**Trabajar con SVN**

Introducción

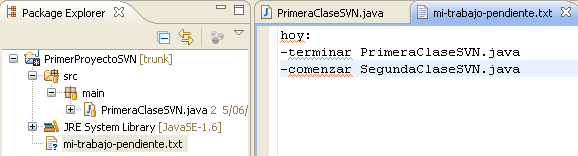
En este tutorial vamos a practicar el uso de los principales comandos de SVN, necesarios para la correcta gestión de nuestros proyectos. Así veremos:

* ignore: evitar que SVN gestione ciertos ficheros que tenemos en local y que no deseamos subir al repositorio.
* add: justo lo contrario que ignore, es decir, añadir un recurso a SVN.
* update: actualizar nuestro proyecto local con los cambios o novedades de SVN realizados por otros desarrolladores.
* delete: eliminar recursos del proyecto local y por ende de SVN.
* Resolución de conflictos: ¿qué sucede cuando dos programadores han modificado las mismas líneas de una clase?
* Visualizar historial de cambios, recuperar ficheros eliminados, volver a una versión anterior.

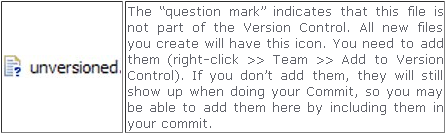
Operaciones habituales

**Ignorar ficheros**

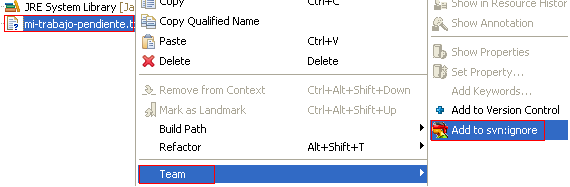
Suele ser habitual que un programador tenga una serie de ficheros que no desea subir a SVN. Por ejemplo, supongamos que hemos creado el fichero ‘mi-trabajo-pendiente.txt’, en el que vamos anotando tareas que pensamos abordar en la sesión actual de trabajo. Este fichero no le interesa a nadie salvo a nosotros mismos y tampoco se trata de algo crítico, por lo que decidimos no subirlo a SVN:



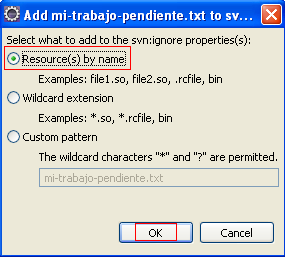
Notad que el fichero aparece en el ‘Package Explorer’ con el icono ‘?’, indicativo de que tal recurso no está gestionado por SVN.



Para informar a SVN de que el fichero jamás formará parte de su repositorio tenemos que hacer clic derecho sobre el fichero y seleccionar **‘Team->Add to svn:ignore’**:

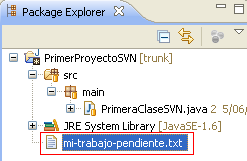


En la ventana que aparece tenemos la posibilidad de ignorar sólo este archivo o todos aquellos con una determinada extensión o incluso establecer un patrón personalizado, cuyos archivos que lo cumplan serán ignorados por SVN:



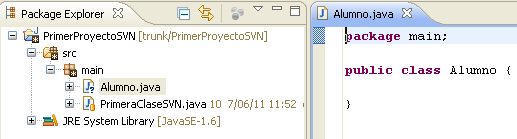
Dejamos seleccionada la que aparece por defecto y aceptamos.

Notad que ha desaparecido el icono ‘?’ y por supuesto, cuando hagamos commit (actualización desde el proyecto local hacia el repositorio), nunca se subirá este fichero a SVN:

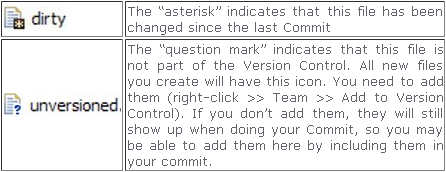


**Añadir recursos a SVN**

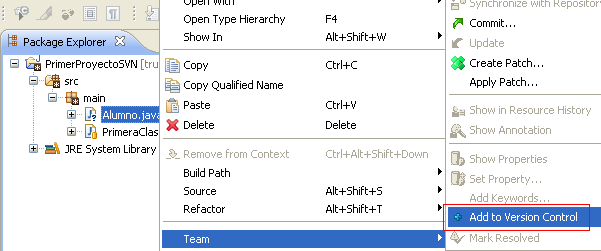
Para abordar esta sección, supongamos ahora que somos otro miembro diferente del equipo de desarrolladores. Estamos trabajando en el mismo proyecto y hemos creado la siguiente clase Alumno:



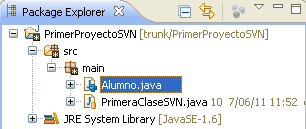
Notad que la clase Alumno tiene un icono en forma de interrogante, lo que indica que SVN desconoce su existencia. Fijaos también en que el proyecto y todas las carpetas tienen un icono en forma de asterisco, indicativo de que el proyecto local ha sufrido cambios respecto al último commit (la última actualización). No obstante, dado que aun no hemos realizado ningún commit, tenemos que interpretarlo como que no ha habido cambios desde la bajada del proyecto desde SVN.



Para añadir ficheros y carpetas al repositorio tenemos que seleccionar **‘Team->Add to Version Control’**, tal y como se muestra a continuación:



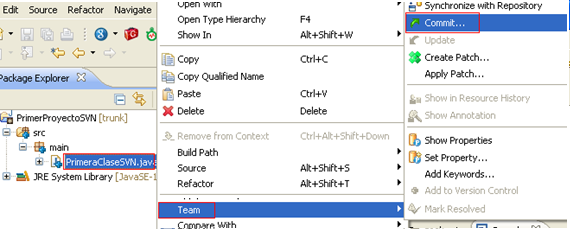
Esto sólo lo tenemos que hacer una vez (siempre hay una primera vez). Ahora podremos ver que el icono ha cambiado, ya no tenemos el interrogante, sino un signo ‘+’ azul:



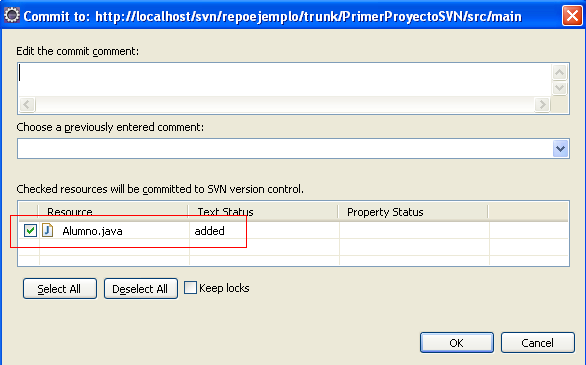
A description...

Esto indica que esta clase ahora está gestionada por SVN y que en cuanto hagamos un ‘commit’ se subirá al repositorio central.

Bien, supongamos que ya estamos satisfechos con nuestra clase y que ha pasado todas las pruebas. Nos disponemos a hacer el commit a SVN, por lo que seleccionamos la clase Alumno y hacemos clic derecho:

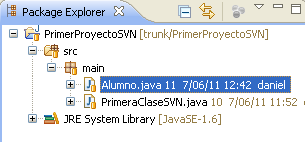


Aparecerá el siguiente cuadro de diálogo, en el que podremos escribir un comentario relativo a la subida del fichero. Notad que mediante ‘Keep locks’ podemos marcar la clase para que esté bloqueada para otros programadores:



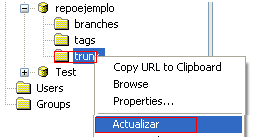
Aceptamos.

Notad que el icono ha vuelto a cambiar. Ahora se muestra esa especie de cilindro amarillo para indicar que el código está ahora en SVN y sincronizado con nuestra versión local:

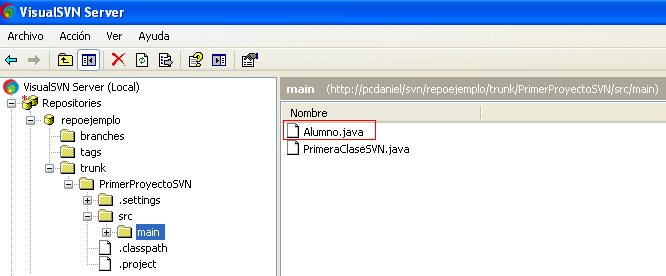


A description...

Fuera de Eclipse, en la consola de VisualSVN Server, podemos hacer clic derecho sobre la carpeta ‘trunk’ de nuestro repositorio y seleccionar ‘Actualizar’ para verificar qué efectivamente la clase Alumno reside ahora en el repositorio:

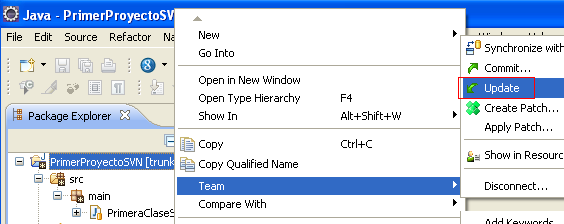


Veremos lo siguiente:

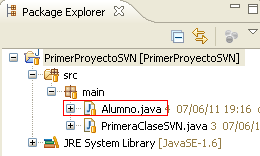


**Actualizar nuestro proyecto local con los cambios/novedades de otros miembros**

Si ahora volvemos al workspace del primer miembro (el que inicio el proyecto), veremos que en su copia local no dispone de la clase Alumno, ya que ésta ha sido creada por el segundo miembro. No obstante la clase existe en SVN, por lo que lo único que hay que hacer es bajarla. Esto se hace mediante **‘Tema->Update’**:



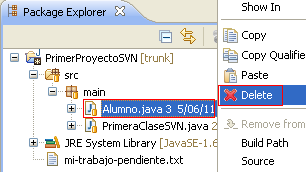
Y ahora sí:



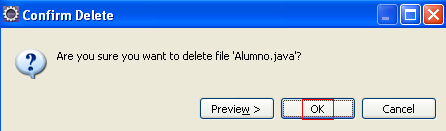
Además de aparecer ahora la clase Alumno notad que los iconos ‘\*’ se han sustituido por el cilindro amarillo, indicando que la copia local está sincronizada con la remota.

**Eliminar recursos**

Ahora vamos a suponer que consideramos que la clase Alumno ya no es relevante para el proyecto, por lo que sencillamente la marcamos y seleccionamos ‘Delete’ (no es necesaria la opción ‘Team’):



Confirmamos la eliminación:

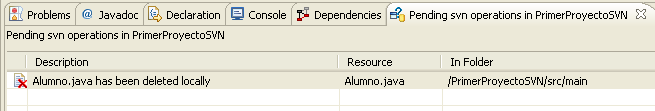


Subclipse ha marcado el fichero para eliminarlo automáticamente y lo ha borrado de nuestro proyecto local. La próxima vez que hagamos un commit será borrado de SVN.

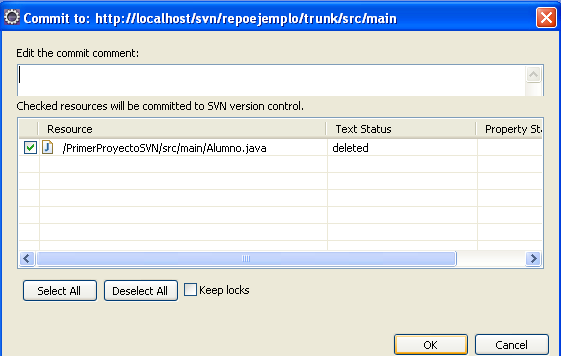
Podemos ver las operaciones locales pendientes de hacer commit mediante **‘Team->Show pending operations’**:



Esto mostrará lo siguiente:

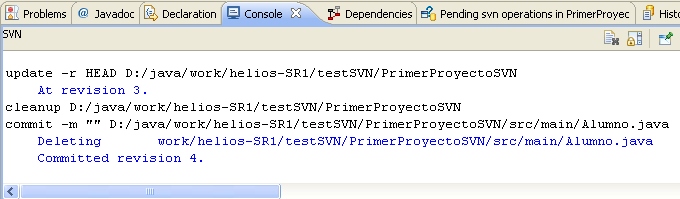


Vamos a hacer el commit, notad cómo la clase Alumno aparece para ser eliminada del repositorio central de código:

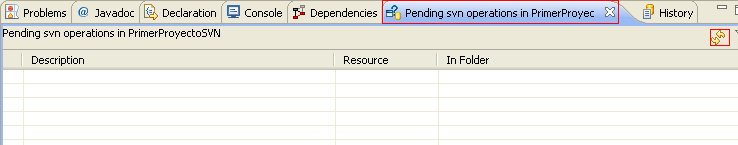


Aceptamos.

En la consola veremos un mensaje parecido al siguiente:



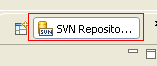
En la ventana ‘Pending svn operations…’ no tenemos que observar ahora nada:



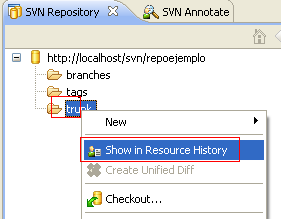
**Visualizar el historial de cambios almacenados en SVN**

Si queremos ver las operaciones realizadas con SVN en orden cronológico tenemos que consultar el historial que hay allí almacenado.

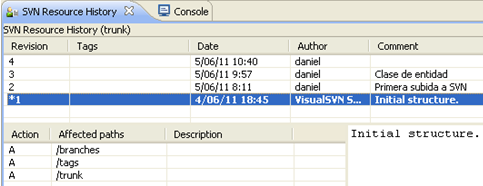
Vamos a la perspectiva ‘SVN Repository Exploring’:



Hacemos clic derecho sobre ‘trunk’ y seleccionamos ‘Show in Resource History’:

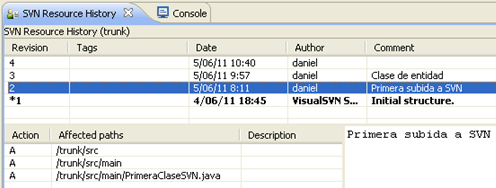


Podremos ver una ventana como la que sigue:

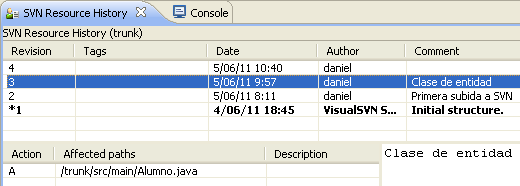


Notad que estamos situados sobre la línea cuyo comentario es ‘Initial structure’ y en la parte inferior vemos como se añadieron a SVN las tres carpetas principales.

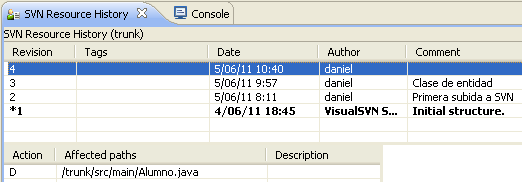
Si ahora vamos a la anterior línea veremos que lo que en ese instante hicimos fue añadir (la columna Action tiene una ‘A’ de append) una serie de directorios y la clase PrimeraClaseSVN.java:



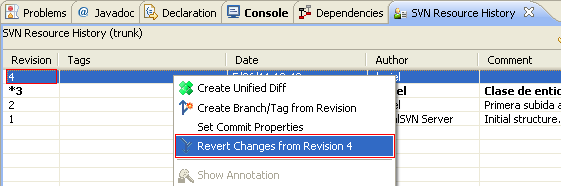
Al ir a la anterior línea vemos cómo en esa ocasión lo que hicimos fue crear la clase Alumno.java:



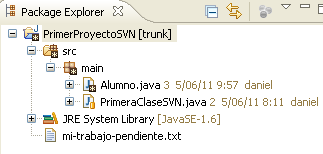
Finalmente, en la primera línea vemos que en ese instante eliminamos la clase Alumno.java (la columna Action tiene una ‘D’ de delete):



Si quisiéramos volver a la versión 3, cuando aún teníamos la clase Alumno, haríamos lo siguiente: marcaríamos la revisión 4, clic derecho, seleccionaríamos ‘Revert Changes from Revision 4’.

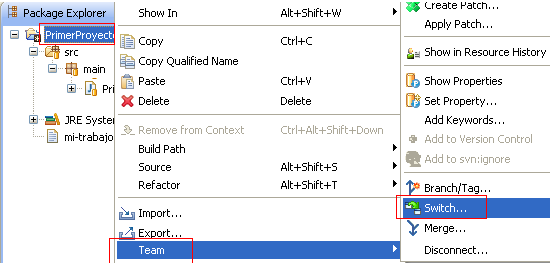


Con lo que obtendríamos lo siguiente:

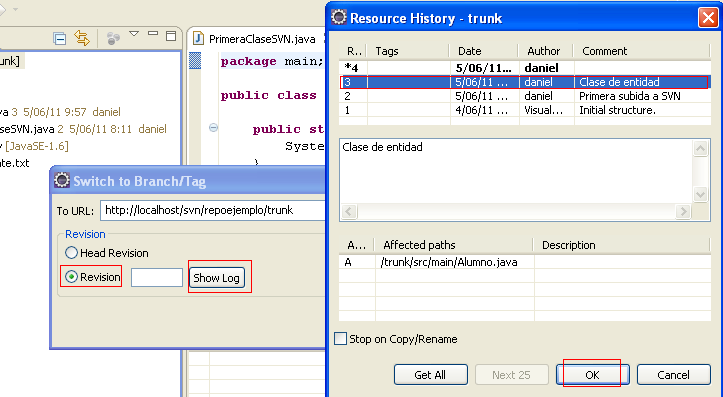


**Otra manera de volver a una versión anterior**

Para que este ejemplo tenga sentido, tendremos que eliminar nuevamente la clase Alumno. Seleccionamos el proyecto, clic derecho y ‘Team->Switch…’:

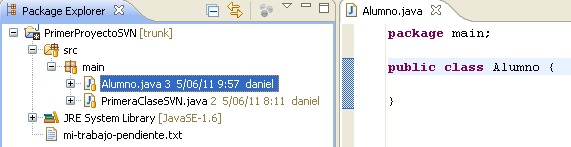


En el cuadro de diálogo que aparece seleccionamos ‘Revision’ y pulsamos sobre ‘Show Log’, con lo que se abrirá una nueva ventana en la que seleccionaremos la revisión 3, donde aún no habíamos eliminado la clase Alumno. Lo aceptamos todo.





Y obtenemos lo siguiente:



**Trabajo en equipo y resolución de conflictos**

Se refiere a la situación en la que al menos dos personas modifican el código.

Mientras otras herramientas obligan a bloquear zonas del repositorio cuando se estén realizando cambios en ellas, SVN permite la modificación paralela de código del repositorio, de modo que varias personas pueden trabajar de forma simultánea sobre cualquier parte del código sin crear interferencias.

En el caso de que dos desarrolladores modificasen el mismo elemento a la vez, Subversion integrará los cambios de forma automática, obligando al usuario a hacerlo de forma manual sólo en casos en los que el conocimiento humano es el único que puede asegurar la correcta integración.

Antes de hacer cualquier modificación en su entorno local, los desarrolladores deben asegurarse de estar trabajando con la última versión del software del repositorio. Lo mismo sucederá al finalizar un desarrollo: antes de persistir los cambios en el repositorio de Subversion se deberá asegurar que no se está interfiriendo con un desarrollo paralelo que ya haya sido guardado en el repositorio. Para esto se utilizará el mecanismo de sincronización de Subversion.

Buenas prácticas: Existen tres formas de sincronizar el código del entorno local con el del repositorio:

* El comando Checkout descargará al entorno local una copia fiel del código del repositorio. Útil para comenzar a desarrollar sobre proyectos nuevos.
* El comando Update descargará al entorno local únicamente las modificaciones que hayan tenido lugar desde la última sincronización. Sólo se podrá hacer esta operación si se dispone ya de una versión local del código del repositorio.
* El comando Commit actualizará el contenido del repositorio con los cambios del entorno local. Subversion sólo permitirá esta operación si no existen conflictos con el código ya existente en el repositorio. Es decir, no permitirá hacer Commit si otro miembro del equipo ha modificado el mismo elemento de forma paralela desde la última sincronización de código.

Para favorecer el trabajo en equipo, se recomienda la orientación del desarrollo a tareas de resolución a corto plazo.

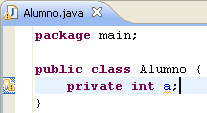
La dinámica habitual de trabajo deberá ser la siguiente:

1. Antes de comenzar con la resolución de una tarea, se deberá asegurar la sincronización con el repositorio, bien con un Update o bien con un Checkout dependiendo de si se dispone previamente del código en el entorno local o no.
2. Una vez finaliza una tarea, se deberá hacer Update para traer al entorno local tanto tareas nuevas como modificadas efectuadas por otros programadores. Subversion sabrá integrar los cambios del repositorio con los del entorno local en la mayoría de los casos, pero existirán situaciones que requieran de intervención humana para la integración. Estos casos, se deberán resolver de forma manual procurando mantener las modificaciones propias y las realizadas por los otros desarrolladores en paralelo.
3. Finalmente se deberá hacer el Commit para hacer público al resto del equipo el código desarrollado. El alcance del Commit deberá limitarse al código relevante a la resolución de la tarea, y no mezclar desarrollos de distintas tareas en un mismo Commit.

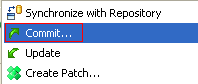
En un equipo de trabajo más tarde o más temprano se acaban produciendo conflictos relativos a las modificaciones del código, siempre y cuando, eso sí, se trabaje sin bloqueos (la forma normal).

Veamos un ejemplo. La situación es la siguiente:

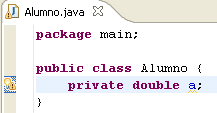
El primer desarrollador modifica la clase Alumno en su copia local:



Y la sube a SVN mediante un commit:

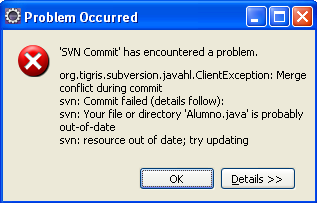


Al rato, el segundo desarrollador también modifica su copia local de la clase Alumno (cambia el tipo del atributo de int a double):



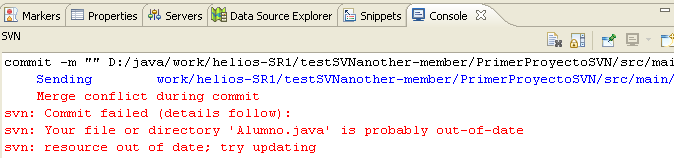
Notad que es exactamente la misma línea que modificó el primer desarrollador.

Si ahora hacemos un commit se producirá un error:



SVN ha detectado el problema: la clase Alumno que este segundo desarrollador ha modificado y pretende subir al repositorio pertenece a una versión más antigua que la existente en el repositorio (la que subió el primer desarrollador).

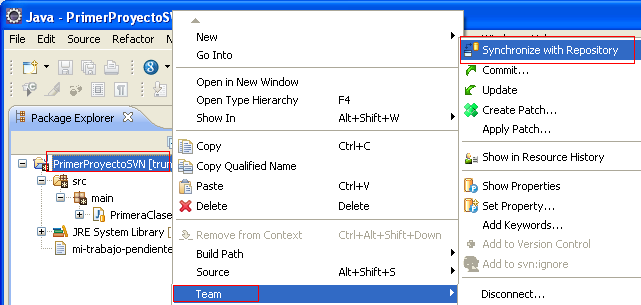
En la consola veremos:



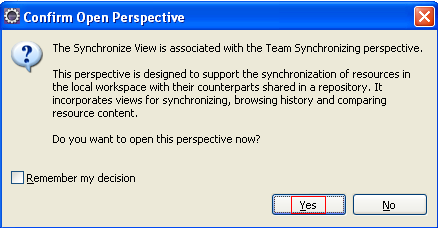
SVN nos pide que realicemos una sincronización con el repositorio central. Veamos qué es esto de la sincronización:

Habrá veces que tendremos que sincronizar nuestro proyecto local y el repositorio SVN, venga esto motivado por los cambios realizados por nosotros mismos entre la versión local del proyecto y el repositorio central o, sea a causa de cambios que otros miembros del equipo de desarrollo han realizado en SVN.

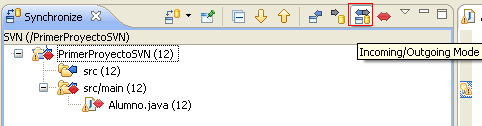
En cualquier caso, para sincronizar haremos: **‘Team->Synchonize with Repository’**:



Aparecerá una ventana informándonos de que esta operación requiere un cambio de perspectiva de Eclipse:



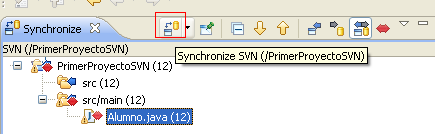
Aceptamos y veremos lo siguiente:



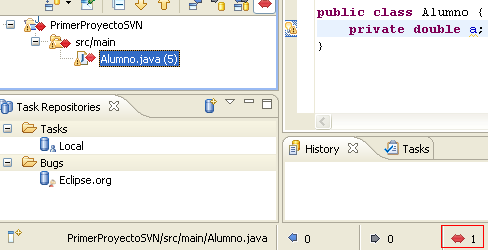
Notad que el modo predeterminado de esta perspectiva de trabajo es Incoming/Outgoing, esto implica que la sincronización entre el repositorio local y SVN será bidireccional y por lo tanto nuestra versión local asumirá los cambios de SVN y viceversa.

Cuando tenemos conflictos, cómo es el caso, entre cambios realizados por nosotros mismos u otros programadores sobre un mismo recurso, aparece un icono rojo en el proyecto. En seguida los resolveremos.

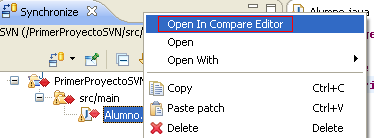
Veamos antes el icono para realizar una sincronización entre la copia local y la remota:



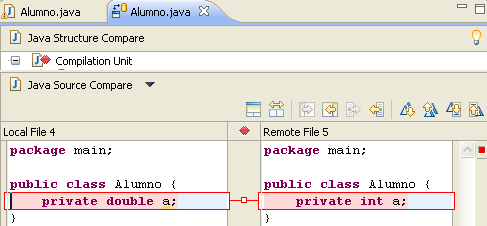
Bien, ocupémonos de los conflictos:



Para ver dónde está el problema podemos abrir la clase Alumno en un editor especial de comparación:

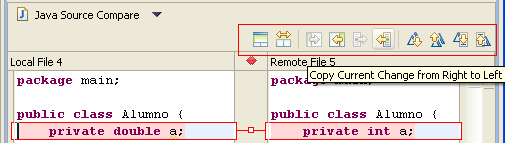


Con lo que obtendremos lo siguiente:

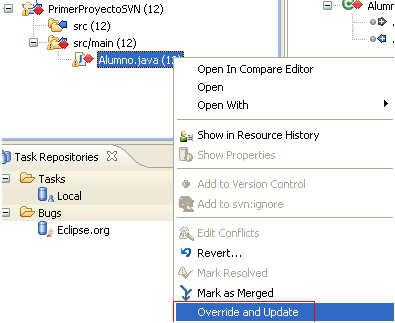


Notad que el código de la izquierda es el local y el de la derecha el de SVN.

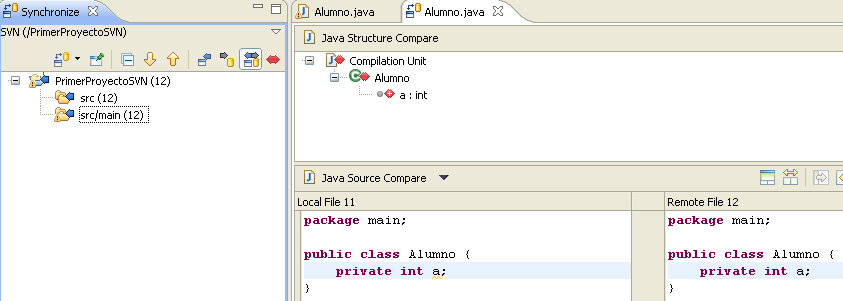
Con estos iconos podemos tanto consultar los puntos conflictivos como resolverlos, dando por bueno el local o el remoto:



Alternativamente, podemos resolver conflictos mediante menús desplegables. Por ejemplo si lo que queremos es sobrescribir los cambios locales por la versión remota podemos hacer lo siguiente:



Esta acción nos mostrará una advertencia informándonos que se perderán los cambios locales para esta clase. Si aceptamos obtendremos lo siguiente:



En cambio, si lo que queremos es que prevalezcan los cambios locales sobre los remotos podemos usar la opción ‘Mark as Merged’:

