

TEMA 4.2 OPTIMIZACIÓN Y DOCUMENTACIÓN: *Servidor subversión, VisualSVN Server*

1.- Introducción

VisualSVN Server es un servidor de subversión que se instala como un servidor web e incluye un servidor Apache.

La web de descarga es: <https://www.visualsvn.com/server/>

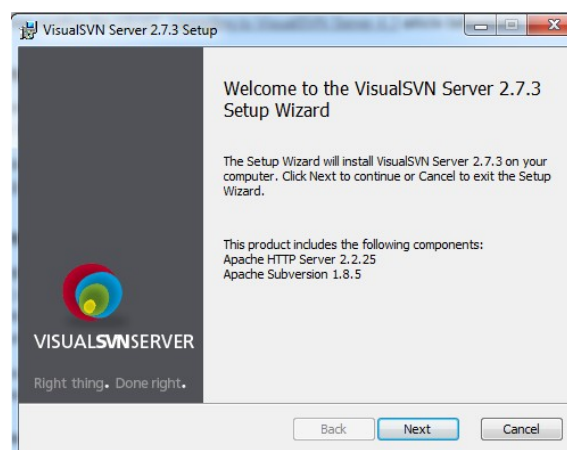
Histórico de versiones

<https://www.visualsvn.com/server/changes/>

Version 2.7.3 (November 27,2013) [[Download](#)]

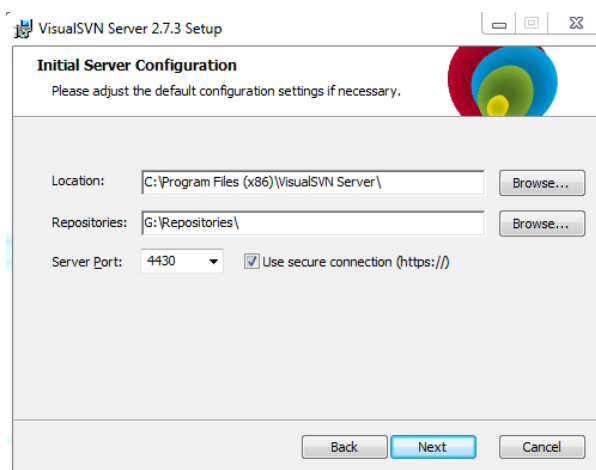
- **Updated to Apache Subversion 1.8.5** with fixes for the following vulnerabilities: [CVE-2013-4505](#), [CVE-2013-4558](#).
For further details please see <http://svn.apache.org/repos/asf/subversion/tags/1.8.5/CHANGES>
- Fixed: service may crash during startup if service account does not have appropriate access rights.
- Several performance improvements in the management console.

Esta aplicación proporciona una consola de administración de repositorios muy útil llamada Management Console, que permite administrar el servidor de forma muy simple.

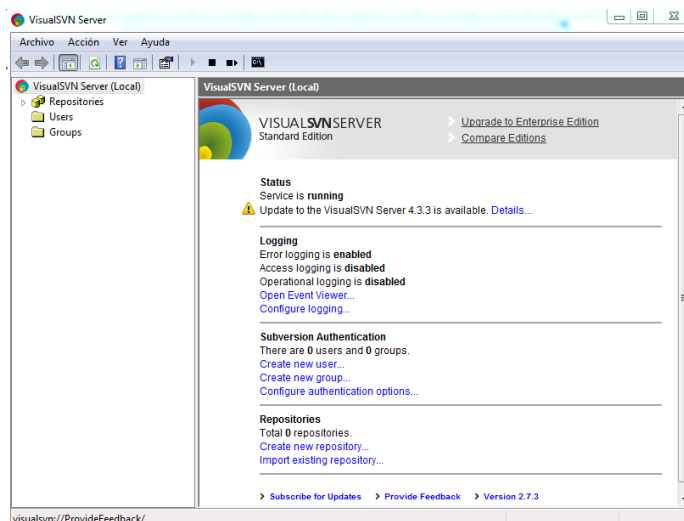


Para su instalación lanzamos el ejecutable, aceptamos los términos de la licencia , seleccionamos la opción VisualSVN Server and Management Console y la Standard Edition.

En la configuración del Server indicamos lo siguiente: en Location, la carpeta de instalación de la aplicación , en Repositories se indica donde se almacenarán los repositorios, y en Server Port, el puerto que utilizará para conexiones seguras, por defecto aparece el 443, pero para evitar conflictos en el caso de que tengamos instalado Apache lo cambiaremos al puerto 4430 u otro puerto distinto que no tengamos ocupado y pulsamos continuar con la instalación.

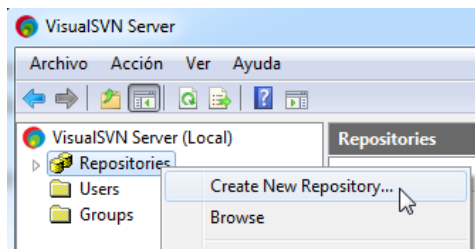


Cuando termina la instalación se abrirá la ventana de administración donde se podrá crear usuarios y repositorios.

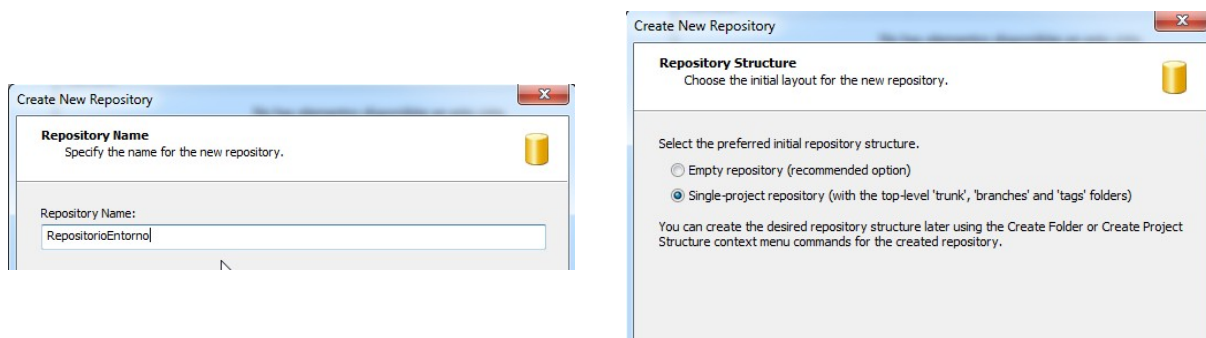


2.- Creación de repositorios.

Sobre la consola de administración seleccionamos **Repositories** y se selecciona **Create new Repository**.



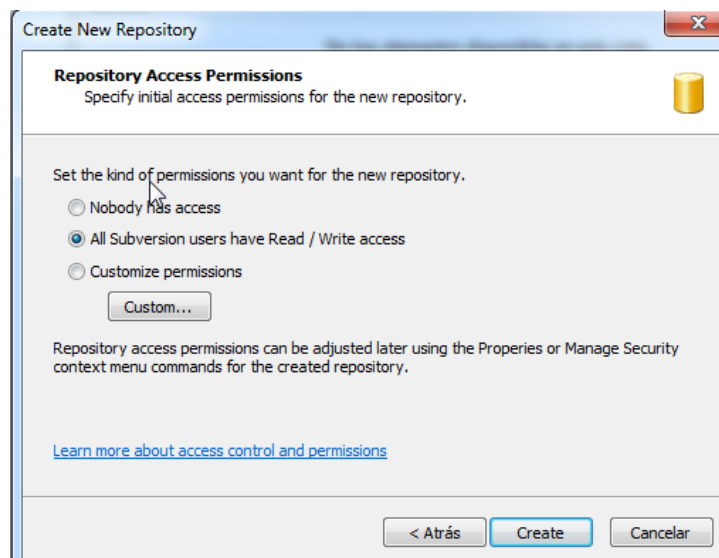
Nos solicitará el nombre del repositorio y nos solicitará crear un repositorio vacío o con la estructura trunks-branches-tags . Seleccionamos crear un repositorio con la estructura trunks-branches-tags



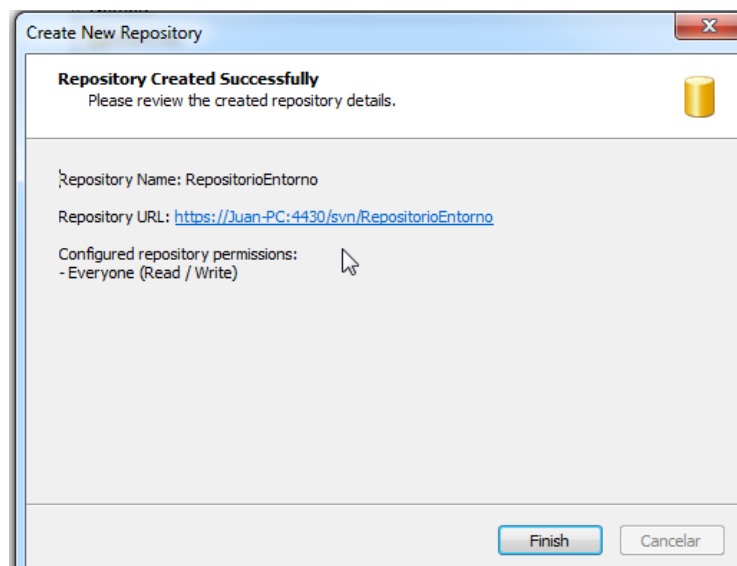
Seguidamente nos solicita el permiso para los usuarios, que podemos elegir tres opciones:

- Nadie tiene acceso
- Todos los usuarios tienen acceso de lectura/escritura
- Custom, donde podemos configurar el acceso a unos cuantos usuarios.

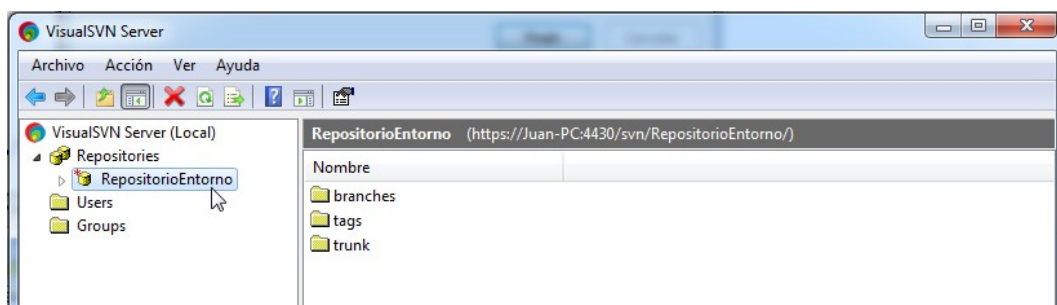
En un principio configuraremos la aplicación con la opción por defecto , que todos tengan acceso.



Al pulsar create nos aparece una ventana que nos muestra la URL que utilizaremos para conectarnos con el repositorio, a través de clientes como TortoiseSVN, o via web desde el navegador.



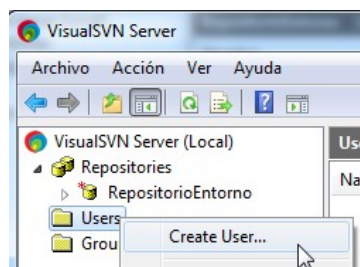
Al pulsar Finish , salimos de la instalación y aparece la consola con el repositorio creado.



2.1.-Creación de usuarios

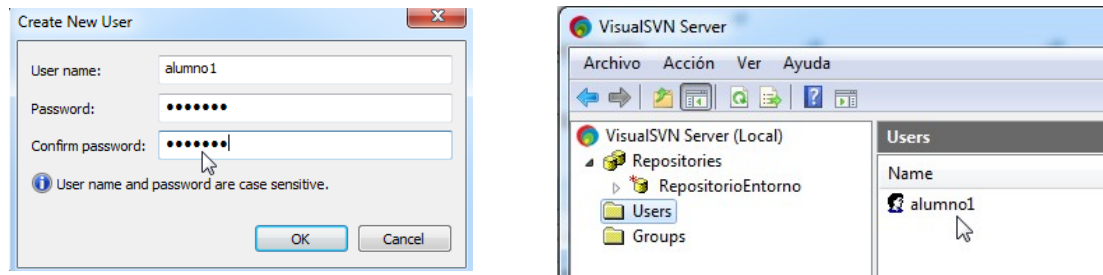
Vamos a crear usuarios para que se conecte al repositorio.

Nos posicionamos en la carpeta “User” → botón derecho del ratón y elegimos “crear usuario”



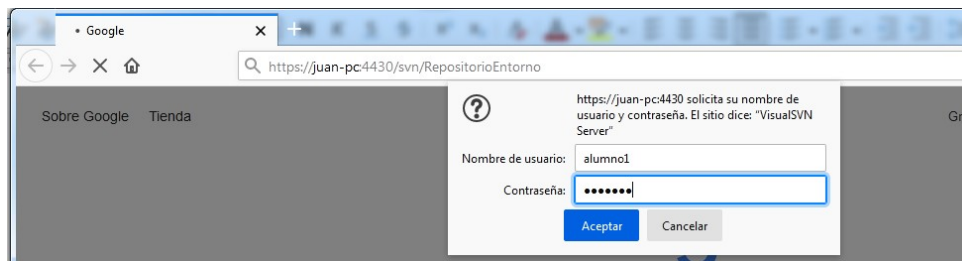
escribimos el usuario y la contraseña.

Ejemplo , crearé un usuario con “alumno1” , “alumno1”

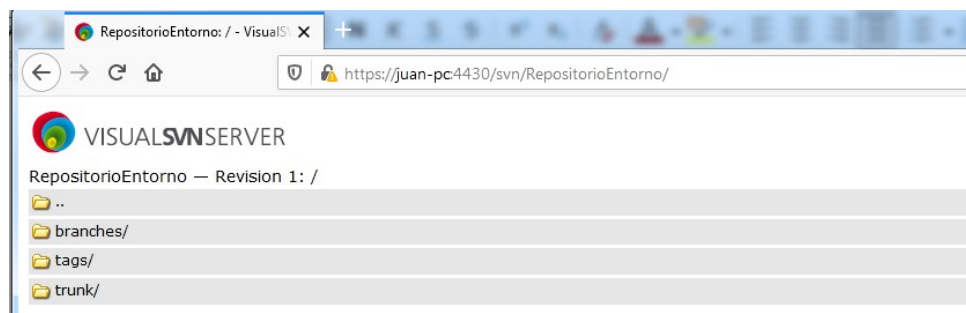


2.2.-Comprobación del funcionamiento del servidor

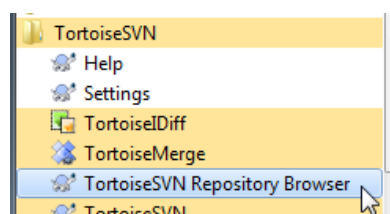
Abrimos una ventana del navegador e introducimos la URL del repositorio creado anteriormente:
<https://Juan-PC:4430/svn/RepositorioEntorno>

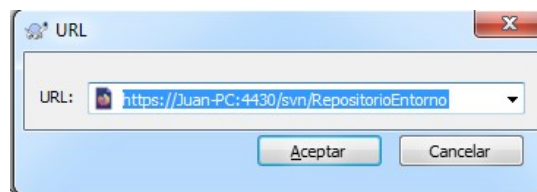


Nos solicitará introducir el usuario y la contraseña para poder entrar.

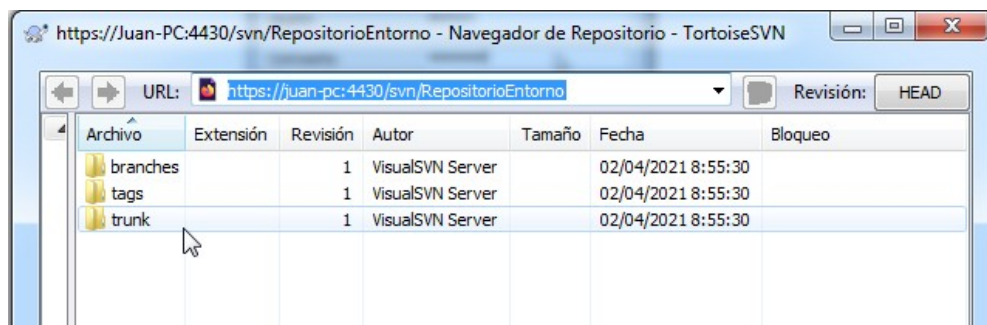
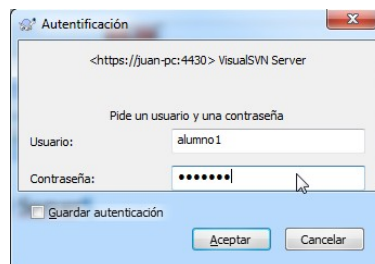


2.3.-Acceso al repositorio desde el navegador de repositorios del cliente TortoiseSVN



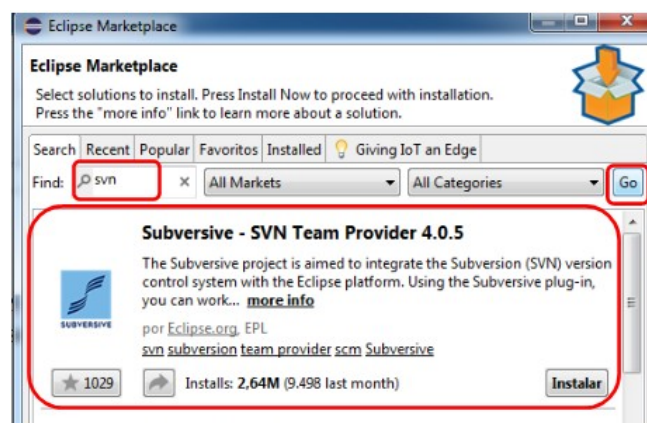
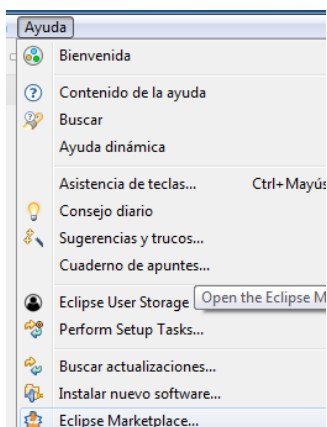


Introducimos la URL de nuestro repositorio VisualSVN Server

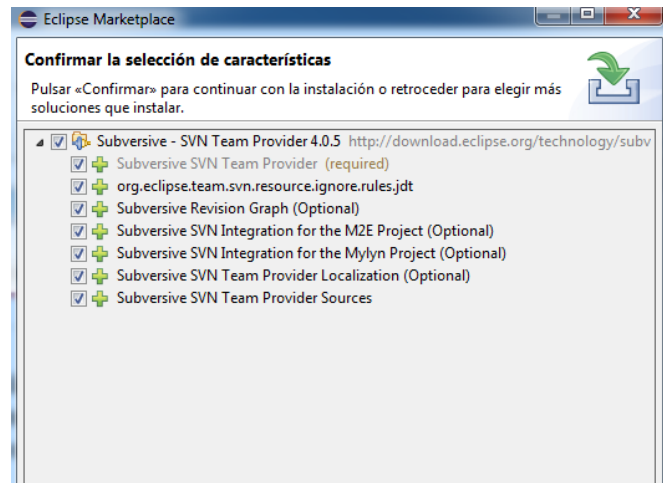


3.- Subversión en Eclipse

Para usar Eclipse como cliente de VisualSVN es necesario instalar el plugin Subversive SVN. Desde el menú Ayuda, seleccionamos Eclipse Marketplace .

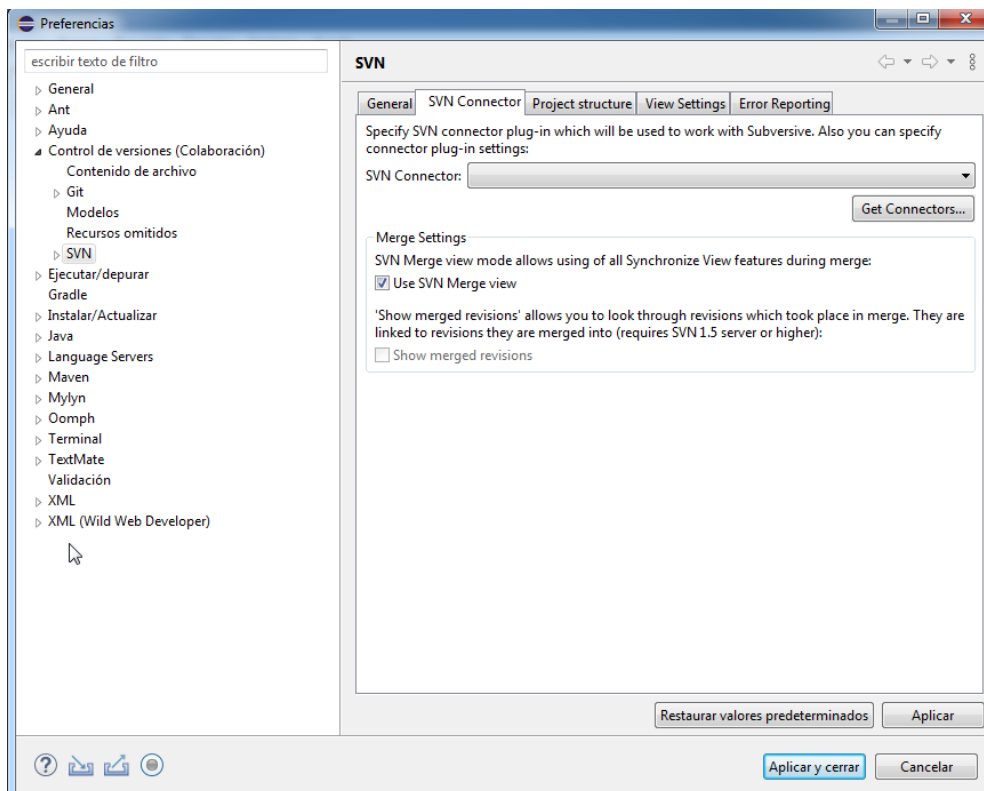


Buscamos svn y de los resultados de búsqueda seleccionamos SVN Team, hacemos clic en instalar . Este plugin permite interactuar directamente nuestros proyectos de eclipse con los repositorios locales y remotos que trabajan con subversion.

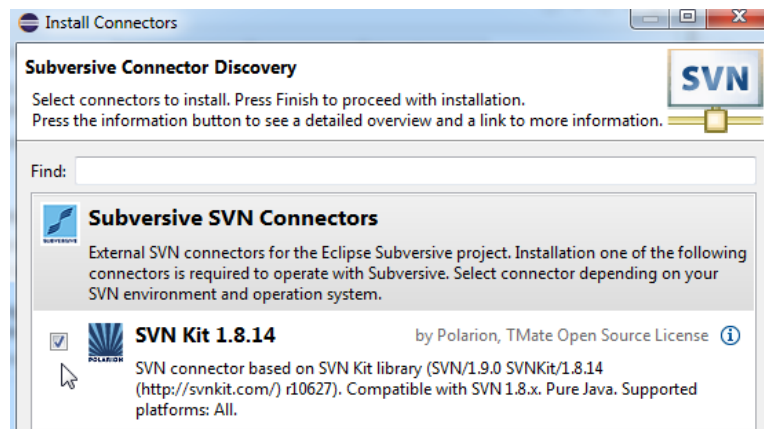


Para conectar los proyectos a un repositorio es necesario instalar los conectores de Subversive. La ventana para seleccionar los conectores se visualizará inmediatamente al crear un proyecto o al cargar un proyecto. Si esta ventana no aparece , se debe comprobar que el conector no está instalado desde el menú *Windows Preferencias*Team/SVN/SVN Conector, pestaña SVN Connector. En ese caso es necesario instalar los conectores.

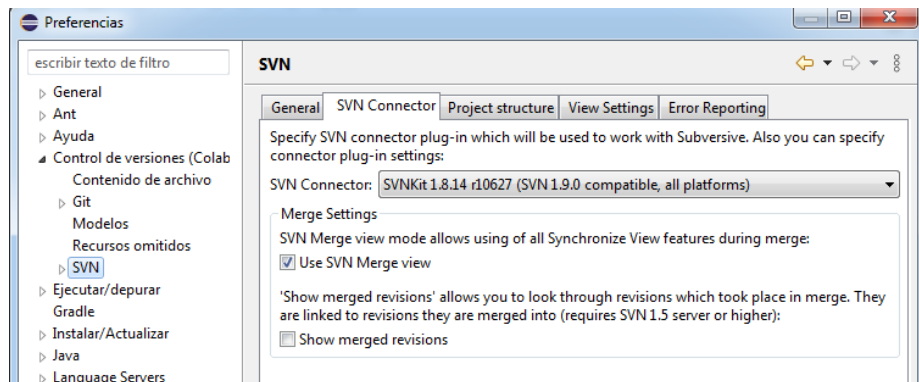
Ventana → Preferencias → Control de versiones → SVN → Pestaña SVN Connector



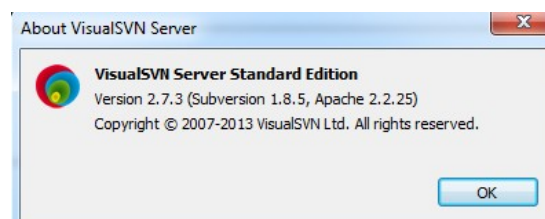
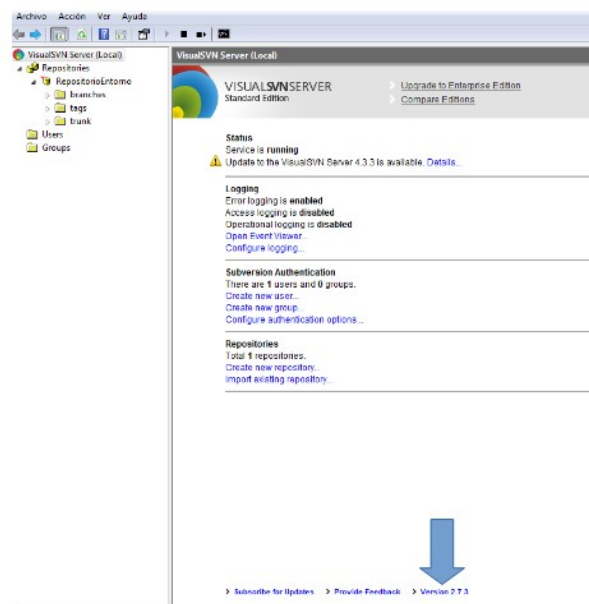
Hacemos clic sobre el botón Get Connectors



Activamos SVN kit 1.8.14 → Finalizar

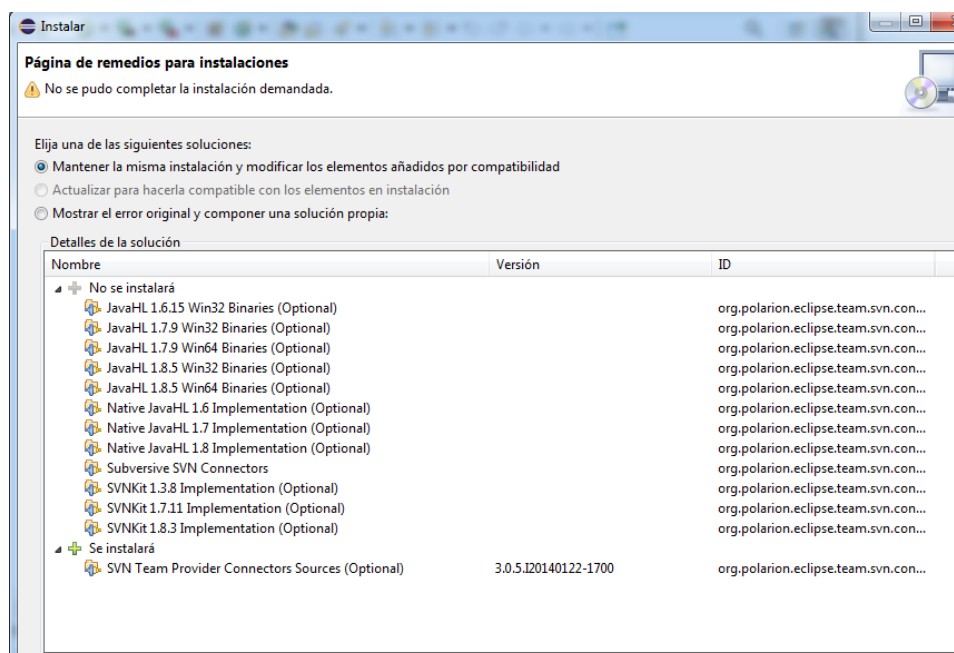
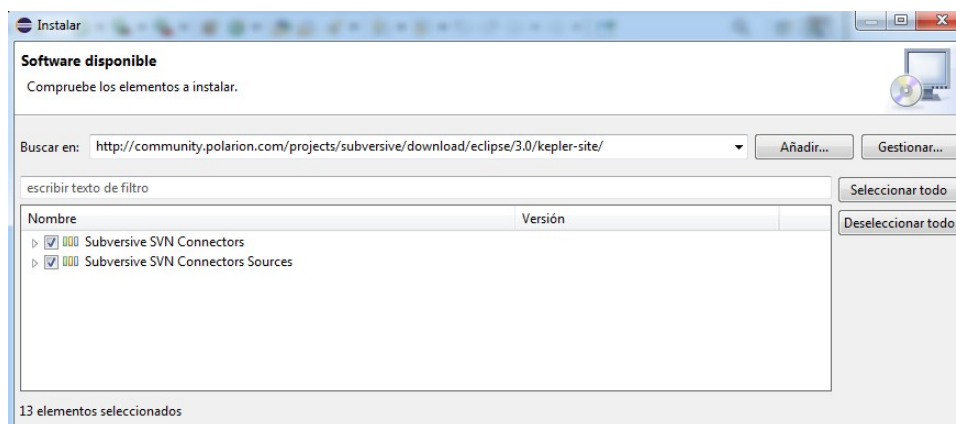


Para saber qué versión de conector instalar es necesario ver la versión de Subversive que se ha instalado. Desde la administración de consola del servidor VisualSVN, hacemos clic en el enlace de la versión de VisualSVN, se visualizará un cuadro indicando la versión de subversión y apache que se instala.



Nota: Si no se instala el conector , hay que entrar en el menú Ayuda → Instalar nuevo Software y en Buscar , introducir la siguiente URL

<http://community.polarion.com/projects/subversive/download/eclipse/3.0/kepler-site/>



Nota: Si al intentar conectar con un repositorio no se visualiza su estructura revisar que los servicios del VisualSVN estén inicializados . Desde Panel de Control → Herramientas Administrativa → Servicios.

VisualSVN Repository Configurator Service
VisualSVN Server

Provides ser...
VisualSVN S...

Iniciado

Manual
Automático

Servicio de red
Servicio de red

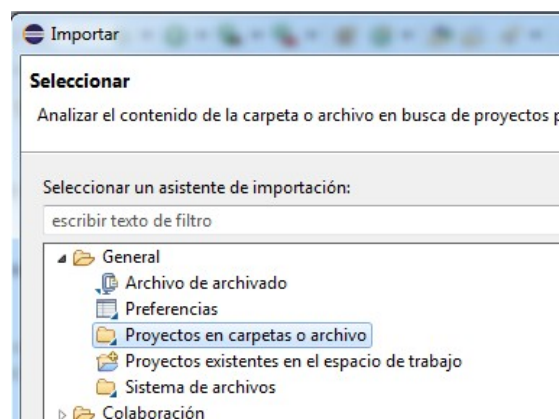
3.1.-Añadir un proyecto al repositorios

Una vez que ya se tiene el plugin y el conector instalado, lo siguiente es añadir un proyecto al repositorio.

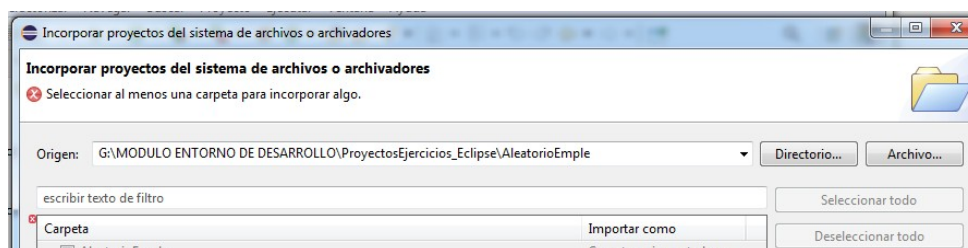
Crearemos un nuevo repositorio “RepositorioED2” desde VisualSVN Server con la estructura trunk-tags-branches , para trabajar con un repositorio limpio.

Creamos un nuevo proyecto Java o importamos un proyecto. En este caso vamos a importar el proyecto “AleatorioEjemplo”

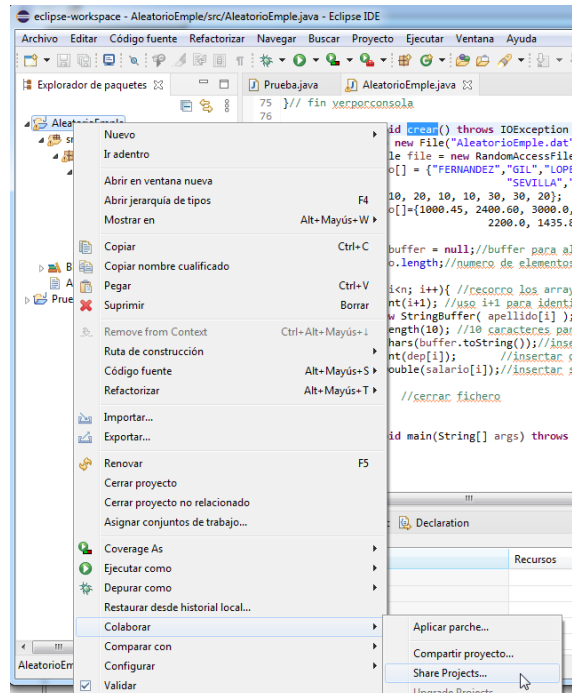
Archivo → Importar → Proyecto en carpeta o archivo



Introducimos el directorio donde se encuentra el directorio

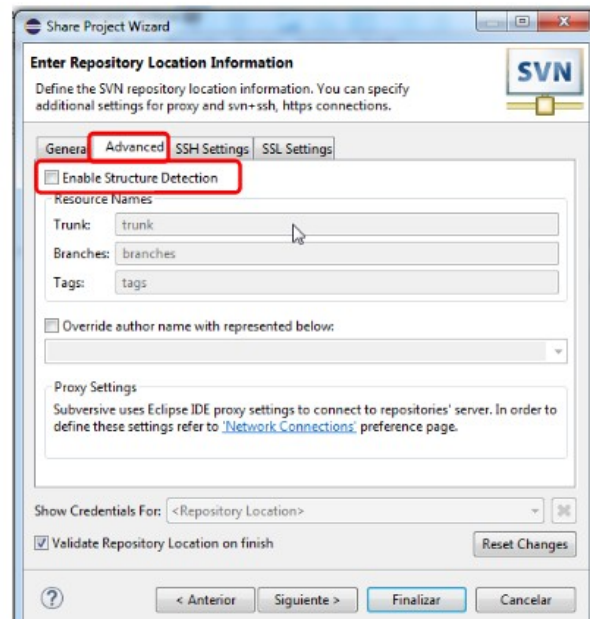
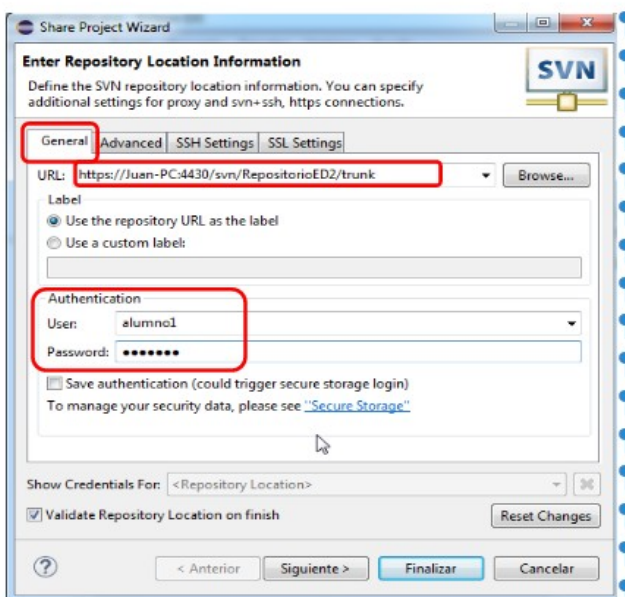


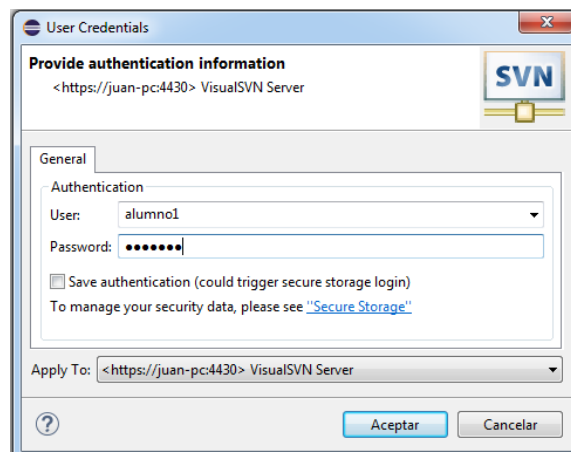
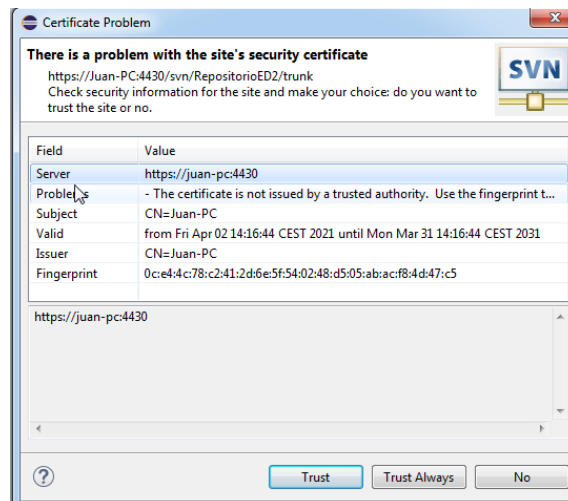
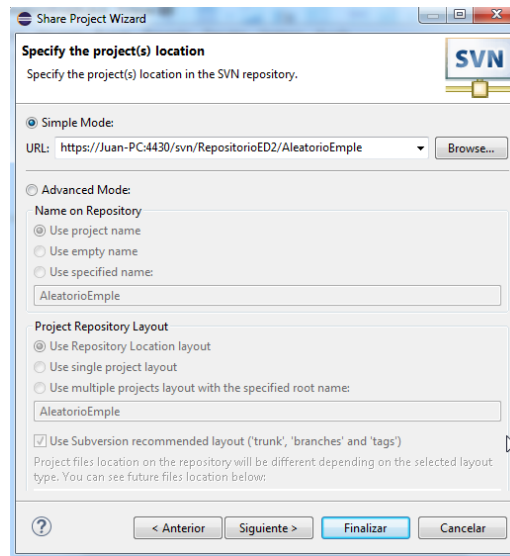
Sobre el proyecto, botón derecho del ratón .

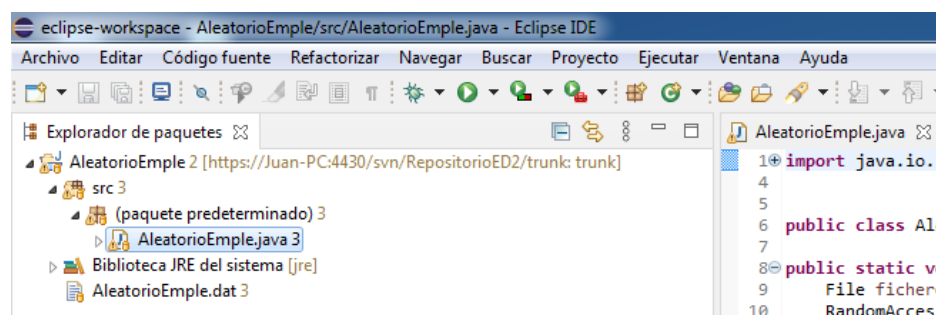
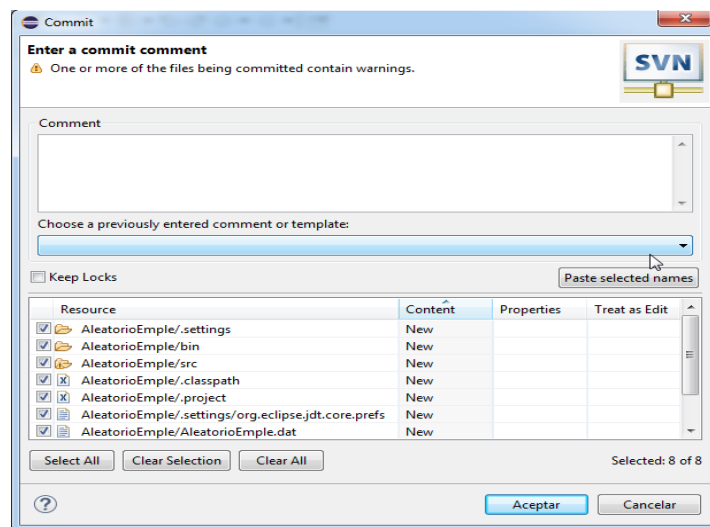


Hacemos clic en Colaborar → Share Project

En el primer cuadro de diálogo se elige el tipo de repositorio al que nos vamos a conectar , se elige SVN , en la pestaña Advanced se desmarca la casilla para poder cargar el proyecto dentro de trunk.

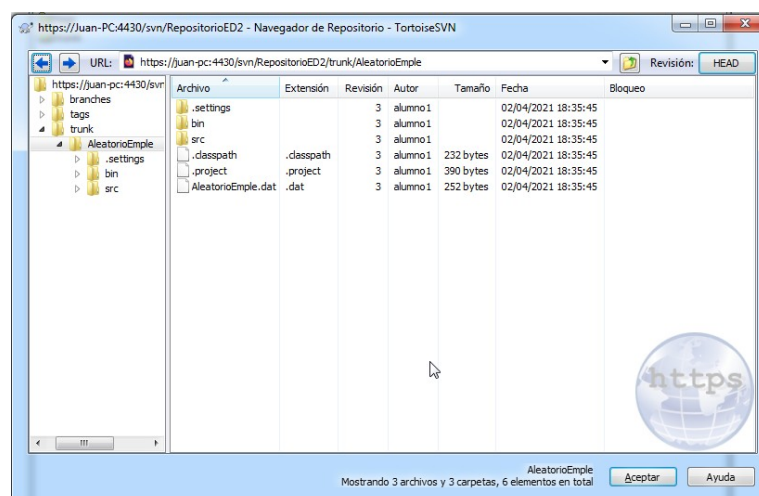






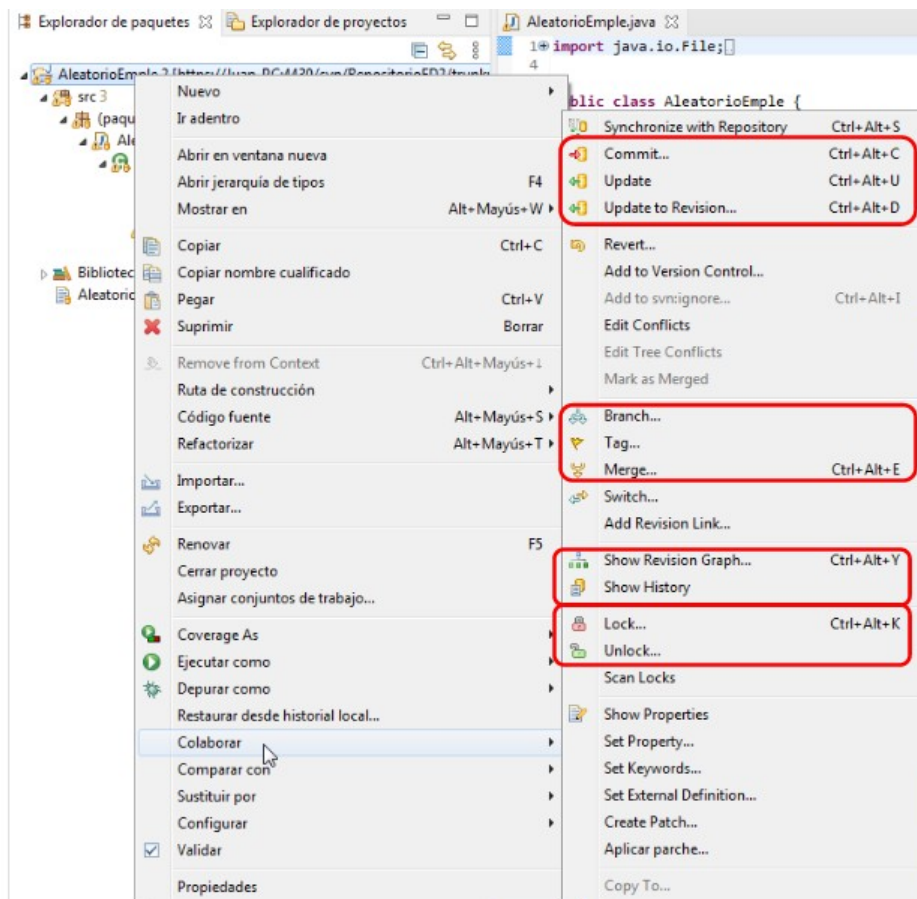
Observamos que cada nodo del proyecto aparece con un icono indicando que está en un repositorio, aparece también el número de revisión al lado y en la carpeta del proyecto se muestra la URL donde se encuentra .

En el Explorador de SVN Repository , podemos observar el repositorio , la estructura del repositorio, el proyecto almacenado , el navegador del repositorio, las revisiones , el usuario que hace los cambios , etc.

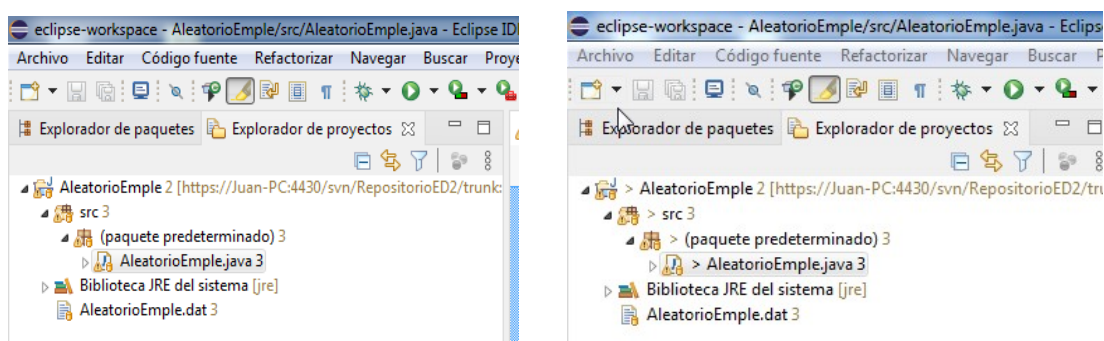


4.- Operaciones con SVN Eclipse

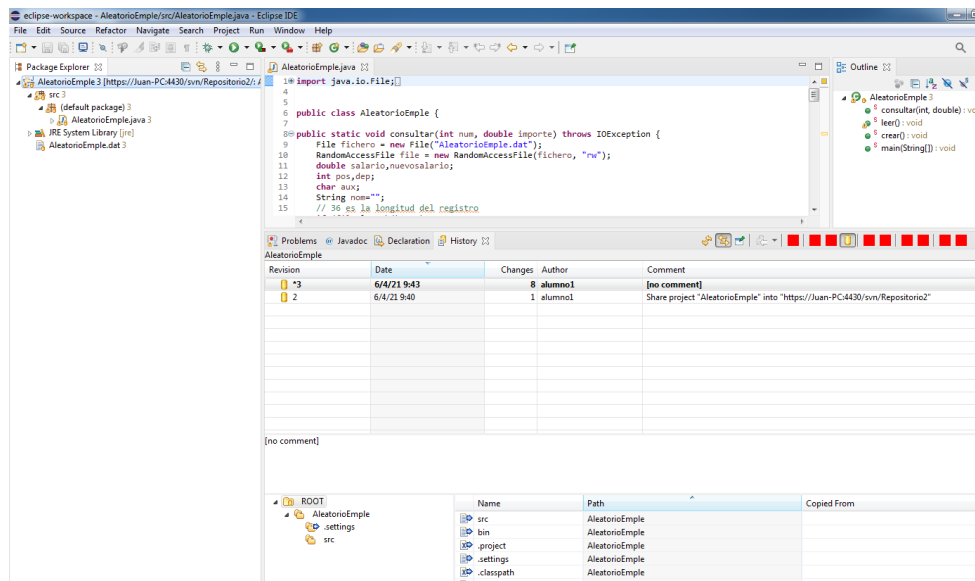
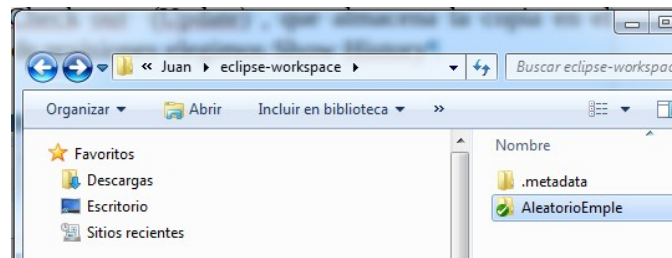
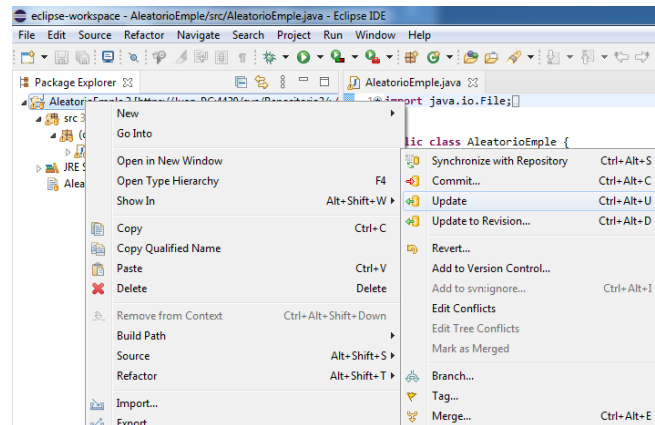
Las operaciones que se pueden realizar con SVN Eclipse son las mismas que las realizadas desde cualquier otro cliente. Nos posicionamos en el proyecto, o en un nodo del proyecto y desde el menú contextual (botón derecho ratón) se verán activas las opciones de SVN.



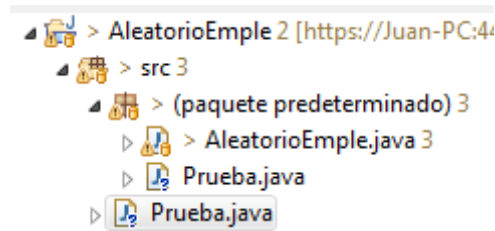
Si en Eclipse , realizamos cambios en el proyecto y guardamos el proyecto, aparecerá un símbolo “>” en los elementos cambiados



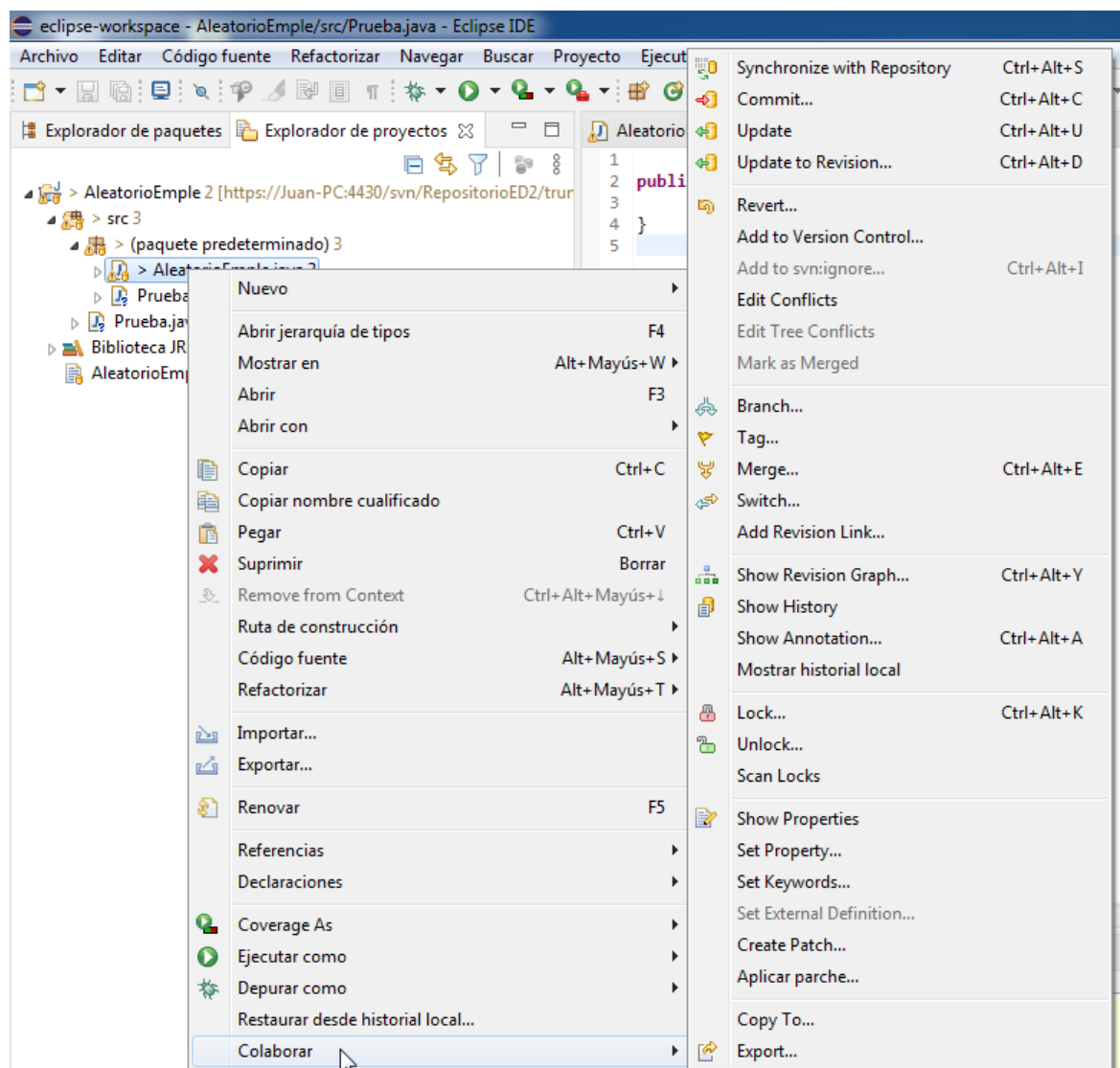
Para obtener una copia de trabajo haremos Check out (Update) , que almacena la copia en el espacio de trabajo actual , para ver el historial de revisiones elegimos Show History



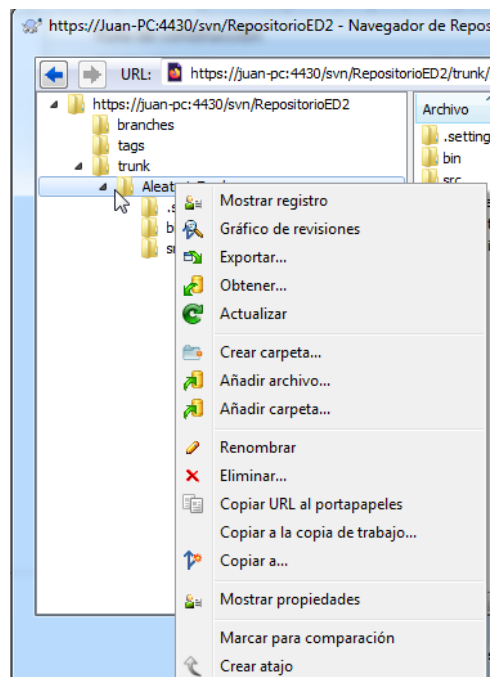
Si ahora creamos una nueva clase ,ejemplo “Prueba” , aparecerá una interrogación junto a los nuevos archivos, eso indica que aún no han sido añadidos al repositorio.



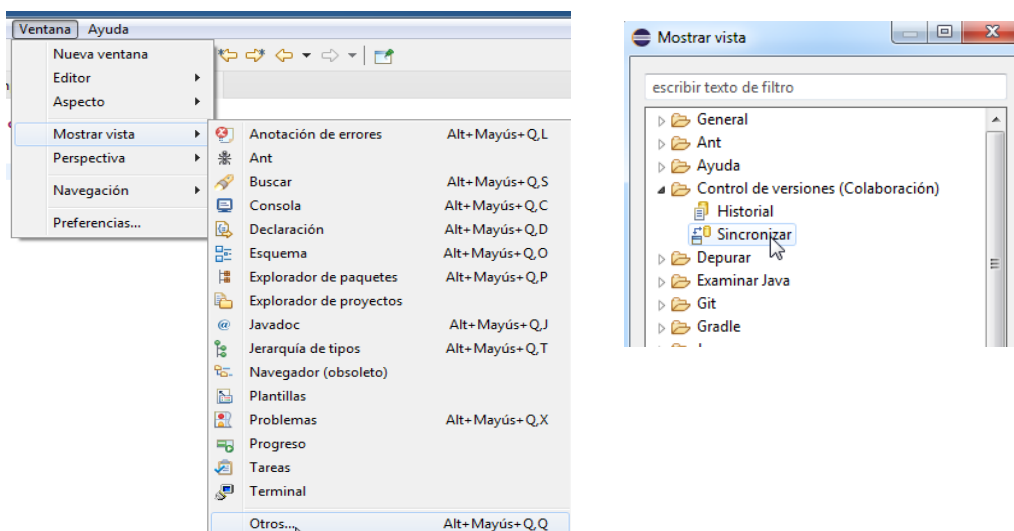
Si ahora se visualiza el menú contextual sobre el elemento, se verán todas las operaciones de SVN como hacer “Commit” o “Update”, crear etiquetas y ramas, hacer fusión, visualizar historial, o bloquear y desbloquear el elemento por ejemplo.



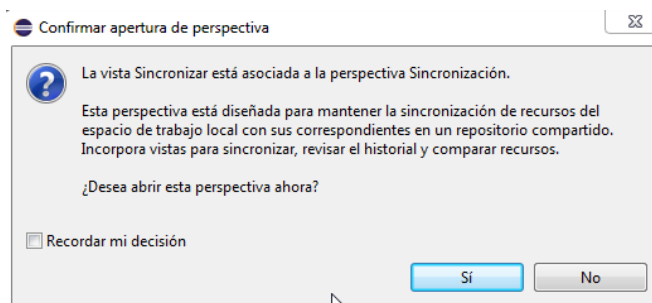
Menú contextual desde el explorador de repositorios



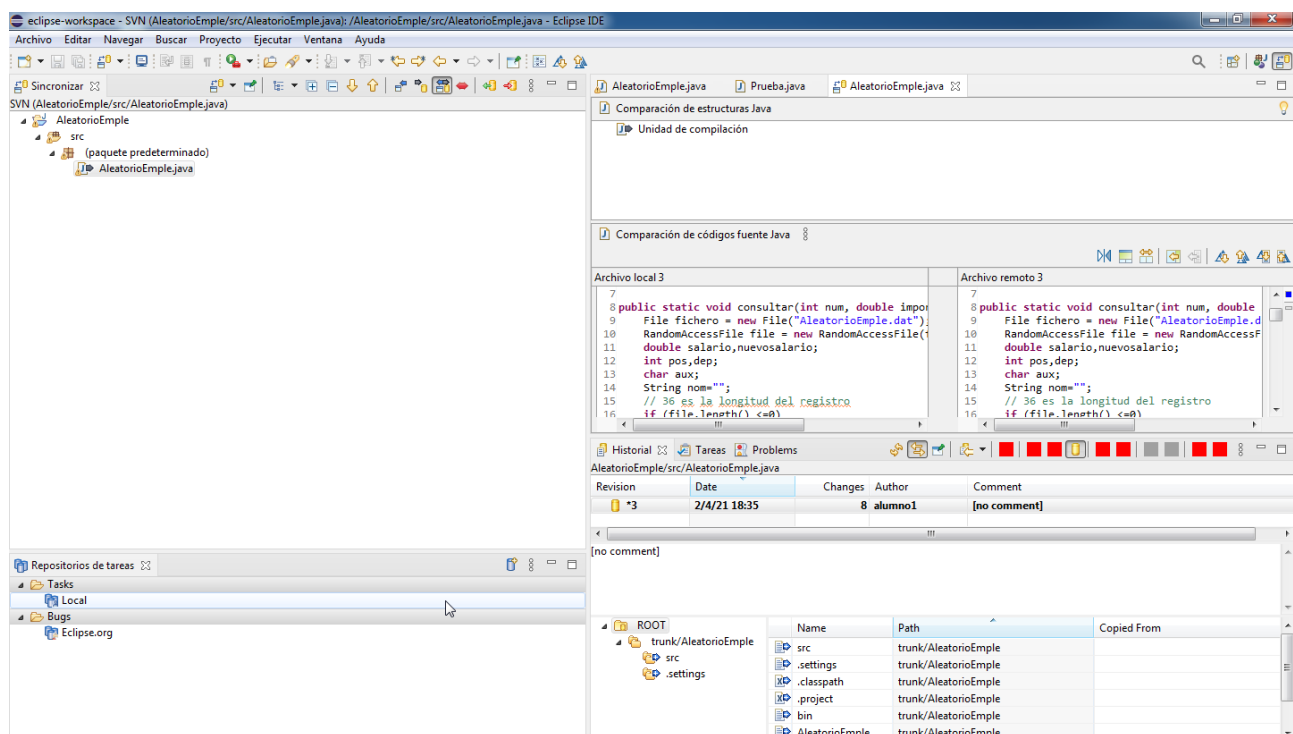
Cuando se realizan cambios se puede acceder a la pestaña “Synchronize” (si no se muestra , hay que ir a: (ventana → Mostrar vista → Otros → Control de versiones → Sincronizar)




Esta perspectiva nos mostrará los cambios locales que se van a subir al servidor, y viceversa, los cambios del servidor que nosotros deberíamos agregar localmente. Mediante esta perspectiva mantenemos la comunicación entre nuestro proyecto local de trabajo y el proyecto que está en el servidor.



Desde la barra de botones de esta perspectiva podremos hacer “update”, “validar”, o ver los conflictos. Cuando hay cambios salientes aparecen una flecha negra hacia afuera, si aparece el signo “+” es que el archivo es nuevo y no está versionado. Lo que deberíamos hacer es validar los cambios salientes mediante el botón

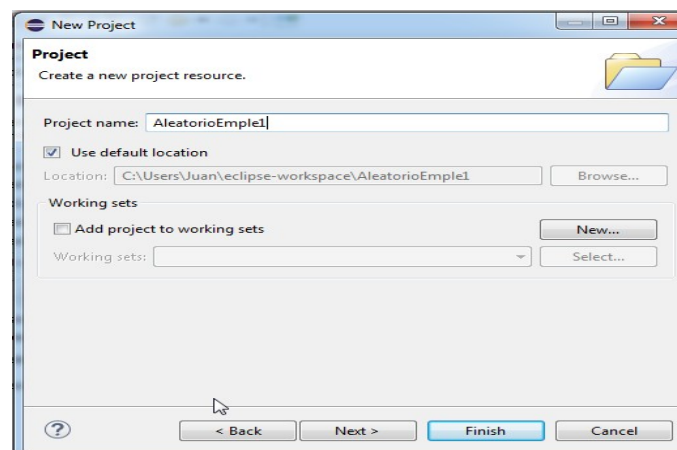
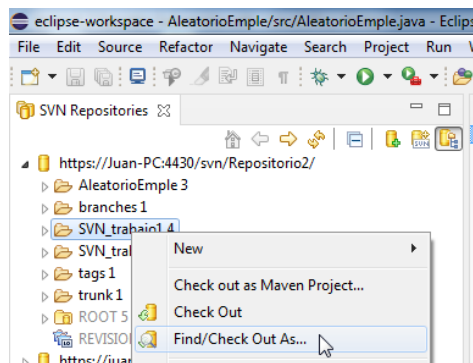
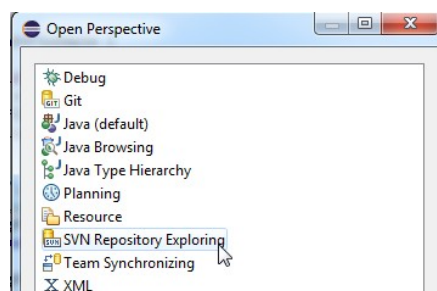
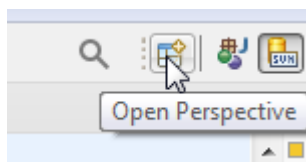


Quando hay cambios entrantes aparece una flecha hacia adentro en azul, si aparece el signo “+” es que el archivo es nuevo. Lo que hay que hacer es actualizar esos cambios entrantes, se pulsa el botón  que aparece en la barra de herramientas para actualizar, y luego validar los cambios salientes.

5.- Solución de conflictos

Un conflicto se va a producir cuando dos usuarios modifiquen el mismo archivo del repositorio, y las mismas líneas del archivo. Uno de ellos confirma los cambios, y a continuación el otro usuario los confirma también, el servidor va a detectar un conflicto pues se quieren validar cambios en una copia que no ha sido actualizada con la versión del repositorio, que fue validada por el primer usuario.

Para provocar un conflicto crea dos carpetas (ejemplo, SVN_TRABAJO1 y SVN_TRABAJO2) y realiza dos Checkout As desde la perspectiva SVN Repository Exploring del mismo proyecto, cambia el nombre al proyecto, llámalo AleatorioEmple1 y AleatorioEmple2, y almacena las copias de trabajo, una en cada carpeta.



Los proyectos se deben de abrir automáticamente en Eclipse , cada uno de ellos está en una carpeta diferente. Añadimos los siguientes cambios en la clase AleatorioEjemplo.java y lo guardamos en local:

En AleatorioEmple1 añadimos la declaración “int dato=0: “ debajo de char aux;

En AleatorioEmple2 añadimos la declaración “String cad=”HOLA” “ ; debajop de char aux;

Se debe de validar ahora los cambios del proyecto AleatorioEmple1, hacer commit, añadir el comentario (Se ha añadido la declaración int dato=0). Si ahora se validan los cambios del proyecto AleatorioEmple2, se visualizará una ventana indicando que hay un conflicto y que no se puede hacer COMMIT.

A continuación abrimos la vista o también la pestaña Synchronize, nos posicionamos sobre el proyecto AleatorioEmple2, pulsamos al botón derecho, se elige Team/Synchronize with Repository, observa que los conflictos aparecen marcados con una doble flecha o rombo en rojo, y en la parte inferior se indican los cambios entrantes , los salientes y los conflictos.

Si hacemos doble clic en el conflicto , es decir en la clase AleatorioEmple.java, se muestra una ventana en la que se comparan las dos versiones, en la parte izquierda se muestra el archivo de la copia local y en la derecha el archivo del repositorio, aparecen marcadas con borde rojo las líneas donde ha aparecido el conflicto. En este caso indicamos que se copie la línea del repositorio (derecha) a la copia local pulsando al cuadrado rojo.

Para resolver finalmente el conflicto, una vez recogidos los cambios en la copia local es necesario hacer Mark as Merged . Hasta que no se haga esto Subversive sigue creyendo que hay un conflicto con lo que no va a dejar validar.

Cuando tenemos cambios en la copia local que no se recogen en el repositorio los veremos marcados con una línea negra , y cuando son cambios que están en el repositorio y no los tenemos recogidos en local aparecerán marcados con una línea azul, pero estos no aparecen en conflicto porque son cambios en líneas diferentes. Si se hace clic en el cuadrado que aparece en el medio(en rojo o azul dependiendo del conflicto) lo pasa a nuestro archivo local , y listo, el archivo ya estaría sincronizado, ya solo faltaría hacer Mark as Merged y a continuación COMMIT.

EJERCICIO 1

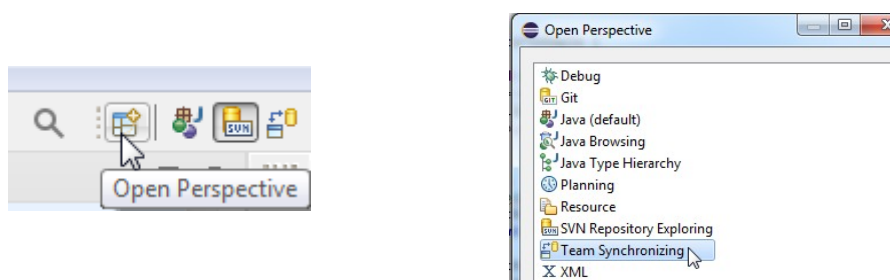
Antes de realizar nuevos cambios, comprueba que los dos proyectos tienen la misma versión y están actualizados con el repositorio. Hay que actualizar AleatorioEmple2 , para actualizar : botón derecho sobre el proyecto elegir Team- → Update. Una vez que los dos proyectos estén actualizados y marcan la misma revisión, realiza los siguientes cambios en la misma clase AleatorioEmple.java

En AleatorioEmple1 añade el método prueba , en cualquier sitio y guarda los cambios en local

```
public static void prueba()
{
    // Se ha añadido un método
}
```

En AleatorioEmple2 añade la declaración `int prueba=0;` debajo de `public static void` . Guardar los cambios en local.

Valida los cambios de AleatorioEmple1 . Con el proyecto AleatorioEmple2 visualiza la pestaña Synchronize y observa los cambios.



Aparecen conflictos , si se hace doble clic sobre el conflicto se muestra la versión del repositorio y la local.

Observa que ahora los conflictos no aparecen marcados en rojo porque no son las mismas líneas la modificadas, ahora se muestran en color negro , los cambios que se han realizado en AleatorioEmple2 (cambios salientes) y en azul los cambios realizado en AleatorioEmple1 y que fueron validados (cambios entrantes).

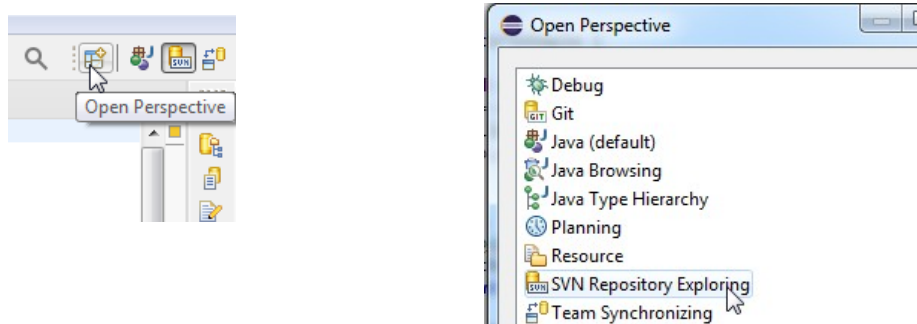
Pulsa en el recuadro azul para pasar los cambios a la copia local , guarda el proyecto , pulsa el update y a continuación valida los cambios al repositorio , haz commit.

Finalmente para que los dos proyectos queden en la misma versión , haz un update en cada uno de ellos, desde la carpeta del proyecto , botón derecho del ratón Team/Update

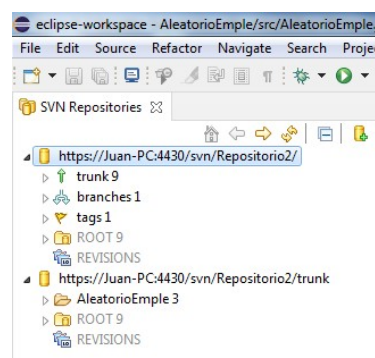
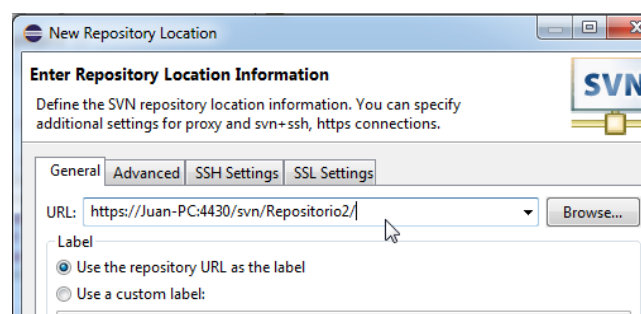
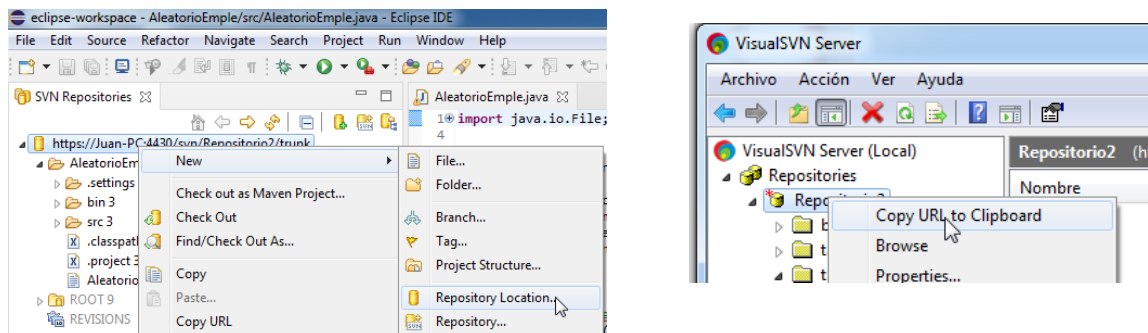
OBSERVACIÓN: Antes de trabajar con un proyecto hacer un sincronize , y un update, para asegurarnos de estar trabajando con la última versión del repositorio.

6.- CREAR RAMAS

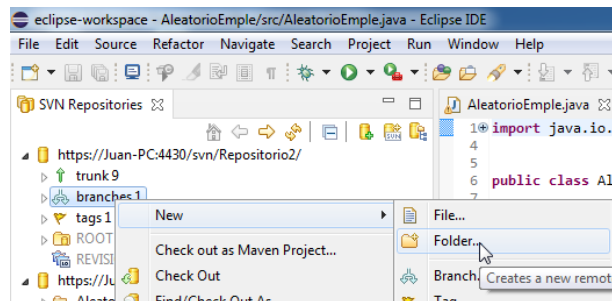
Para crear una rama se hace desde la perspectiva SVN Repository Exploring de Eclipse.



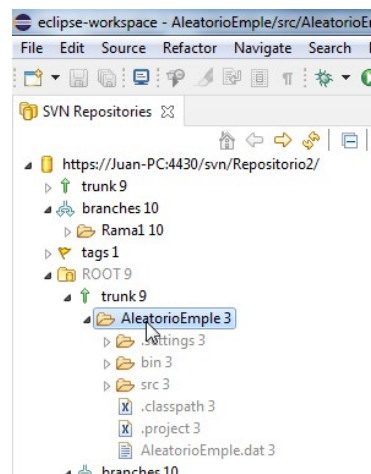
Nos situamos en la raíz del del repositorios



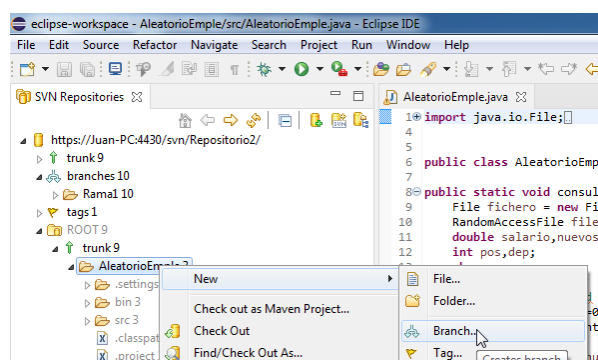
Para subir un proyecto a una rama se hace desde la perspectiva SVN Repository Exploring . Hay que situarse en la raíz del repositorio, y creamos una carpeta dentro de la rama (botón derecho new → folder)



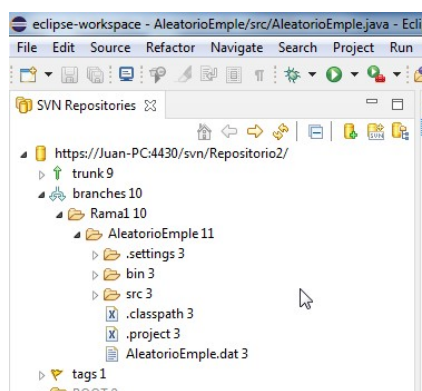
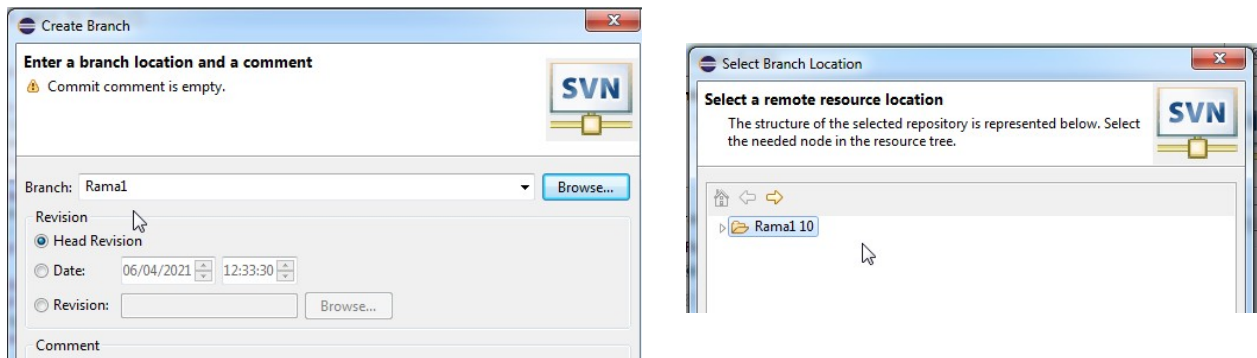
Nos posicionamos sobre Root (seguimos situados en la raíz del repositorio) , desplegamos y buscamos el proyecto a subir a la rama



Nos posicionamos sobre él , pulsamos botón derecho del ratón y elegimos new → Branch

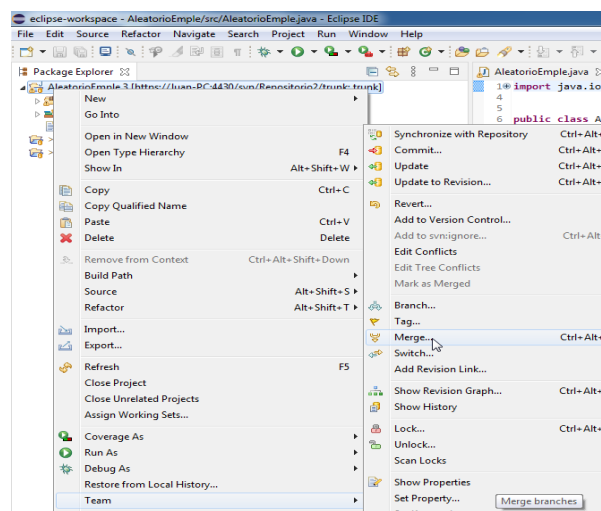


En la ventana que se muestra se elige la carpeta que creamos anteriormente en la rama

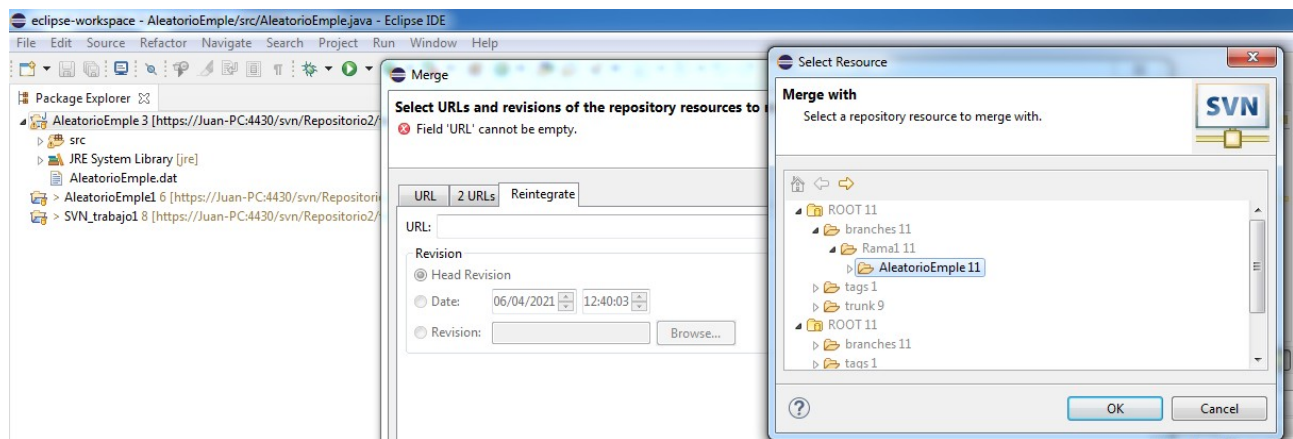


A partir de ahora , los Check out se harán desde las ramas . Se cargarán en el disco duro, se modificarán las copias locales de las ramas , y se validarán con la correspondiente rama del repositorio.

Para añadir los cambios de las ramas a la versión del trunk , se hace un Merge/Reintegrate



desde la copia de trabajo del trunk con la copia del repositorio de la rama . Nos posicionamos en la copia de trabajo del proyecto (suponemos que está sincronizado y actualizado con la versión del repositorio) . Pulsamos el botón derecho del ratón y se elige (Team/Merge pestaña Reintegrate)



A continuación se muestra la vista Synchronize con los cambios detectados, se aceptan los cambios y luego se realiza un commit de la copia local del proyecto de trunk con la del repositorio. Por último se realiza un update para tener sincronizadas la versión de la copia de trabajo y la versión del repositorio.

EJERCICIO 2

Copia el proyecto FicheroAleatorioVentana al workspace de trabajo y ábrelo.

Sube el proyecto al repositorio, colócalo dentro de trunk y añade un comentario a la subida.

Crea una rama llamada RamaFicheroAleatorioVentana. Añade a la rama una copia del proyecto.

Haz un Checkout as de esa rama en una carpeta de trabajo, puede ser la misma del workspace , ponle un nombre diferente al proyecto para distinguirlo.

Realiza cambios en la copia de la rama, por ejemplo , añade un método de prueba, comentarios, etc. Valida esos cambios.

Guarda los cambios que tiene la rama en la versión del trunk , recuerda que hay que hacerlo desde una copia de trabajo del trunk , esta copia debe de estar sincronizada con su copia del trunk, es decir , realiza un Merge- → Reintegrate

Valida los cambios y sincroniza ambas copias de trabajo , para que todas ellas se queden en la misma versión.

6.1.-Conflictos en la reintegración de Ramas

Para integrar los cambios realizados en las copias de trabajo de las ramas , a las copias de trabajo del trunk(tronco), el número de revisión de las ramas debe de ser mayor que la de trunk , pues se supone que los cambios se realizan en la rama.

Se intenta integrar cambios que modifican las mismas líneas en los dos proyectos, se van a producir conflictos. En el fichero local aparecerán unas líneas como las siguientes:




```
«««««««« .working
public static void prueba2_metodo_rama()
{
=====

public static void prueba2Cambio_metodo_rama()
{
    int a=0;
»»»»»»»»»»»»> .merge-right.r24
}
```

En estas líneas se indica el conflicto encontrado , lo que hay en la copia de trabajo (.working) y lo que hay en la revisión 24 en esas mismas líneas.

Además se generarán varios archivos: uno de texto con la extensión .java.working (por ejemplo AleatorioEjemplo.java.working) que contiene una copia de trabajo de la clase en conflicto. Y otro dos con las dos versiones estables a las que poder restaurar para solucionar el conflicto, la versión de la izquierda que llevará el nombre de .merge-left.r, es el último Update exitoso que se hizo en el trunk , por ejemplo AleatorioEmple.java.merge-left.r19 , y la versión de la derecha que llevará el nombre .merge-right.r, que es la versión del último Update exitoso de la rama , por ejemplo AleatorioEmple.java.merge-right.r24 (el número indica la versión).

Para resolver este conflicto podemos proceder de dos formas:

1. Editar el conflicto (en la ventana Synchronizing de Eclipse , botón derecho del ratón sobre el archivo en conflicto, elegir “Edit Conflicts”), se muestra la ventana con los archivos en conflicto, y marcadas en rojo las líneas conflictivas . Se pulsa al cuadrado rojo , se hace Update pulsa el botón  , se marca el fichero como “mergeado” Mark as Merged (botón derecho sobre el fichero  y elegimos *Team/Mark as Merged*) . observar que desaparecen los ficheros que se crearon en el conflicto. Finalmente se pulsa el botón de commit 
2. Otra posible solución es quedarse con una de las versiones en conflicto. Para ello se borra el fichero Java que da el conflicto. A la versión elegida se le quita la extensión .merge-right.r o .merge-left.r , (botón derecho sobre el fichero, se elige Refactor/Rename, y se cambia el nombre). Y ya solo queda marcarlo como “mergeado” Mark as Merged (botón derecho sobre el fichero y elegir Team/Mark as Merged), y hacer commit.

EJERCICIO 3

Realiza cambios en la actividad del ejercicio anterior . En la copia de trabajo de la rama, cambia el nombre al método creado, añade un parámetro, y cambia una de las líneas de comentario creadas. Valida los cambios.

- Reintegra los cambios de la rama con la copia de trabajo del trunk.
- Resuelve el conflicto surgido.