que indicar el nombre del nuevo método y el modificador de acceso (público, protegido, privado, o sin modificador)

```java
public static int calcularMayor(int x, int y)
{
  int my;
  if(x>y)
  {
      my=x;
  }
  else
  {
      my=y;
  }
  return my;
}
```

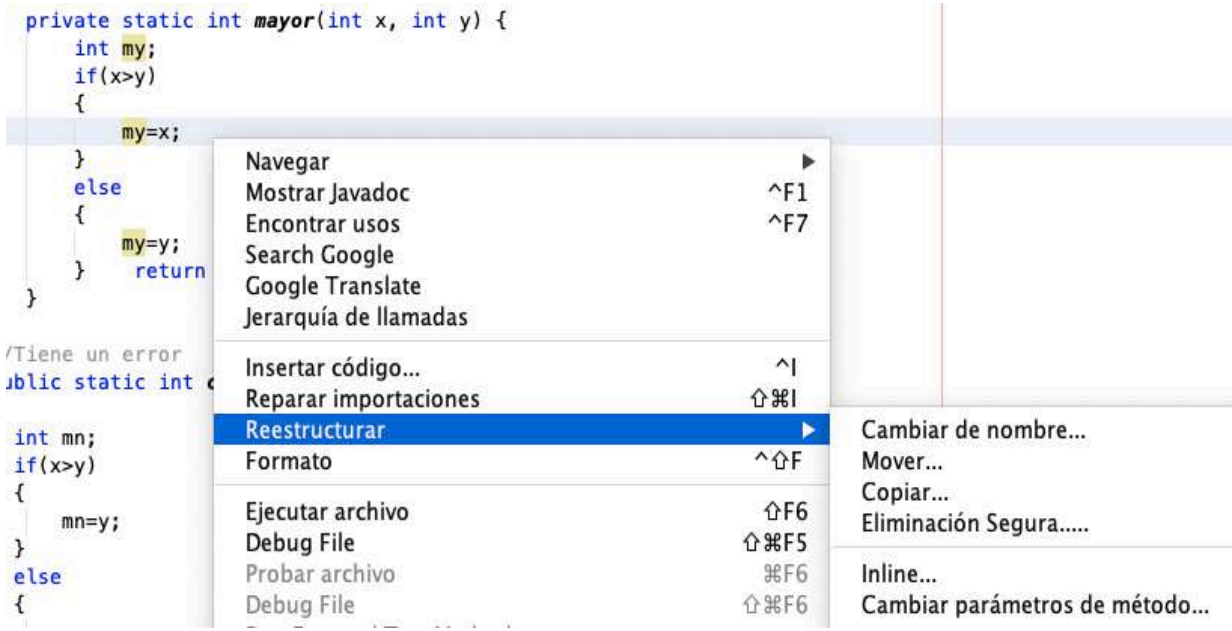




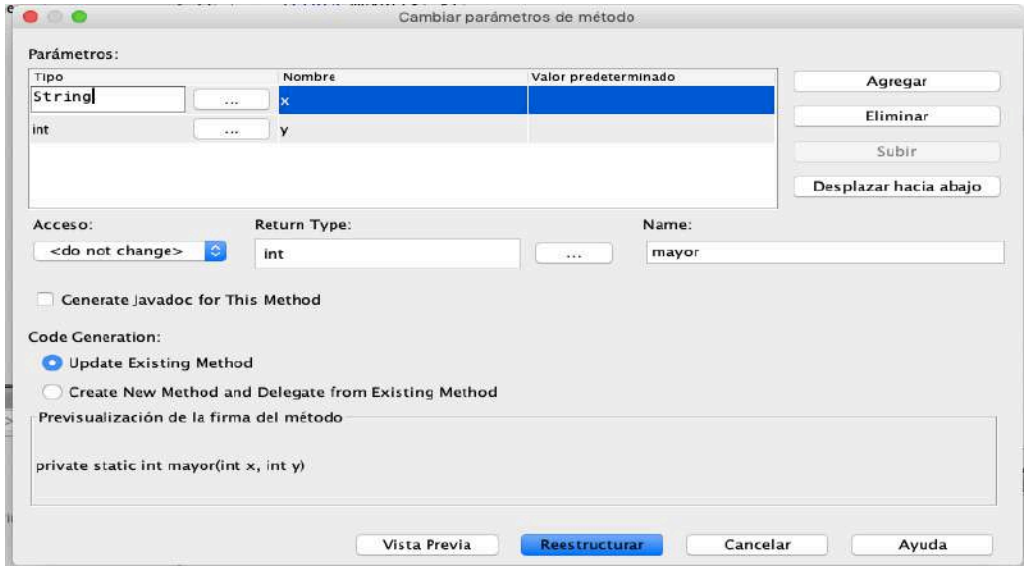

```java
public static int calcularMayor(int x, int y)
{
  int my;
  return mayor(x, y);
}

   private static int mayor(int x, int y) {
        int my;
        if(x>y)
        {
            my=x;
        }
        else
        {
            my=y;
        }    return my;
   }
```

- **4. Cambiar parámetros de método (Change method signature).** Permite cambiar la firma de un método, es decir, el nombre del método y los parámetros que tiene. De forma automática se actualizarán todas las dependencias y llamadas al método dentro del proyecto.



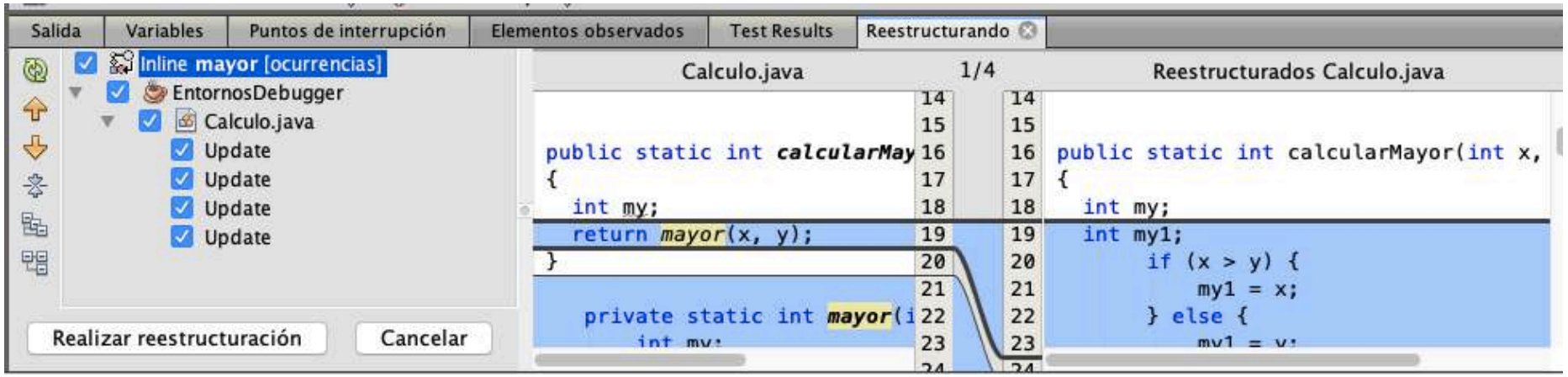
- Netbeans no será capaz de resolver todos los cambios que se realicen. Si cambias por ejemplo el tipo de una variable es posible que debas realizar posteriormente cambios en tu código.




```
public static int calcularMayor(int x, int y)
{
  int my;
  return mayor(x, y);
}

   private static int mayor(String x, int y) {
        int my;
        if(x>y)
        {
            my=x;
        }
        else
        {
            my=y;
        }    return my;
   }
```

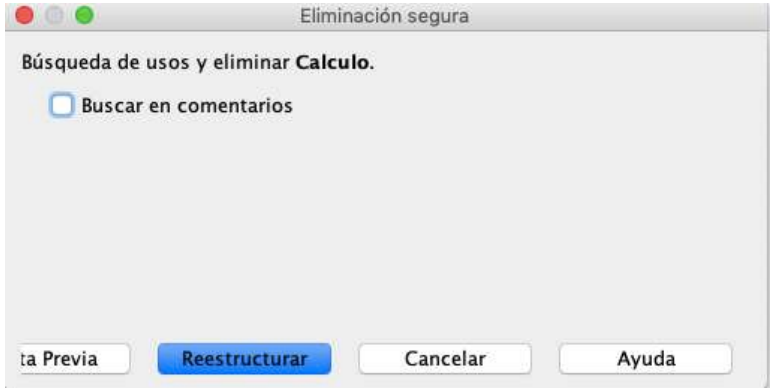
- **5. Inline.** Pone el cuerpo de un método en el cuerpo de quién lo invoca. Si al método solo se llama desde allí se elimina el mismo.

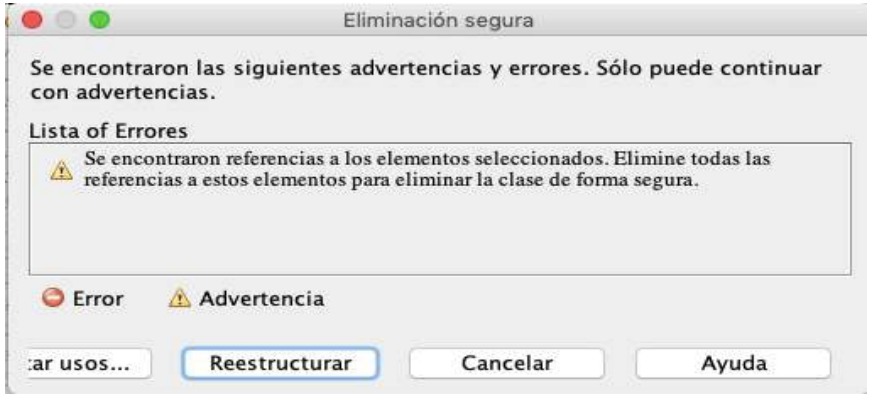


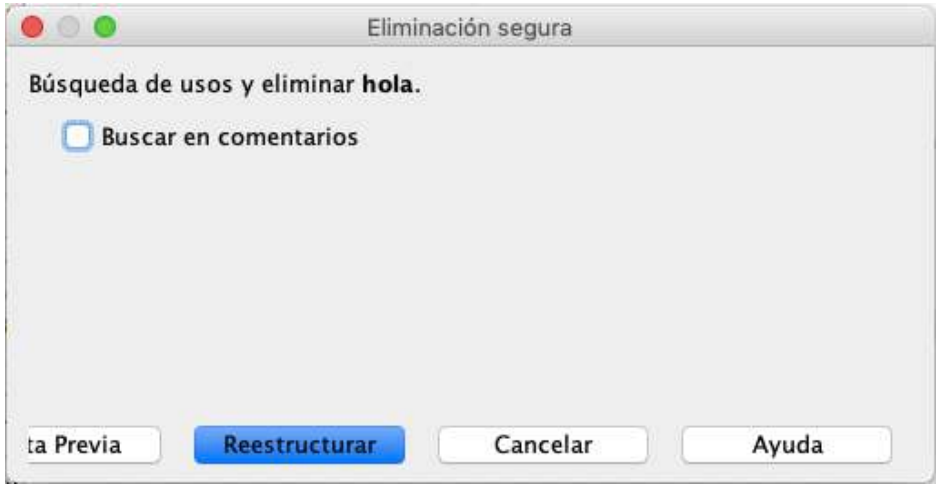

```java
public static int calcularMayor(int x, int y)
{
   int my;
   int my1;
         if (x > y) {
             my1 = x;
         } else {
             my1 = y;
         }
   return my1;
}
```

- Las razones por las cuales aplicar uno (extract method) u otro (inline method) dependen totalmente de el objetivo de la aplicación, por lo que, para saber cuando hacer o no un refactoring es muy importante conocer que se intenta hacer con el software.
- La idea detrás de esto es que el diseño mejore. No tiene caso ir ciegamente un día quitando métodos y al siguiente ponerlos si no hay una razón para ello.
- Otro aspecto fundamental es no romper la aplicación existente, debe de haber forma de probar la aplicación para que después de hacer el refactoring se vea si sigue funcionando igual o no. Si no es posible garantizar que el software sigue corriendo igual entonces es mejor no hacer el refactor.

- **6. Eliminación segura.** Nos permite borrar una clase, método, variable, etc. de una manera segura. Para ello hace una búsqueda para ver si es referenciado y nos dará información para ver si debemos o no eliminar
  - Ejemplo. Voy a tratar de borrar con la Eliminación Segura una clase que sí es referenciada lo cuál me será indicado y a continuación un nuevo método que he definido y que no se usa en ningún sitio por lo que lo eliminará sin problema.

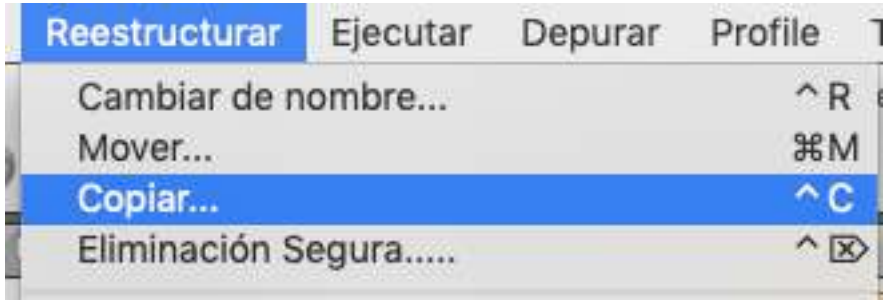






- Tras pulsar Reestructurar este nuevo método ha desaparecido de la clase de manera instantánea.

- **7. Copiar**. Nos permite copiar la clase al mismo proyecto. A la hora de copiar nos permite indicar un nuevo nombre para el destino que tendrá la copia correspondiente.



8. Otras refactorizaciones

- Ascender. Permite subir un método o campo a otra clase de la cual hereda la clase que contiene al método o campo que deseamos subir.
- Descender. Permite bajar una clase anidada, método o campo a otra clase la cual hereda de la clase que contiene a la clase anidada, método o campo que deseamos bajar.
- Extraer interfaz. Nos permite escoger los métodos de una clase para crear una Interface. Una Interface es una especie de plantilla que define los métodos acerca de lo que puede hacer o no hacer una clase (los define pero no los desarrolla)
- Extraer superclase. Este método permite extraer una superclase. Se pueden seleccionar los métodos y atributos que formaran parte de la superclase (también hará que la clase refactorizada la extienda.)

2. CONTROL DE VERSIONES

- Se puede definir control de versiones como la capacidad de recordar todos los cambios que se realizan tanto en la estructura de directos como en el contenido de los archivos.
- Esto es de mucha utilizad cuando se desea recuperar un documento, o una carpeta, o un proyecto en un momento concreto para su desarrollo.
- También es muy útil cuando se necesita mantener un cierto control de los cambios que se realizan sobre documentos, archivos o proyectos que comparten varias personas o un equipo de trabajo: Se hace necesario saber qué cambios se hacen, quién los hace y cuándo se realizan.

2.1 TERMINOLOGÍA

- **1. Repositorio.** Lugar donde se almacenan todos los datos y los cambios. Puede ser un sistema de archivos en un disco duro, un banco de datos, un servidor, etc.
- Cada vez que se actualiza el repositorio, este recuerda cada cambio realizado en el archivo o estructura de directorios. Además, permite establecer información adicional por cada actualizar, pudiendo tener un changelog de las versiones en el propio repositorio.
- Cada herramienta de control de versiones tiene su propio repositorio y, por desgracia, no son interoperables, es decir, no puedes obtener los datos del repositorio o actualizar si el repositorio y el control de versiones no coinciden.
- Al ser un sistema de archivos nosotros podemos acceder a dichos archivos en cualquier momento, pero es al asociar ese sistema de archivos al control de versiones lo que nos permite llevar el control adecuado con los mismos.

- **2. Revisión o versión.** Una revisión es una versión concreta de los datos almacenados. Algunos sistemas identifican las revisiones con un número contador (como Subversión). Otros identifican las revisiones mediante un código de detección de modificaciones (en Git usa SHA1). La última versión se la identifica como la cabeza o HEAD.
- **3. Etiquetar o roturlar (tag).** Cuando se crea una versión concreta en un momento determinado del desarrollo de un proyecto se le pone una etiqueta, de forma que se pueda localizar y recuperar en un momento. Las etiquetas permiten identificar de forma fácil revisiones importantes en el proyecto (por ejemplo una versión publicada).
- **4. Tronco (trunk).** Es el tronco o la línea principal de desarrollo de un proyecto.

- **5. Rama o ramificar (branch).** Las ramas son copias de archivos, carpetas o proyectos. Cuando se crea una rama se crea una bifurcación del proyecto y se crean dos líneas de desarrollo. Son motivos habituales de creación de ramas la creación de nuevas funcionalidades o la corrección de errores.
- **6. Desplegar (Checkout).** Crea una copia de trabajo del proyecto, o de archivos y carpetas del repositorio en el equipo local. Por defecto se obtiene la última versión, aunque también se puede indicar una versión concreta. Con el checkout se vincula la carpeta de trabajo del equipo local con el repositorio y se crean los metadatos de control de versiones (carpetas y archivos ocultos que se crean).
- **7. Confirmar (commit o check-in).** Se realiza commit cuando se confirman los cambios realizados en local para integrarlos al repositorio.

- **8. Exportación (export)**. Similar a Checkout, pero en esta ocasión no vincula la copia con el repositorio. Es una copia limpia sin los metadatos de control de versiones.
- **9. Importación (import).** Es la subida de carpetas y archivos del equipo local al repositorio. Se puede hacer en cualquier momento desde el sistema de archivos.
- **10. Actualizar (update).** Se realiza una actualización cuando se desea integrar los cambios realizados en el repositorio en la copia de trabajo local. Los cambios puede ser realizados por personas del equipo de trabajo.
- **11. Fusión (merge).** Una fusión consiste en unir los cambios realizados sobre uno o varios archivos en una única revisión. Se suele realizar cuando hay varias líneas de desarrollo separadas en ramas y en alguna etapa se necesitan fusionar los cambios hechos entre ramos o en una rama con el tronco principal, o viceversa.

- **12. Conflicto.** Ocurre cuando dos usuarios crean una copia local (Checkout) de la misma versión de un archivo, uno de ellos realiza cambios y envía los cambios (commit) al repositorio, y el otro no actualiza (update) esos cambios y realiza cambios sobre el archivo e intenta enviar luego sus cambios al repositorio.
- Entonces se produce el conflicto y el sistema no es capaz de fusionar los cambios. Este usuario deberá resolver el conflicto combinando los cambios o eligiendo uno de ellos.
- **13. Resolver conflicto.** La actualización del usuario para atender un conflicto entre diferentes cambios al mismo documento.

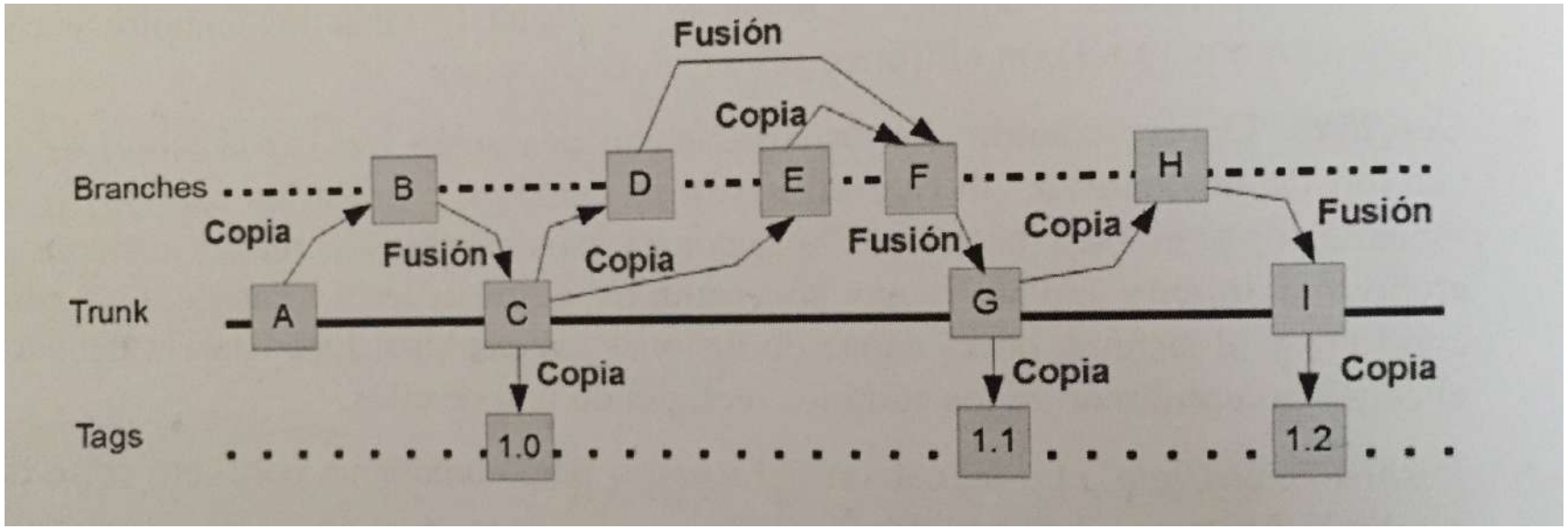
- Para trabajar en proyectos utilizando un sistema de control de versiones, lo primero que hay que hacer es crearse una copia en local de la información del repositorio con checkout, de esta manera se vincula la copia con el repositorio.
- A continuación, el usuario realizará sus modificaciones y una vez finalizadas sube las modificaciones al repositorio con commit.
- Si la copia del usuario ya está vinculada la repositorio, antes de modificar y realizar cambios tiene que hacer Update, para asegurarse que los cambios se están realizando sobre la versión última del repositorio y evitar así los conflictos de trabajar con una versión antigua.

2.2 SUBVERSIÓN

- Subversión es una herramienta multiplataforma (Windos, Linux, Max, etc.) de código abierto para el control de versiones.
- Usa una base de datos central, el repositorio, que contiene los archivos cuyas versiones y respectivas historias se controlan.
- El repositorio actúa como un servidor de ficheros, con la capacidad de recordar todos los cambios que se hacen tanto en sus directorios como en sus ficheros.
- Un proyecto es subversión debe verse como un árbol que tiene su tronco (**trunk**) donde está la línea principal de su desarrollo; que tiene sus ramas (**branches**) en las que se añadirán nuevas funciones o se corregirán errores; y que además tiene sus etiquetas **(tags)** para marcar situaciones importantes, o versiones finalizadas.

- Así la estructura de carpetas recomendada en la creación de proyectos utilizando estas herramientas y la funcionalidad que se le debe dar a cada carpeta dentro del repositorio son las siguientes:
  - **Trunk (tronco).** Base común para guardar las carpetas del proyecto o trabajo a controlar. Es donde está la versión básica, es decir, la rama de desarrollo principal.
  - **Tags (etiquetas).** Una etiqueta es una copia del proyecto, de una carpeta o del archivo que se hace con el objetivo de obtener una versión que no se va a modificar. Debe ser copias del tronco (trunk). Útil para crear versiones ya finalizadas, aquí se guardarán las versiones cerradas de los proyectos
  - **Branches (ramas).** En las ramas se desarrollan versiones que luego se van a publicar. Es una copia del tronco, de un proyecto, de una carpeta o de un archivo con la intención de modificar sobre ella, para conseguir un producto final diferente y alternativo al original.

- Vamos a ver un posible ciclo de vida de subversión

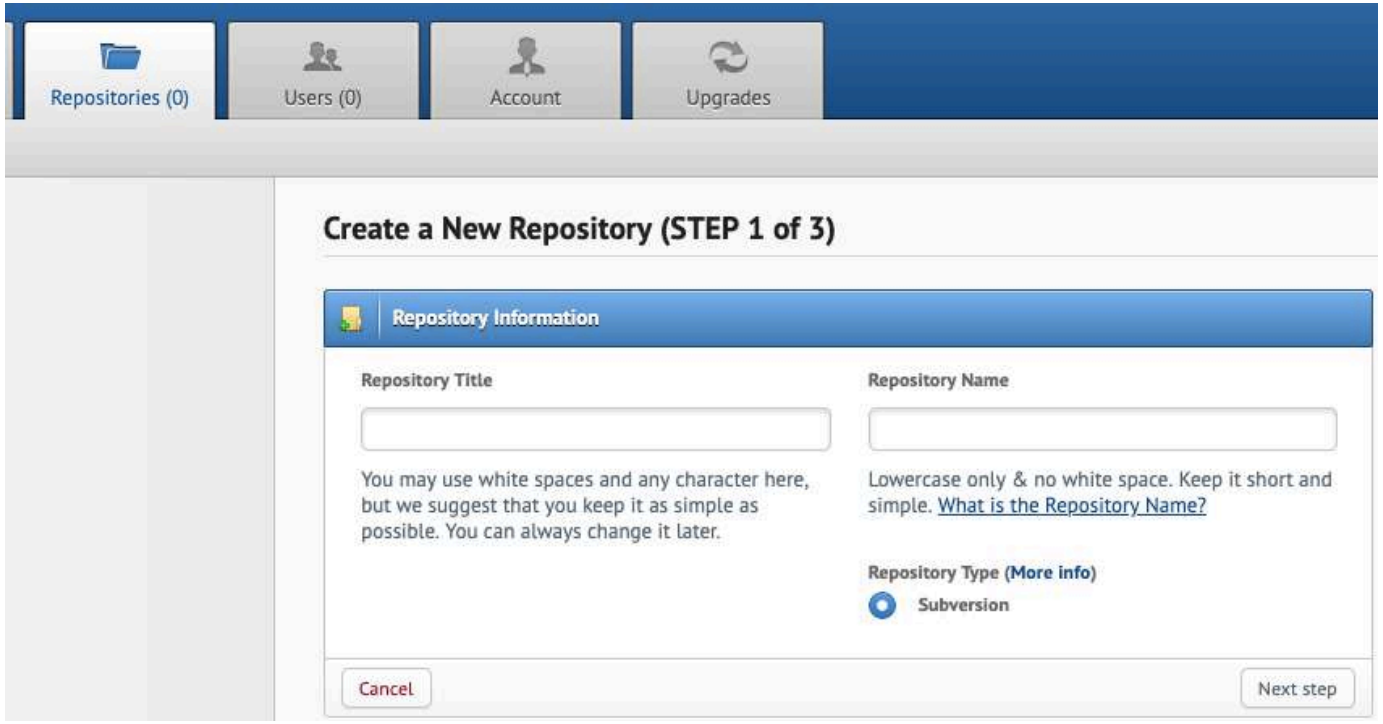


- A. Partimos del desarrollo inicial en el trunk
- B. Se crea una rama porque hay que añadir una nueva funcionalidad.
- C. Mientras tanto se ha corregido un bug en el tronco principal (C). Una vez que está lista la rama se fusiona con el trunk. Cuando esté lista esta versión libre de bugs y con nueva funcionalidad se crea la primera versión disponible para el público (Tag 1.0)
- D. Después de salir la primera versión se han detectado nuevos bugs, y se necesita añadir nuevas funcionalidades. Entonces se crea una rama para desarrollar una nueva funcionalidad.
- E. Se ha creado otra rama porque se necesitan otras funcionalidades

- F. Se realiza la fusión de las dos ramas, primero se realiza una copia de ellas y luego se fusiona con la otra.
- G. Se incorporan las nuevas funcionalidades al tronco principal. Una vez que está lista esta revisión se crea una nueva versión para el público (Tag 1.1).
- H. Se han detectado nuevos bugs y se necesita añadir nuevas funcionalidades. Se crea una nueva rama para desarrollar una nueva funcionalidad.
- I. Se incorporan las nuevas funcionalidades al tronco principal. Se crea una nueva versión para el publico (Tag 1.2).

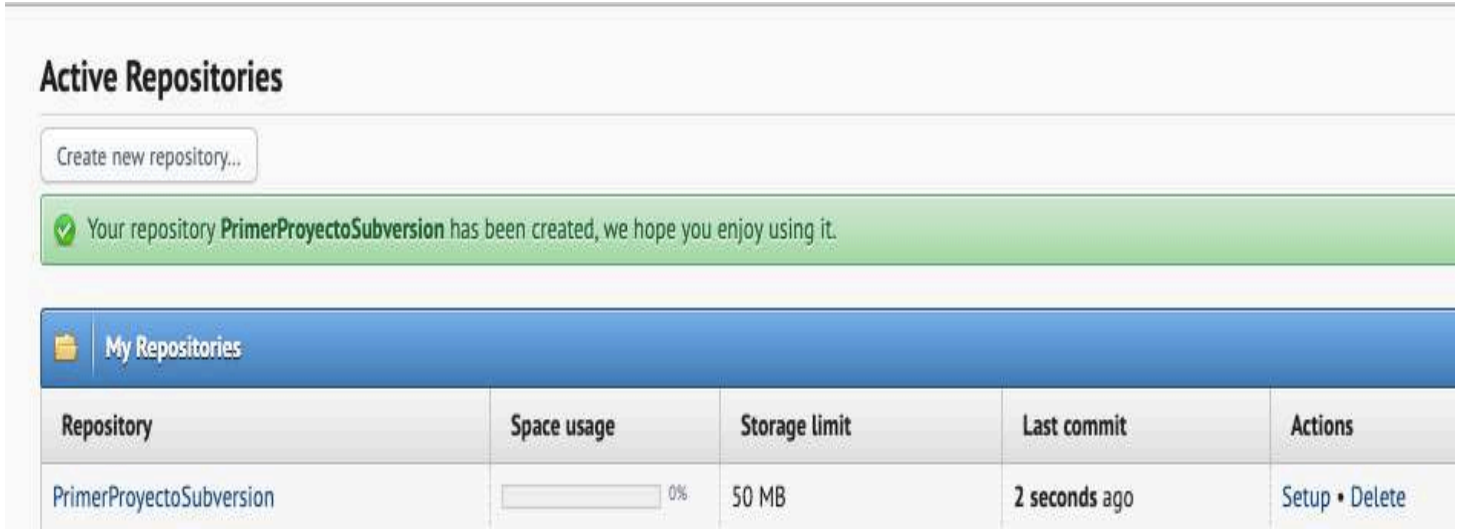
2.2.1 EJEMPLO SUBVERSIÓN EN NETBEANS

- Para no tener que montar un servidor subversión en local (que podríamos hacerlo sin ningún tipo de problema) vamos a utilizar uno gratuito en la nube. Al ser un servicio gratuito tiene algunas limitaciones pero para nuestro proyecto será suficiente. Para nuestro caso vamos a utilizar: https://riouxsvn.com/
- Nos tenemos que crear una cuenta (Sign up) y nos llegará un código de activación.
- Una vez que hemos creado la cuenta y nos hemos logeado, lo primero que haremos será crear un repositorio que es lugar donde iremos almacenando todos los datos y los cambios que se vayan realizando.
- Para ello nos iremos a la pestaña de Repositories y pulsaremos sobre crear nuevo repositorio


Repositories (0)
Users (0)
Account
Upgrades
Create a New Repository (STEP 1 of 3)
Repository Information
Repository Title
Repository Name
You may use white spaces and any character here, but we suggest that you keep it as simple as possible. You can always change it later.
Lowercase only & no white space. Keep it short and simple. What is the Repository Name?
Repository Type (More info)
Subversion
Cancel
Next step

- Podemos indicar como título PrimerProyectoSubversion y como nombre proyectosvn1 y pulsaremos en Next Step.
- En el segundo paso podemos indicarle que nos cree que ya el tronco, las ramas y las etiquetas de directorio pero vamos a dejarlo desmarcado como está.

- En el último paso podemos indicar si queremos que nos notifiquen cualquier cambio que se haga en el repositorio vía e-mail. Lo dejamos tal y como está y tras pulsar en Confirm Creation ya tendríamos el repositorio creado.



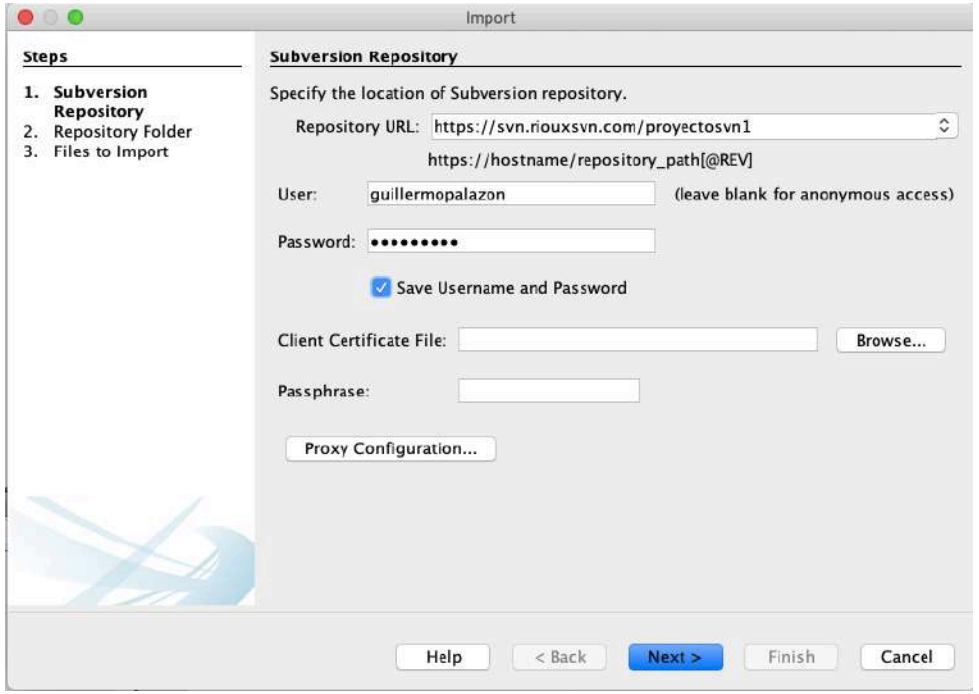
- Si pulsamos en el menú de la izquierda sobre el repositorio nos lleva a un formulario de actividad en el que podemos comprobar que aún no hemos realizado ningún cambio

- En este formulario tenemos un dato MUY IMPORTANTE que es la URL del repositorio de subversión

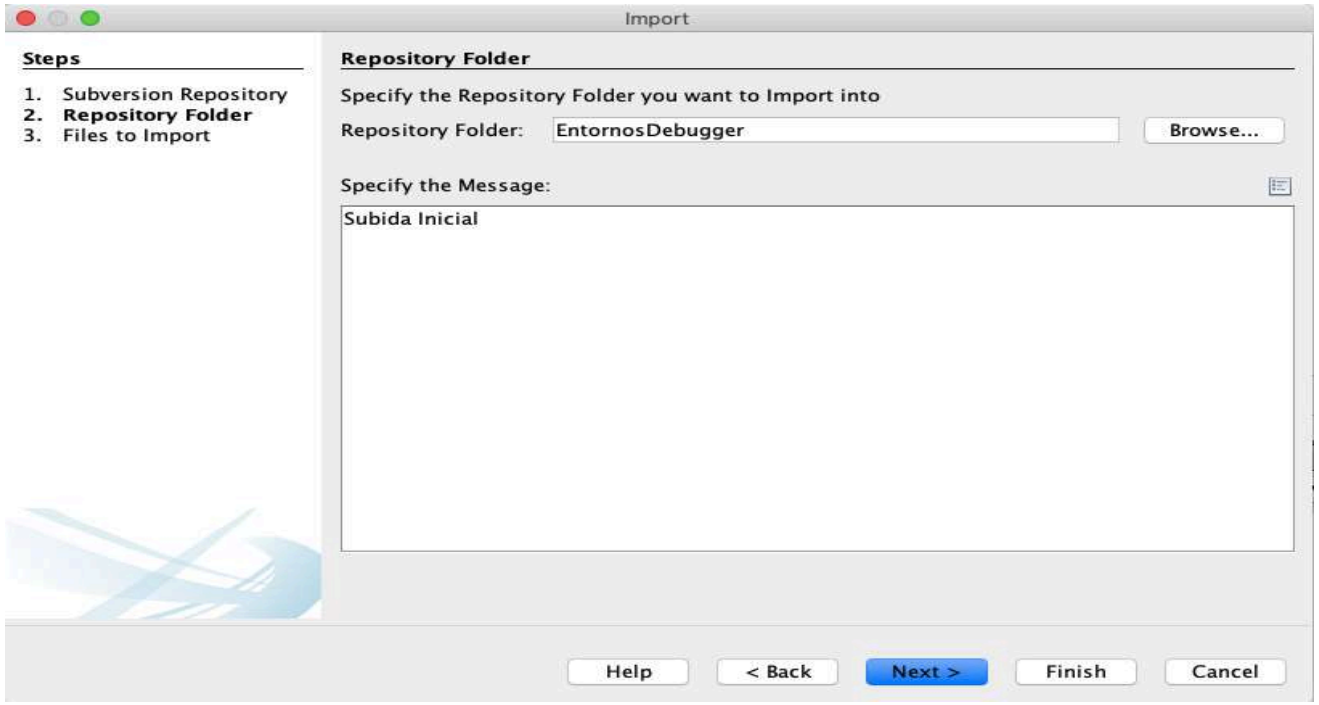
- Esta es la dirección que utilizaremos en NetBeans, es decir, será el repositorio que sincronizaremos con NetBeans.
- Copiaremos esta dirección y la utilizaremos en un proyecto que tengamos ya creado en NetBeans y que queramos subir a nuestro repositorio (para nuestro ejemplo utilizaremos el proyecto EntornosDebugger ya utilizado en otras ocasiones)

- Una vez copiada la dirección de la URL de nuestro repositorio y antes de hacer nada haremos un Import hacia el repositorio (que en la actualidad no tiene nada). Entre otras formas esto se puede hacer, pulsando botón derecho sobre el proyecto y elegimos la opción Versioning → Import into Subversion Repository

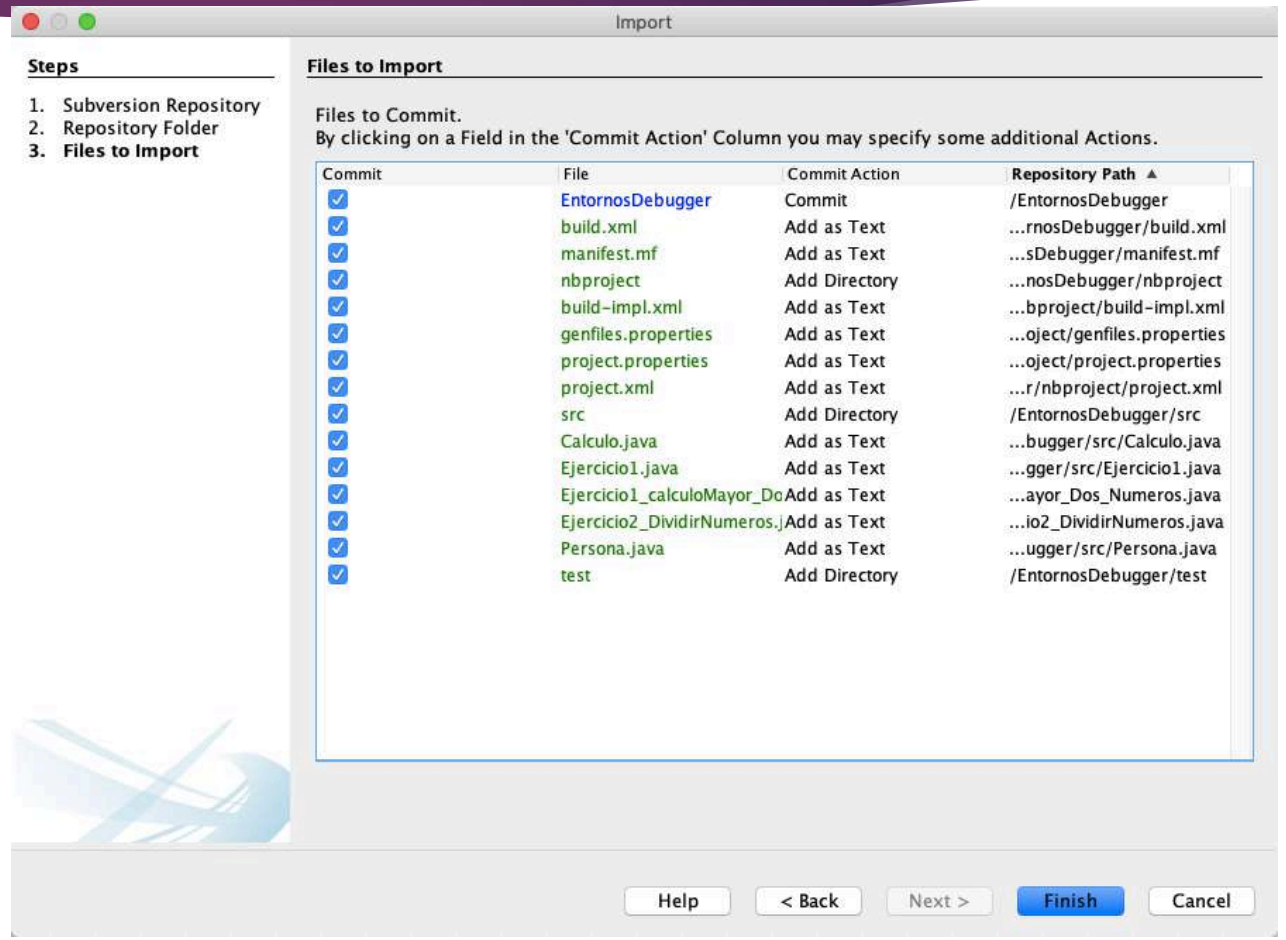
- Deberemos indicar los datos de Subversión



- Indicaremos el proyecto y tendremos que indicarle una carpeta (podemos seleccionar EntornosDebugger) y especificar un mensaje (yo he puesto Subida Inicial, aunque deberíamos ser más específicos).

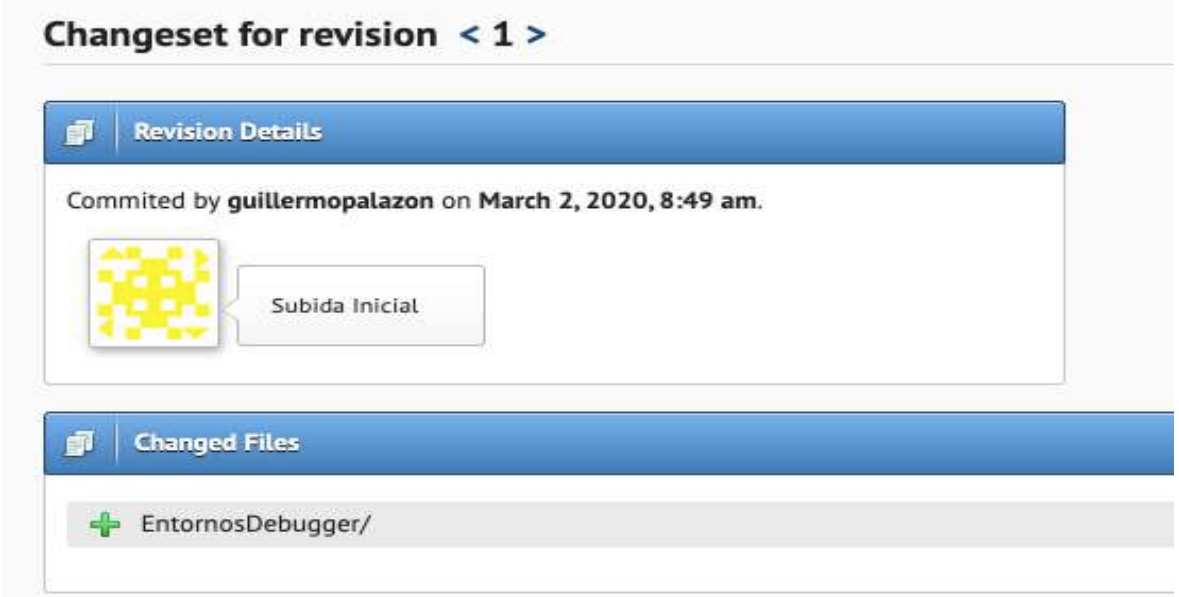


- Le indicaremos todos los ficheros que deseamos que incorpore a Subversión. Por defecto vienen marcados todos y así lo dejaremos.
- Podemos ver que tenemos el directorio (EntornosDebugger) y el resto de ficheros

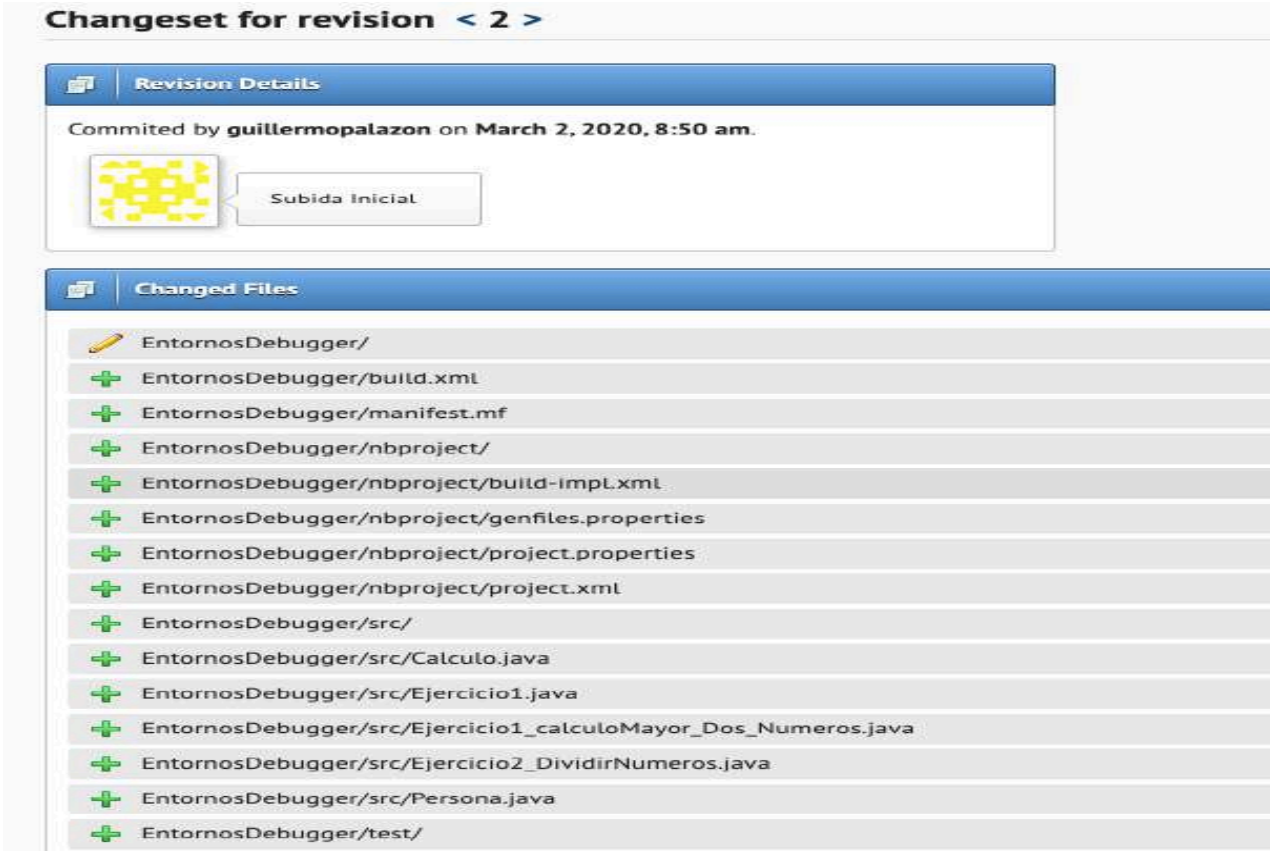


- Si actualizamos ahora nuestro repositorio subversión podemos ver que tenemos ya dos actividades en el mismo.

- En el primero ha creado la carpeta



- En el segundo podemos ver las subidas de los diferentes ficheros

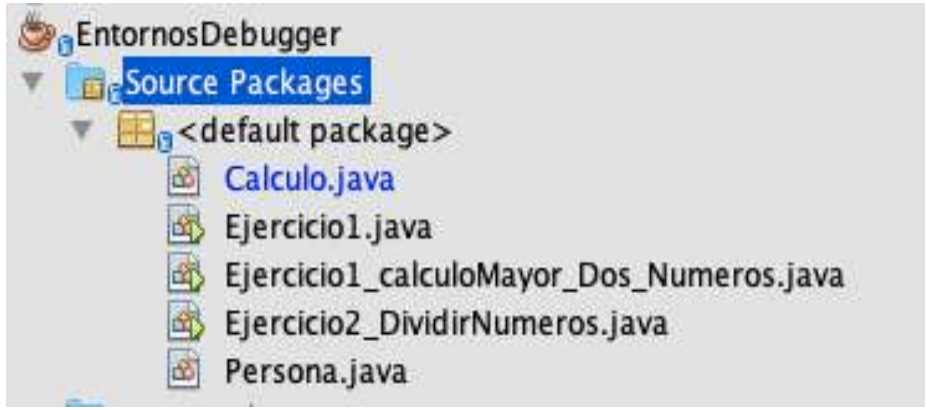


- Si volvemos con el botón derecho sobre el proyecto y pulsamos sobre Subversión -> Mostrar Cambios (Show Changes), podemos revisar que no se ha encontrado ningún cambio entre el proyecto que tenemos ahora mismo en local y el que está en el repositorio. Lo tenemos actualizado respecto al repositorio

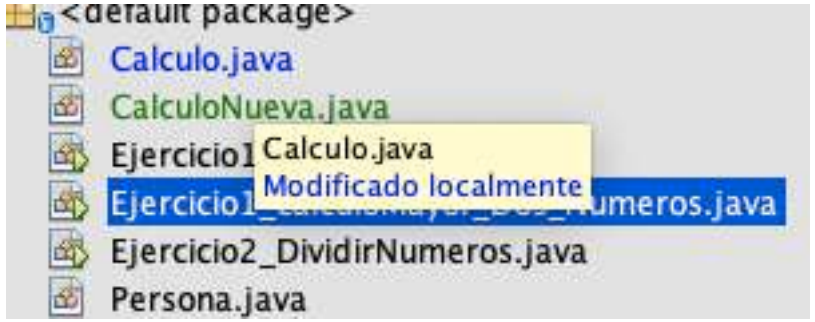
- También si tratamos hacer un commit (Confirmar) de nuestro proyecto, Subversión, podemos ver que Subversión no nos muestra nada que podamos confirmar ya que no hemos hecho ningún cambio.

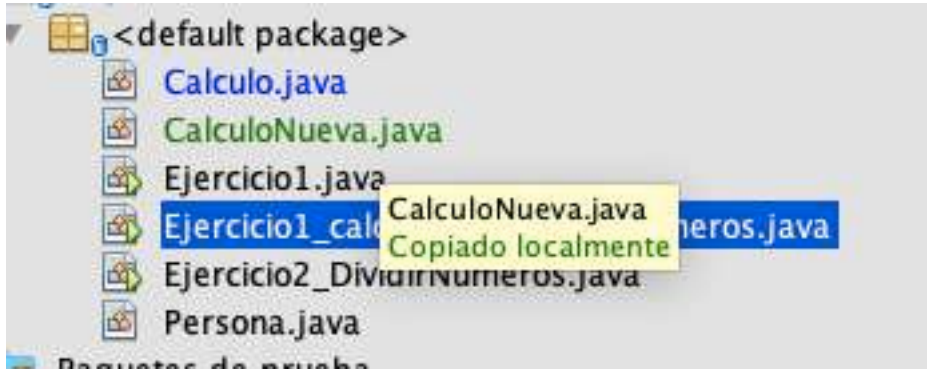
- Probamos a añadir un nuevo cambio en nuestro proyecto. En este caso vamos a cambiar la clase Calculo.java añadiendo un nuevo método que muestre un mensaje por pantalla.
- Netbeans directamente nos va a mostrar en verde los cambios nuevos que hemos añadido.
- En la estructura del proyecto, podemos ver también un icono azul avisándonos de que tiene cambios el proyecto

```
public void mensaje(){
    System.out.println("cambio para subversión");
}
```

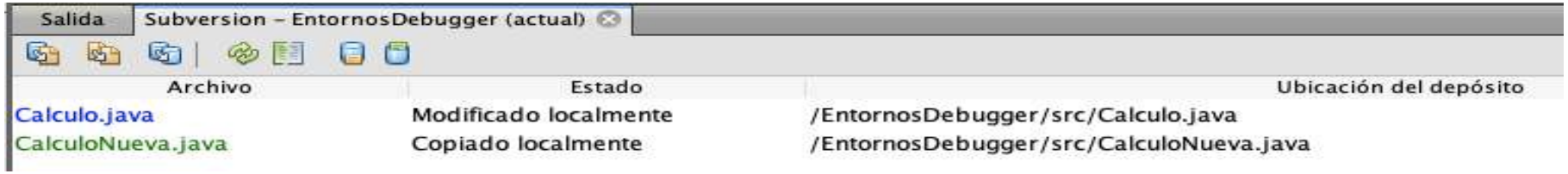


- También vamos a crear una nueva clase que va a ser copia de Calculo y que se va a llamar CalculoNueva.
- Podemos ver que la clase modificada (Calculo) nos aparece ahora en color azul de que esta modificada localmente y la nueva clase en Verde ya que es una nueva clase local

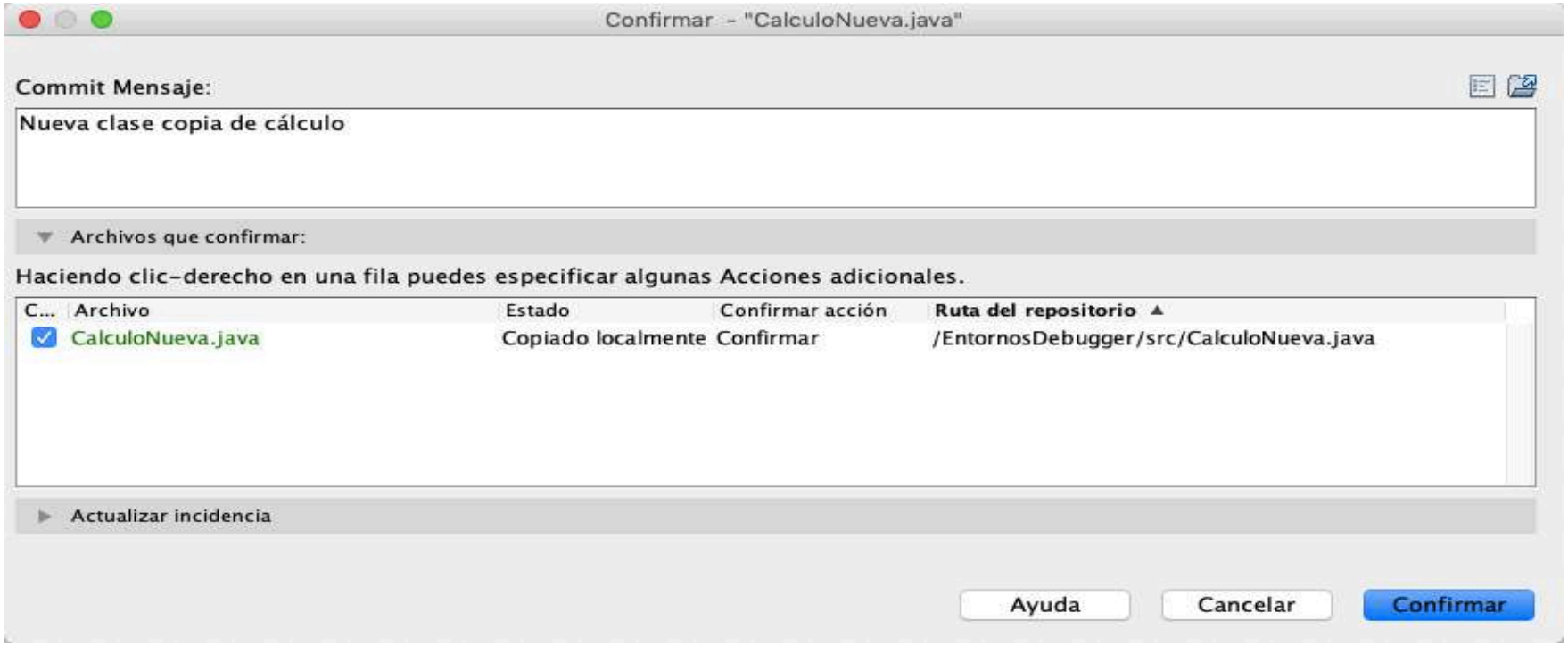




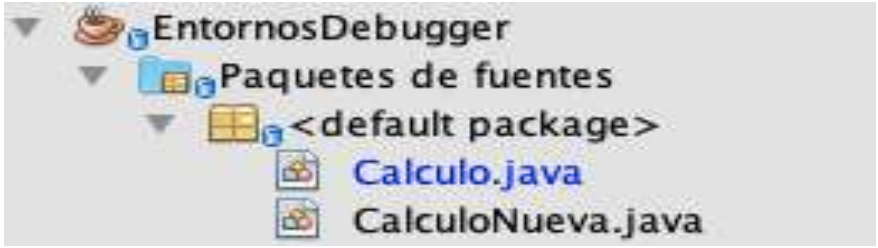
- Si comprobamos ahora los cambios que se han realizado nos aparece ahora los cambios que hemos ido realizando.



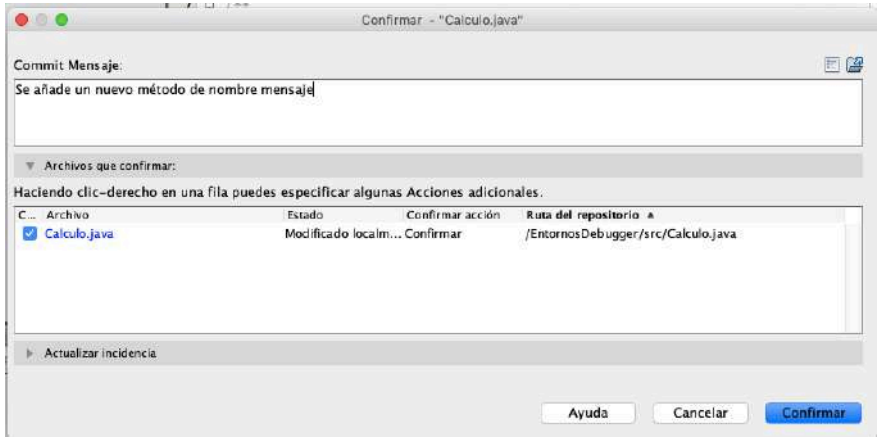
- Podemos ahora hacer un commit y confirmar estos cambios, pero es posible que no queramos hacer commit de todo porque por ejemplo no hemos aún hecho todos los cambios en Calculo y solo queremos subir la clase nueva (CalculoNueva). Podemos pulsar sobre el botón derecho seleccionando la clase y no sobre el proyecto.


Confirmar - "CalculoNueva.java"
Commit Mensaje:
Nueva clase copia de cálculo
Archivos que confirmar:
Haciendo clic-derecho en una fila puedes especificar algunas Acciones adicionales.
C... Archivo Estado Confirmar acción Ruta del repositorio
CalculoNueva.java Copiado localmente Confirmar /EntornosDebugger/src/CalculoNueva.java
Actualizar incidencia
Ayuda
Cancelar
Confirmar

- Si comprobamos ahora la estructura del proyecto, vemos que CalculoNueva ya se encuentra en color “normal” y no en estado verde

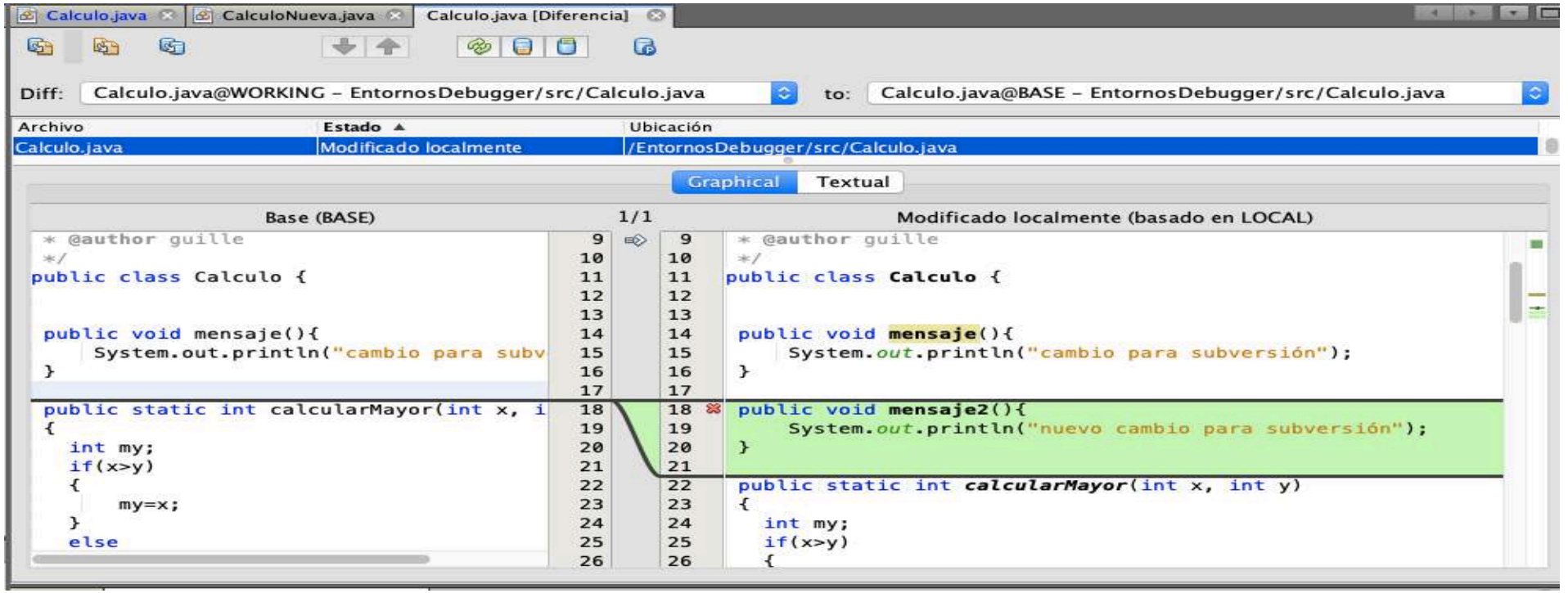


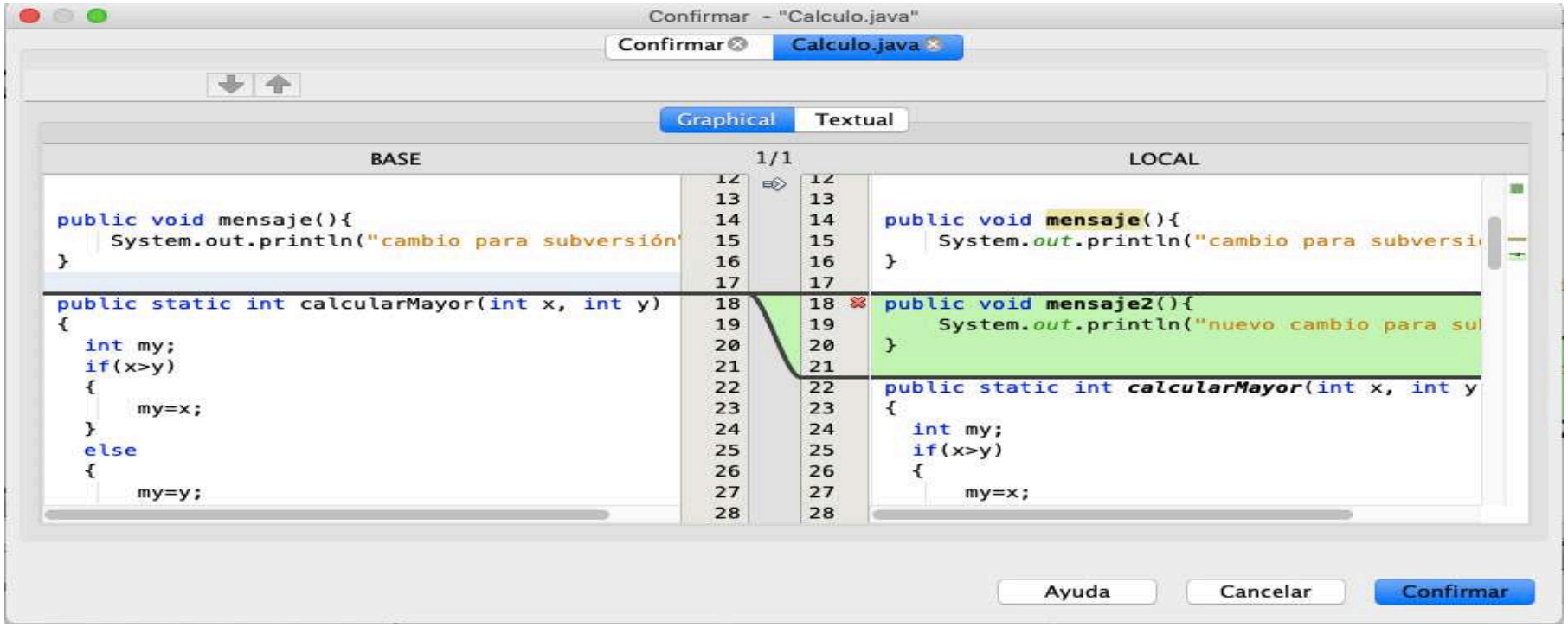
- A continuación hacemos también commit (confirmar) con la clase que hemos modificado (Calculo). Podemos poner como mensaje que se ha añadido un nuevo método que hemos llamado mensaje.



- Si actualizamos ahora nuestro repositorio podemos comprobar que tenemos ya 4 versiones del proyecto:
  - 1. Carpeta del proyecto
  - 2. Subida Inicial
  - 4. Nueva clase copia de cálculo
  - 5. Se añade un nuevo método a cálculo
- Podemos pulsar en cualquier versión para ver los cambios realizados en la misma y tenemos más información sobre ella

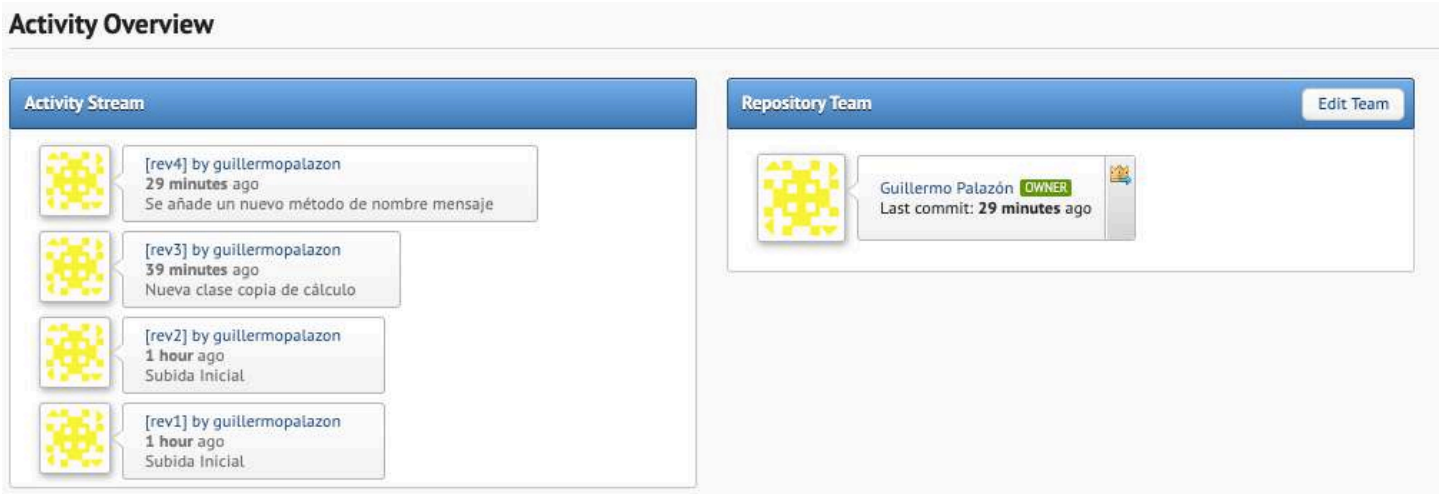
- Vamos a añadir otro nuevo método a nuestra clase Calculo, en este caso lo vamos a llamar mensaje2 y también va a mostrar por pantalla un texto.
- Imagínate que ha pasado un tiempo o que no te acuerdas de los cambios que has hecho. Para ello cuando le das a mostrar cambios (siguiente diapositiva) o a confirmar (posterior a la siguiente diapositiva) puedes pulsar sobre el nombre del archivo en la ventana de subversión y te indica los cambios de la copia local versus repositorio.


Calculo.java
CalculoNueva.java
Calculo.java [Diferencia]
Diff: Calculo.java@WORKING – EntornosDebugger/src/Calculo.java to: Calculo.java@BASE – EntornosDebugger/src/Calculo.java
Archivo Estado Ubicación
Calculo.java Modificado localmente /EntornosDebugger/src/Calculo.java
Graphical Textual
Base (BASE) 1/1 Modificado localmente (basado en LOCAL)
* @author guille
*/
public class Calculo {
public void mensaje(){
System.out.println("cambio para subv
}
public static int calcularMayor(int x, i
{
int my;
if(x>y)
{
my=x;
}
else
* @author guille
*/
public class Calculo {
public void mensaje(){
System.out.println("cambio para subversión");
}
public void mensaje2(){
System.out.println("nuevo cambio para subversión");
}
public static int calcularMayor(int x, int y)
{
int my;
if(x>y)
{

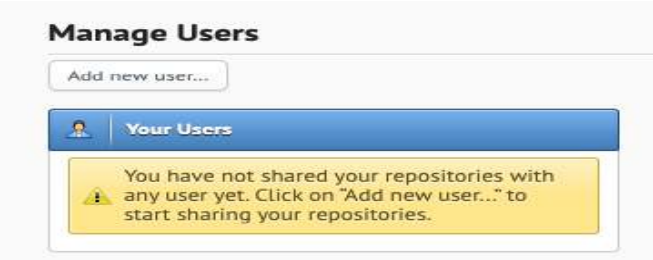

Confirmar - "Calculo.java"
Confirmar
Calculo.java
Graphical
Textual
BASE
1/1
LOCAL
public void mensaje(){
System.out.println("cambio para subversión'
}
public static int calcularMayor(int x, int y)
{
int my;
if(x>y)
{
my=x;
}
else
{
my=y;
public void mensaje(){
System.out.println("cambio para subversi
}
public void mensaje2(){
System.out.println("nuevo cambio para su
}
public static int calcularMayor(int x, int y
{
int my;
if(x>y)
{
my=x;
Ayuda
Cancelar
Confirmar

- Pero claro, esto que estamos describiendo con un solo usuario tiene utilidad por tener un respaldo de los cambios realizados. No obstante, un control de versiones tiene más potencia o es más interesante su uso cuando tenemos varios usuarios trabajando (o pueden trabajar) sobre el mismo proyecto.
- Para poder añadir nuevos usuarios pulsamos en el menú superior en Dashboard y a continuación en Users. Estos usuarios deben haberse registrados, es decir, ser usuarios de https://riouxsvn.com/
- En mi caso dispongo ya de otra cuenta por lo que para añadir ese usuario solo tendría que indicar su nick.

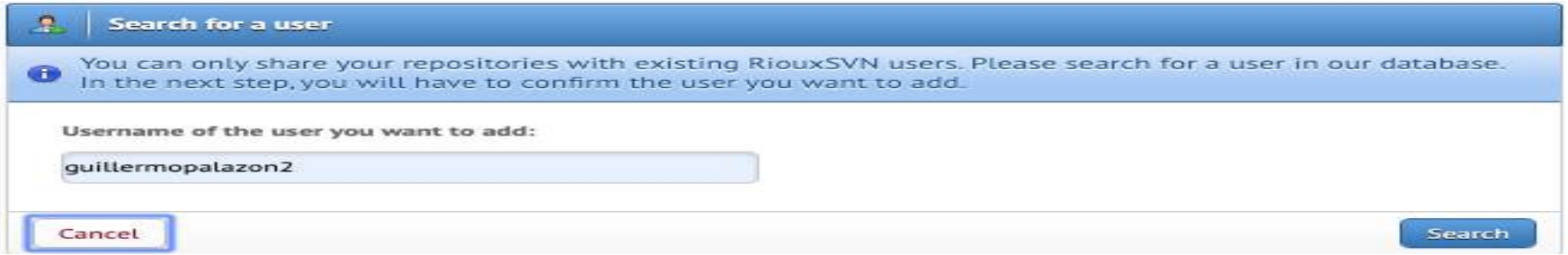
- Existen muchas maneras de añadir al usuario a tu proyecto. En nuestro vamos a añadir al usuario guillermopalazon2.
- Para ello editamos el equipo de nuestro proyecto.



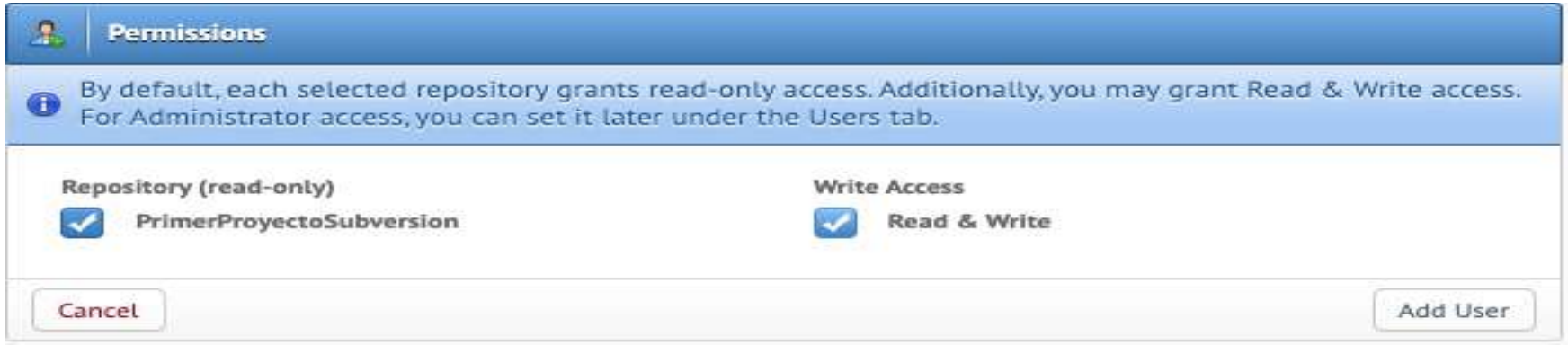
- Añadiremos un nuevo usuario



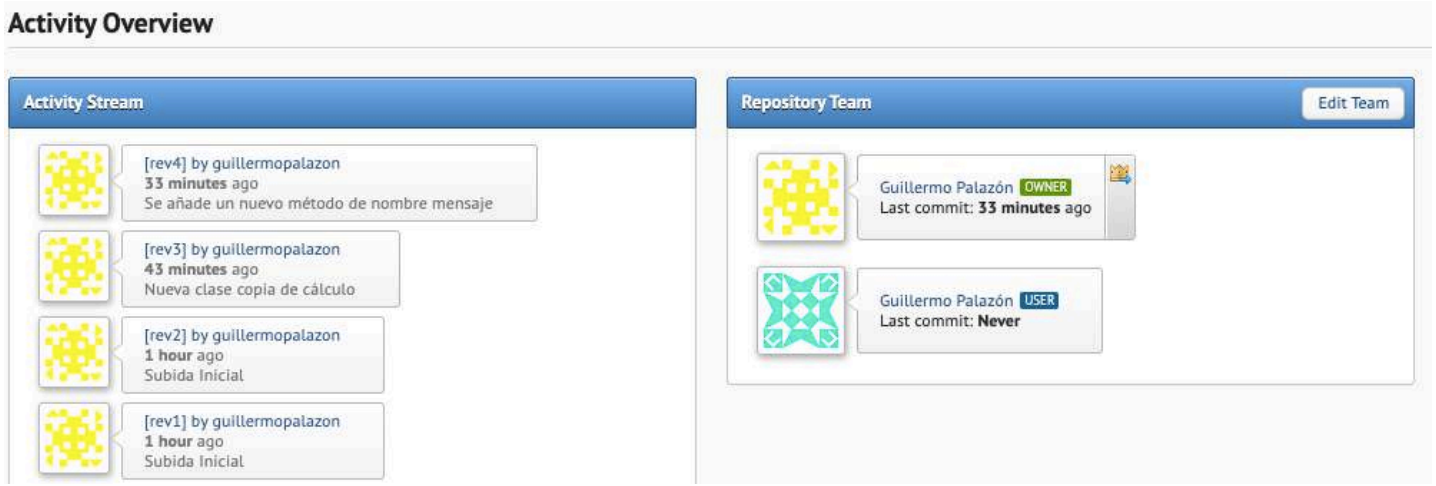
- Indicaremos el nick del usuario que queremos añadir



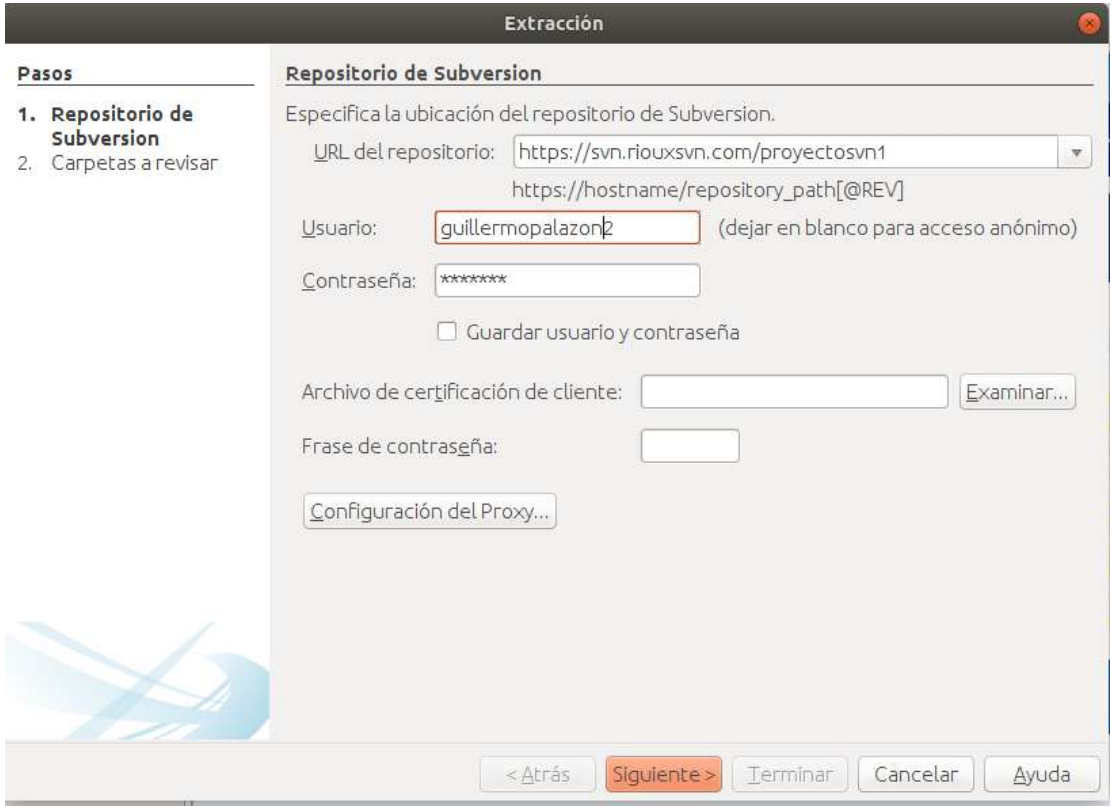
- Una vez encontrado lo único que debemos hacer es seleccionarlo. Una vez hecho le indicamos para qué proyectos (en nuestro solamente tenemos uno) y qué permisos le otorgamos (se los damos todos para que pueda trabajar plenamente en el proyecto). Pulsamos Add User



- En nuestro proyecto ya se verían los dos usuarios (con el mismo nombre por las circunstancias ya indicadas, pero lo usual es que fueran usuarios de procedencia distinta).



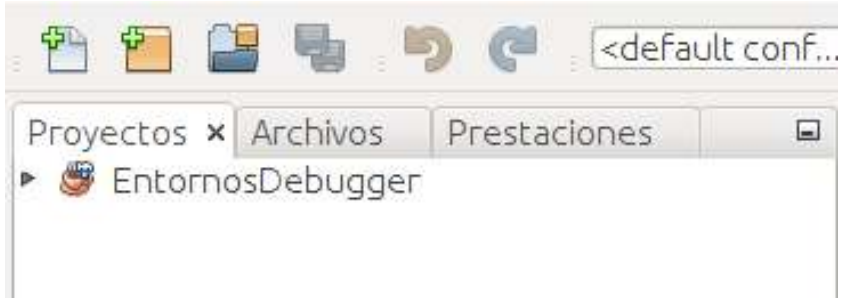
- Desde otra máquina y teniendo la dirección del repositorio, nos vamos a conectar al mismo con el nuevo usuario para descargar dicho proyecto y poder trabajar con él.
- Para ello realizaré un checkout del mismo.



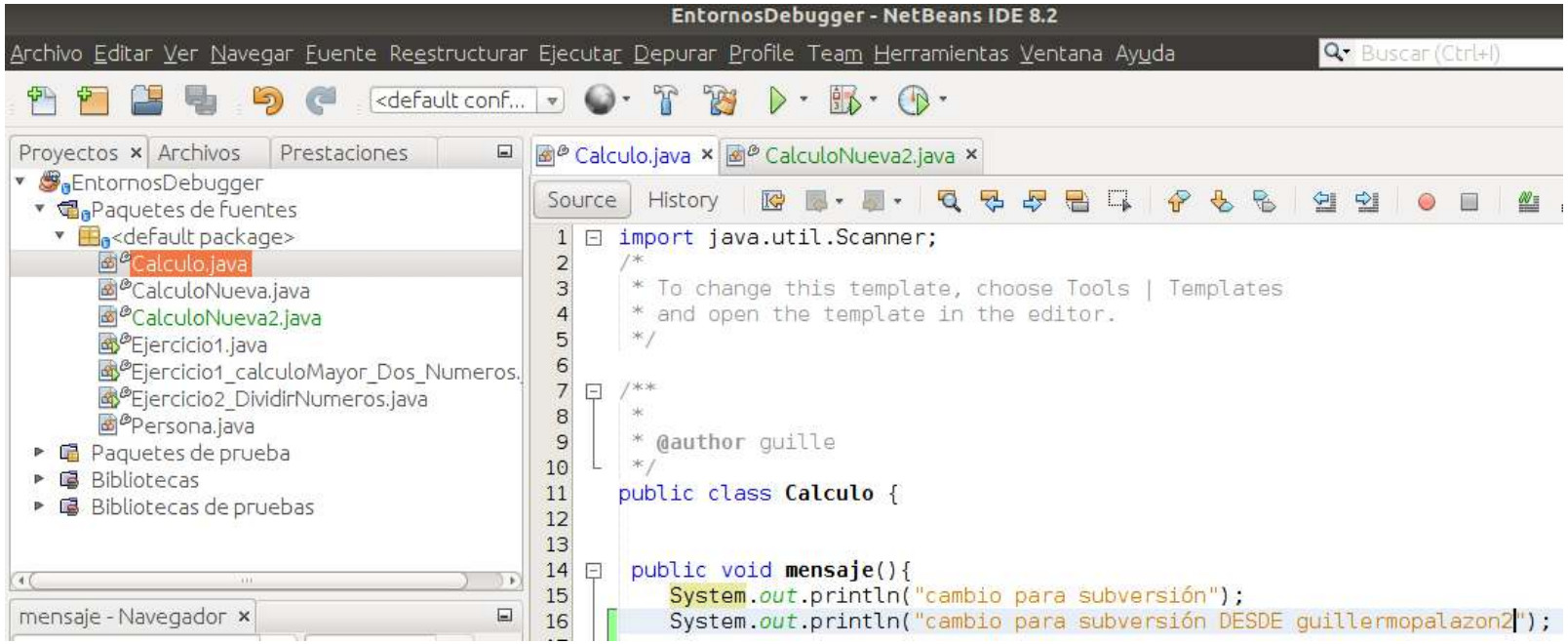
- Pulsaremos en especificar para indicar el proyecto que queremos.


Examinar carpetas del repositorio
Elija las carpetas que desea revisar del repositorio
Nodos
Revisión
Fecha
Autor
Historial
https://svn.riouxsvn.com/proyectosvn1
START
EntornosDebugger
4
25-abr-2019 12:50:17
guillerm...
Aceptar
Cancelar
Ayuda

- Tras seleccionar el proyecto, dejo el resto de parámetros tal y como están para descargar la revisión del proyecto (versión) que se encuentra en la cabecera (HEAD). Podría haber señalado alguna más antigua, pero se deja así para descargar la versión más actual y pulsamos en Terminar.
- Me indico que se ha incorporado bien el proyecto y si deseo abrir el mismo, le indico que sí y ya puedo ver dicho proyecto en el menú de proyectos correspondiente.



- Desde el segundo usuario hemos añadido cambios en Calculo y hemos añadido una nueva clase CalculoNueva2



- Si actualizo el repositorio podemos ver que no ha habido cambios en las versiones que tenemos allí. Esto es debido a que el segundo usuario no ha realizado el commit correspondiente.

- El segundo usuario realiza el commit de todos los cambios, por lo que subo una nueva revisión que tiene los cambios en Calculo y la nueva clase CalculoNueva2.

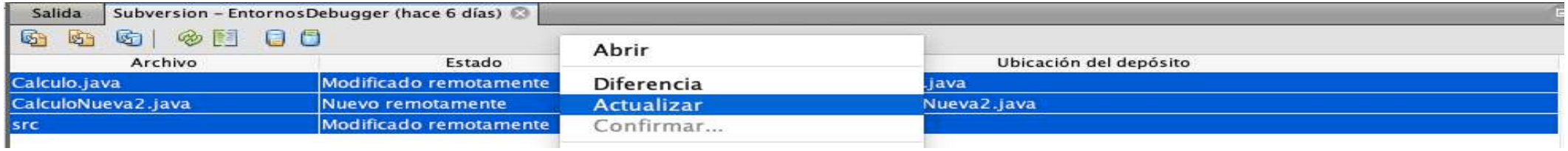
- Al haber mezclado diferentes pruebas **NO VAN A COINCIDIR LOS NÚMEROS DE REVISIONES.**


Activity Stream
[rev5] by guillermopalazon2
1 minute ago
Cambios realizados en el segundo usuario
[rev4] by guillermopalazon
1 hour ago
Se añade un nuevo método de nombre mensaje
[rev3] by guillermopalazon
3 hours ago
Se añade un nuevo método de nombre mensaje
[rev2] by guillermopalazon
3 hours ago
Nueva clase copia de cálculo
[rev1] by guillermopalazon
1 day ago
Subida Inicial

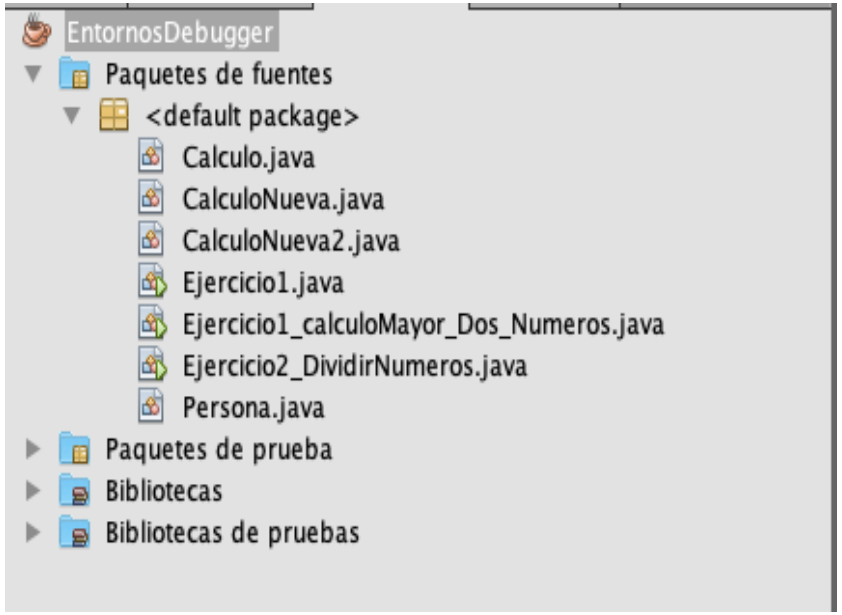
- Si le doy ahora desde mi proyecto a Mostrar Cambios estos cambios que se han realizado.

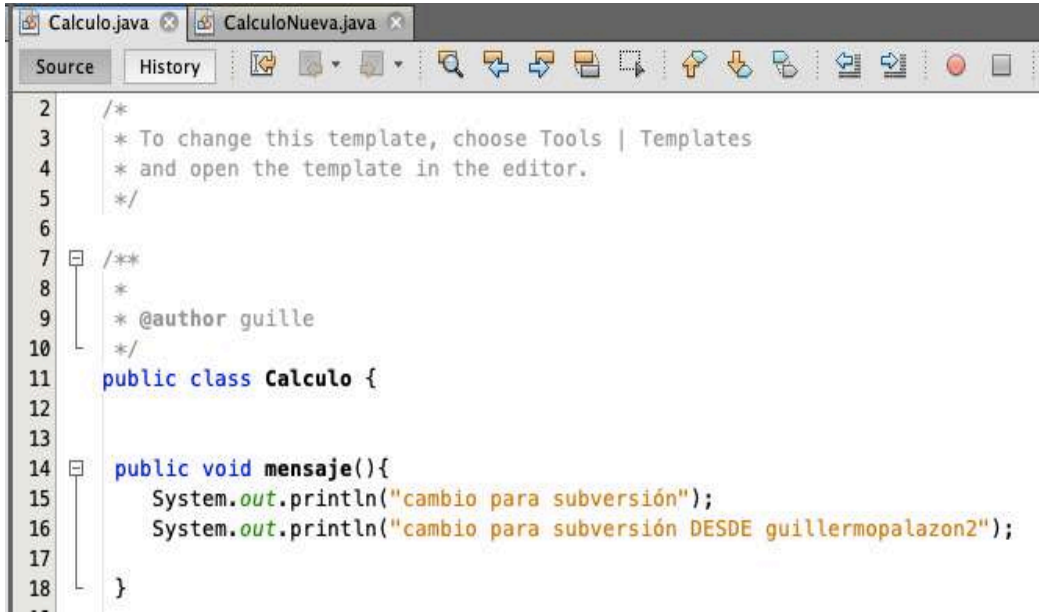
- **OJO: En ocasiones no es tan inmediato que se actualice en el subversión los cambios del repositorio**.
- Se puede obligar a que haga la búsqueda de cambios pulsando en el botón "mostrar archivos modificados remotamente" o "Mostrar archivos modificados local y remotamente" (tercer y primer botón respectivamente de la captura de pantalla anterior)

- Para que estos cambios que ha provocado el segundo usuario se confirmen en el primer usuario será necesario hacer un update de estas clases que nos han aparecido.
- Para ello puedo pulsar de manera individual en cada clase o elegir todas las que quiero actualizar (update) y pulsar botón derecho del ratón y dar sobre actualizar (aparece también una carpeta src ya que el otro usuario ha organizado las clases con una nueva estructura, es posible que no aparezca).
- En mi caso lo señalo todo y actualizo



- Una vez pulsado en Actualizar, estos cambios se reflejan automáticamente en nuestro proyecto.



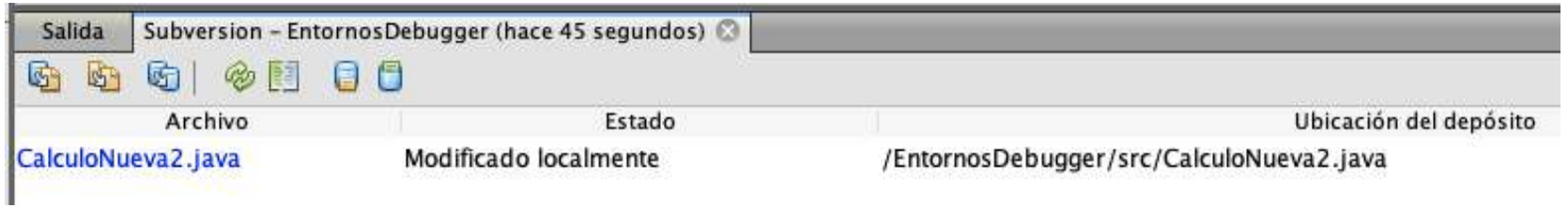


- En la nueva clase CalculoNueva2, voy a añadir desde el primer usuario un método al principio de la clase (lo llamaré nuevometodo1)
- Desde el segundo usuario un método al final de la misma clase a la misma vez (lo llamaré nuevometodo2).
- El código de ambos métodos puede ser tan sencillo como el siguiente:

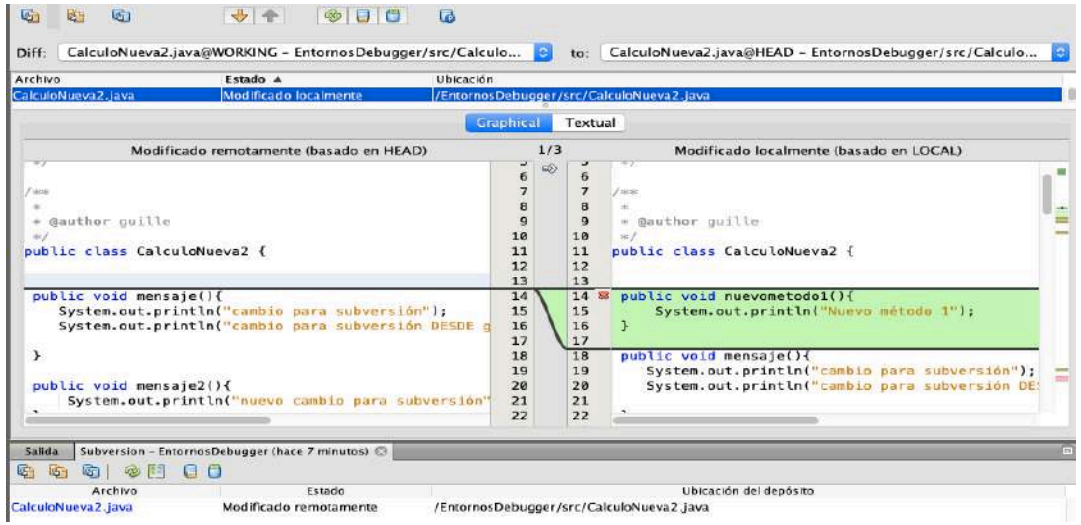
```
public void nuevometodo1(){
    System.out.println("Nuevo método 1");
}
```

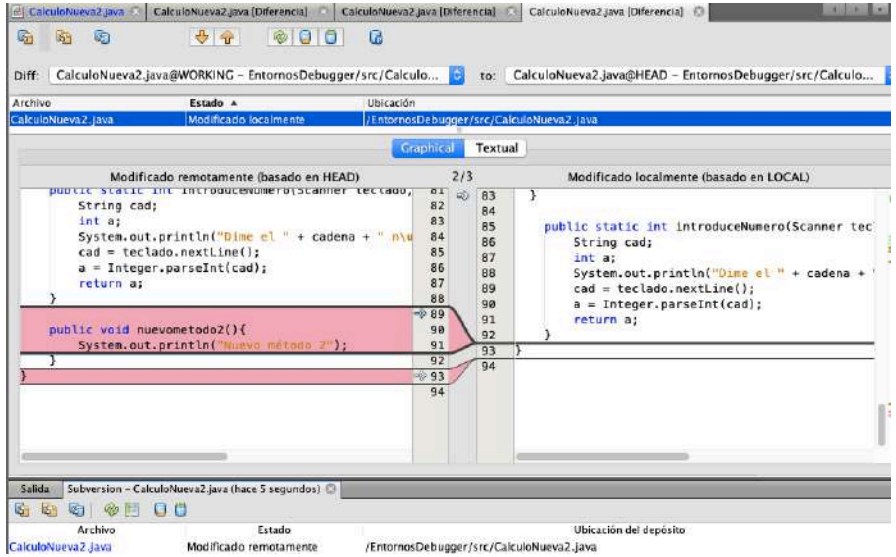
- Está claro que si desde el primer usuario (o segundo usuario) pido mostrar cambios con respecto a subversión, los únicos cambios es que tengo un archivo modificado localmente que no se ha subido al Repositorio.
- Hago un commit (confirmar) de este fichero desde cualquiera de los dos equipos. En este caso he elegido hacerlo primero desde el segundo usuario.

- Volvemos ahora a hacer desde el primer usuario un mostrar cambios (o refrescamos los datos que se habían mostrado antes).
- Podemos ver que ahora el mismo fichero nos aparece tanto en los modificados localmente como remotamente.

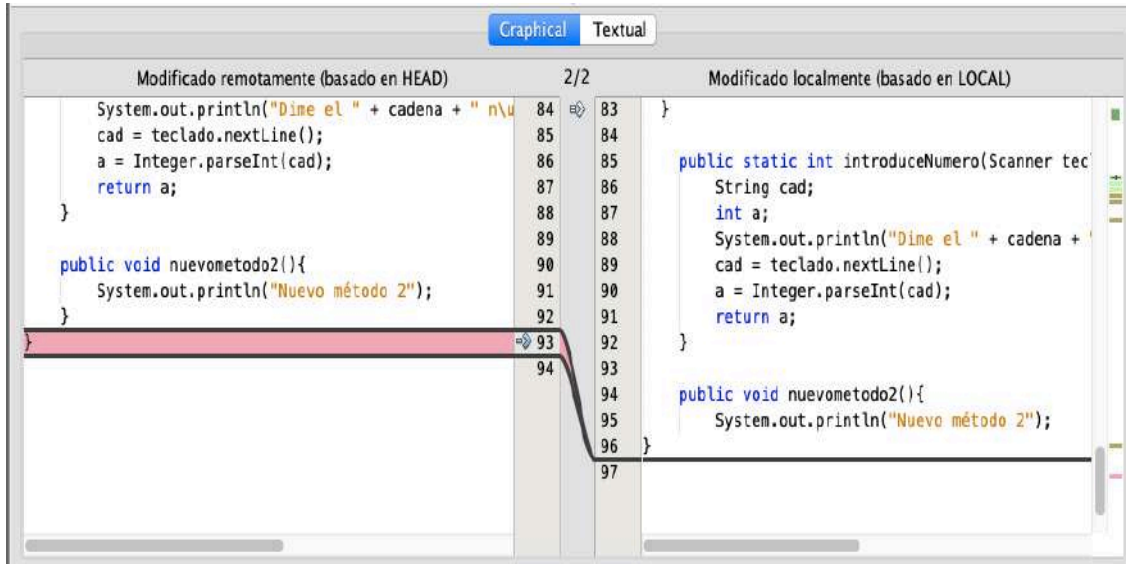


- En estos casos lo suyo es ver primero las diferencias que tengo con el que estamos en remoto.
- Sobre el nombre de la clase, botón derecho y mostrar cambios.





- Trabajo con mi copia local para insertar en la misma aquellas cosas que están en remoto y que no van a perjudicar mi copia local.

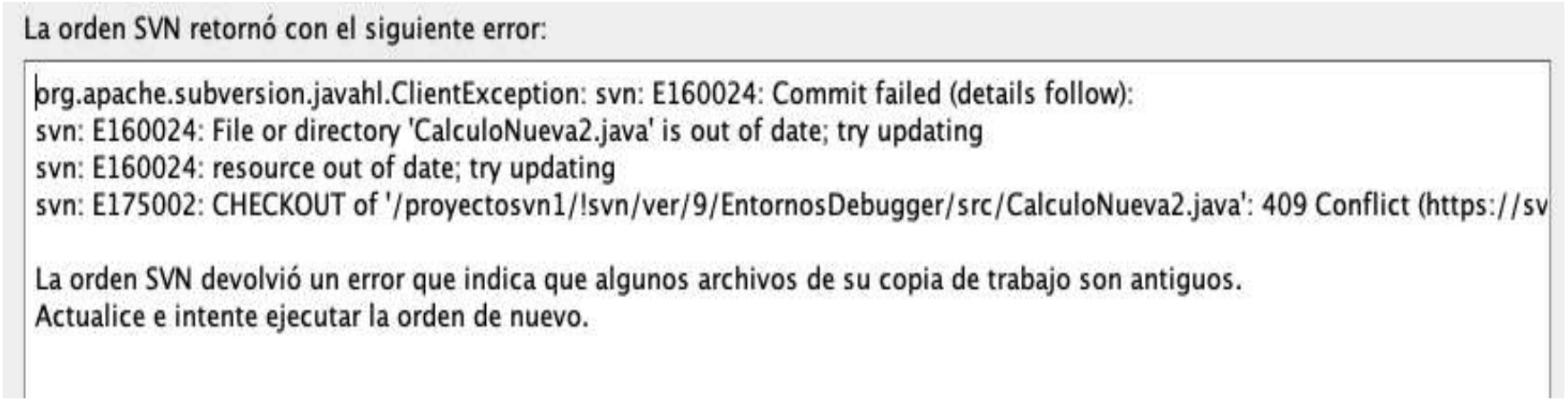


- Una vez que le doy a Insertar lo que desee ya le puedo dar a grabar en ese mismo sitio y si me voy entonces a la pestaña original de la clase se visualizarán los cambios y ya me indicará que no tengo cambios remotos.

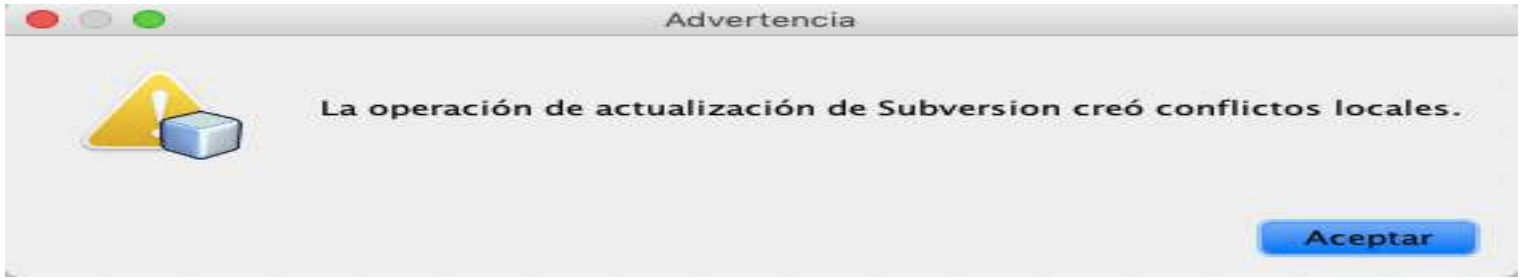
- Ahora debería darle a confirmar (commit) para que los cambios que yo hice antes de mezclar con el de remoto se suban también al repositorio y puedan ser descargados por el resto de usuarios.
- Es posible que si en la clase original NO has grabado, cuando quieras hacer el confirmar te de un problema de subversión ya que estás intentando subir una versión que tiene una fecha y en el repositorio existe una versión de ese fichero con una fecha posterior.
- Podemos comprobar que se ha subido correctamente (Ojo, como ya he comentado alguna revisión está de más por pruebas que se hicieron)

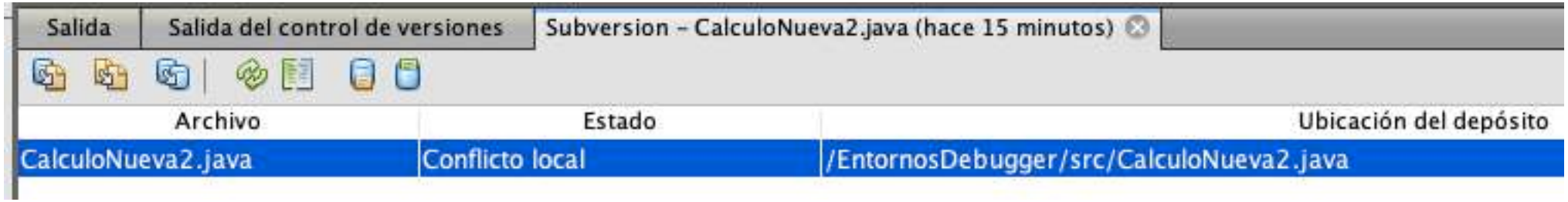
- Ahora desde el segundo usuario modifico el método nuevometodo2 (aún no he actualizado en ese usuario para descargarme la última revisión que ha subido el primer usuario).
- Si tras cambiar el nuevometodo2 quiero hacer un confirmar no me deja porque me dice que hay una copia que no he actualizado. Si pulso en actualizar directamente me descarga la copia y la mezcla con la mía ya que el contenido del repositorio NO da ningún conflicto con mi copia local.
- Ahora ya sí puedo hacer desde el segundo usuario la subida de esta nueva revisión en la que se ha modificado nuevometodo2.

- Desde el primer usuario modifico ahora la misma línea que ha modificado antes el segundo usuario y cuando trato de hacer un confirmar me da el mismo error de conflicto de fecha del fichero.

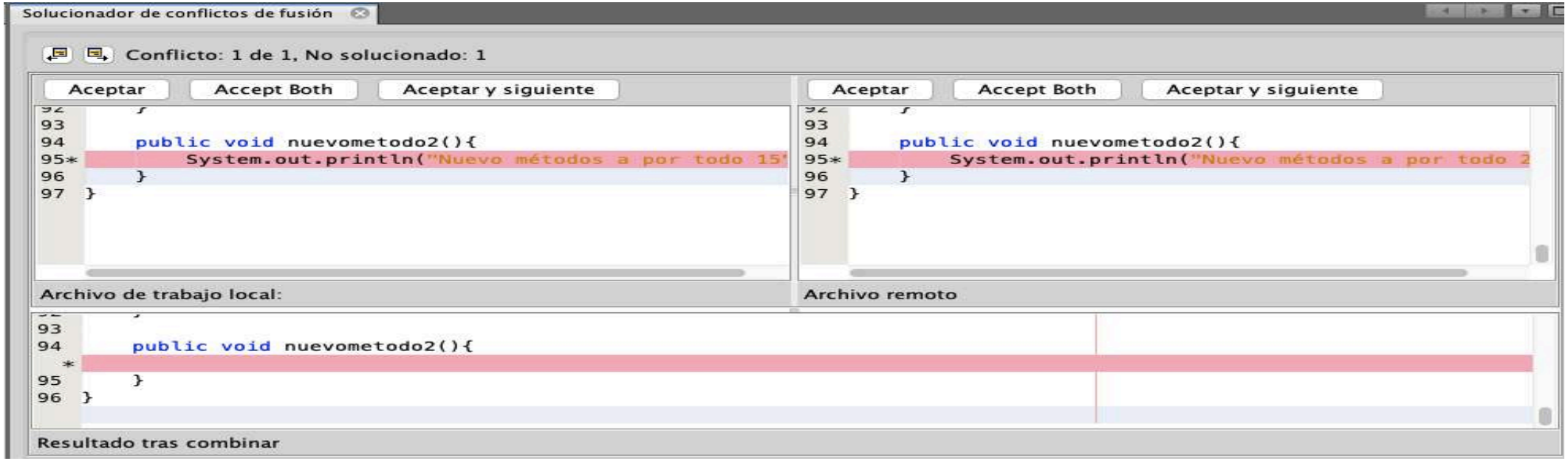


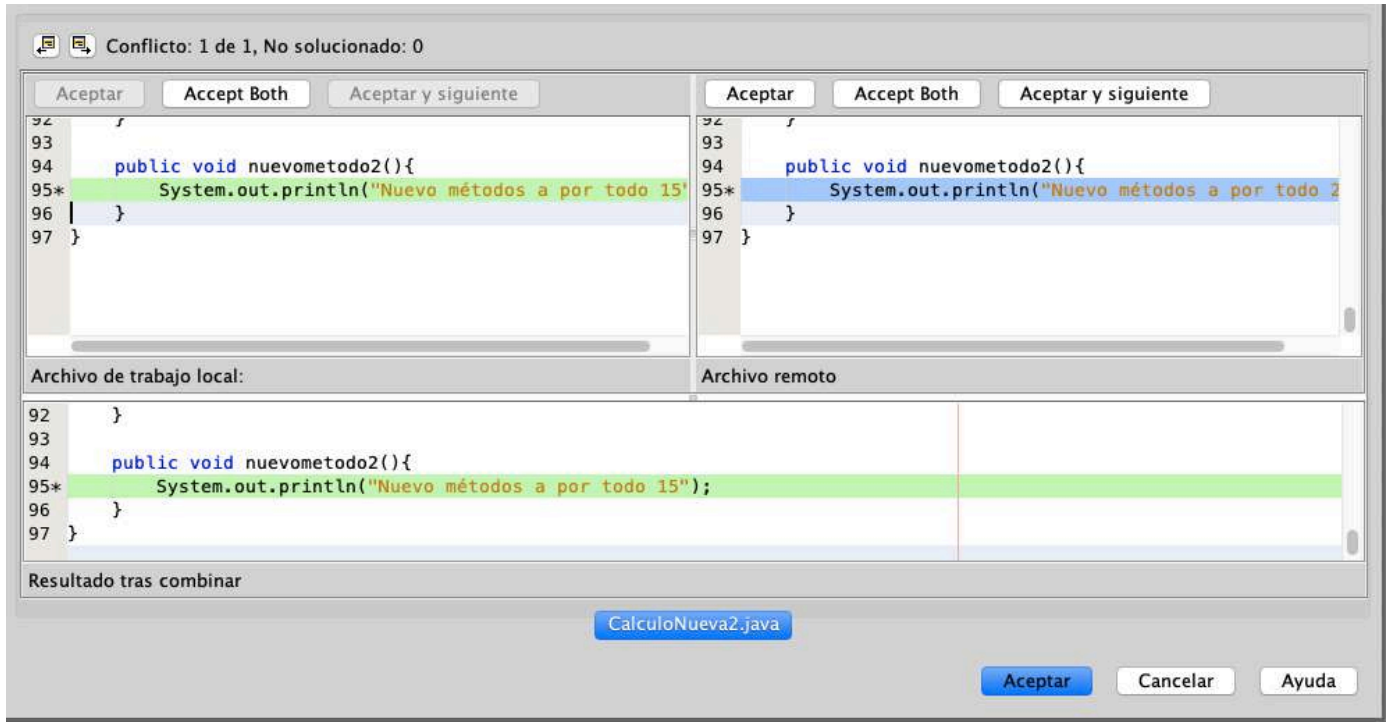
- Si trato ahora de actualizar en el primer usuario no me va a hacer la mezcla porque Subversión detecta que la versión del repositorio entra en conflicto con mi versión local.

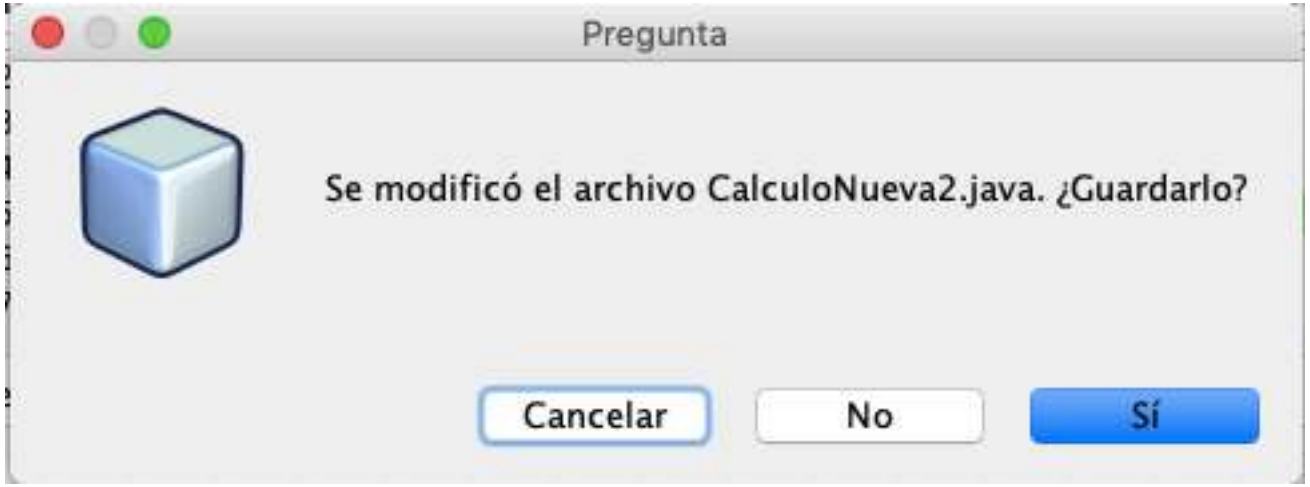




- En este caso, soy yo el que debo solucionar este problema de manera manual. Solo tenemos un conflicto como se puede ver en la imagen y debemos aceptar uno de los dos que tenemos en la parte superior (o directamente escribir en la parte inferior). En este caso yo he elegido la opción de la izquierda




Conflicto: 1 de 1, No solucionado: 0
Aceptar
Accept Both
Aceptar y siguiente
Archivo de trabajo local:
Archivo remoto
Resultado tras combinar
CalculoNueva2.java
Aceptar
Cancelar
Ayuda


Pregunta
Se modificó el archivo CalculoNueva2.java. ¿Guardarlo?
Cancelar
No
Sí

- Ahora ya podemos confirmar los cambios que hemos realizado.