Gestion de versions avec TortoiseSVN

Objectif du TP

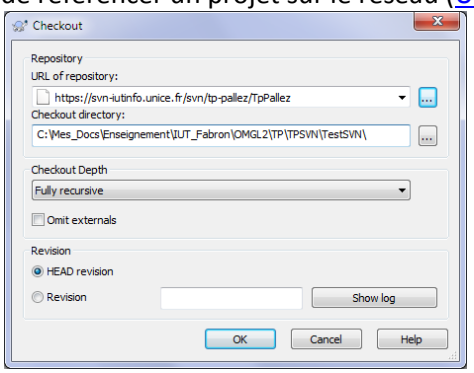
L'objectif de ce TP est de se familiariser avec SVN (Subversion) sous Windows avec `TortoiseSVN`. Le principe de cet outil est de faciliter le développement de fichiers sources (surtout pas exécutable ou binaire !) de manière collaborative, synchrone ou asynchrone. D'autres gestionnaires de versions existent (Mercurial ou Git - dépôt distribué) ; un comparatif est disponible ici.

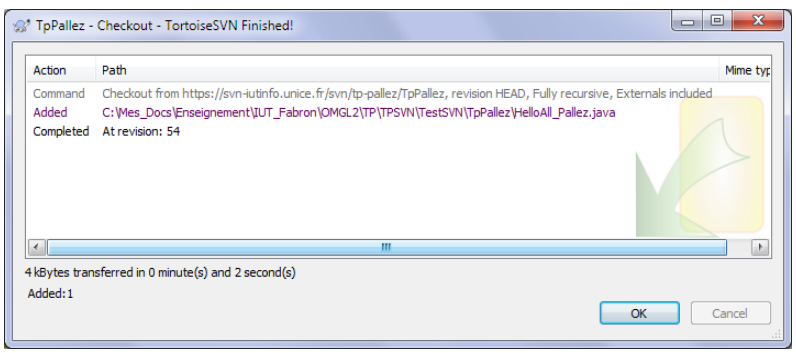
Création d'un dépôt

1. Créez un dossier sur votre espace personnel (par exemple `p:\repoSVN`). Avec le menu contextuel (bouton droit de la souris sur ce dossier), dites à `TortoiseSVN` que ce fichier est un dépôt (repository). Que constatez-vous dans ce dossier ? Lisez le fichier README.txt.
   Ce répertoire est considéré comme un dépôt, ainsi votre machine est considérée comme un serveur est DEVRAIT être constamment accessible par le groupe de développeurs par le réseau.
2. Comme votre machine n'est pas constamment connectée sur le réseau, le département Informatique de l'IUT de Nice-Côte d'Azur à mis à votre disposition un endroit pour héberger votre projet de développement (http://forge-iutinfo.unice.fr/). Lisez la page d'accueil et voyez comment déposer un projet sur cette forge. Pour utiliser cette forge, vous devez vous connecter. Une fois connecté, le manager du projet (l'enseignant), devra vous ajouter comme membre dans la configuration du projet. Des forges grand public existent : Sourceforge, Google Code…
3. Vous pouvez supprimer le répertoire créé en 1.

Utilisation d'un dépôt

4. Connectez-vous à la forge de l'iut à l'adresse suivante : `https://svn-iutinfo.unice.fr`. Ensuite, demandez à votre enseignant de vous ajouter au projet prévu pour ce TP : `groupeX` où X va de 1 à 5.
5. L'utilisation quotidienne d'un gestionnaire de versions se résume à quelques commandes accessibles par le menu contextuel de `TortoiseSVN`. Prenez le temps de lire le cycle de vie de travail de base. Nous allons détailler quelques unes de ces commandes. Créez un nouveau répertoire de travail (appelée `Working Copy` ou `WC`) sur votre espace personnel (par exemple `p:\TestSVN`). Référencez un dépôt existant en faisant un `checkout` d'un dépôt. Pour cela, faites un clic droit sur le dossier `TestSVN` créé, et utilisez le menu TortoiseSVN et utilisez l'URL `https://svn-iutinfo.unice.fr/svn/groupeX/TP` où X prendra une valeur entre 1 et 5 en fonction de votre groupe de TP. Si cela fonctionne, vous obtenez un résultat similaire à la fenêtre ci-dessus (à droite) et votre répertoire local de dépôt a changé d'aspect : TestSVN . Cela signifie que vous disposez en local de la dernière version du projet. Il y a plusieurs façons de référencer un projet sur le réseau (URL d'accès au dépôt).





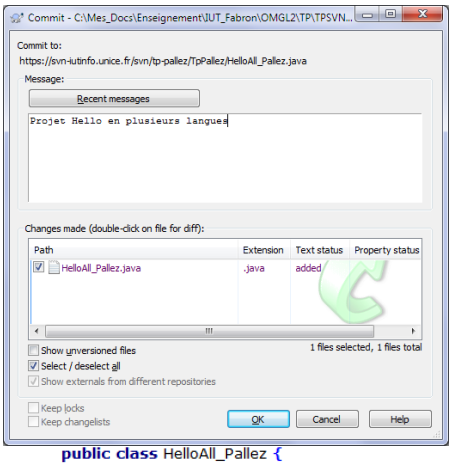
6. Votre projet consiste à dire bonjour en plusieurs langues. Comme vous êtes le manager de ce projet, vous saisissez le début de votre projet (programme) dans `TestSVN` (cf. ci-contre) en ajoutant <`votre nom`> à la classe. Ensuite, vous ajoutez ce code source dans le dépôt (menu contextuel `TortoiseSVN>Add...`). Que se passe-t-il ? Est-ce que votre fichier se trouve dans le dépôt ? Pour savoir exactement ce qu'il y a dans le dépôt, consultez-le à l'adresse suivante : https://forge-iutinfo.unice.fr/projects/groupeX/repository/show/TP.

```
public class HelloAll_VotreNom {
    public static void main(String[] args) {
    }
}
```

7. Pour que votre travail soit enregistré dans le dépôt, il faut absolument « valider » votre travail en faisant un « commit » (bouton droit sur le fichier `HelloAll_VotreNom.java`, option `SVN Commit…`). Vous devez saisir un message qui résume votre action de « validation ». Une nouvelle fenêtre apparait pour vous signaler ce qu'il s'est passé…
   Comment s'affiche votre fichier `HelloAll_VotreNom.java` dans votre répertoire ? Vérifiez que l'icône qui apparaît correspond bien à l'état de votre fichier ; vous trouverez ici la liste des icones utilisés dans TortoiseSVN.

Travailler de manière collaborative

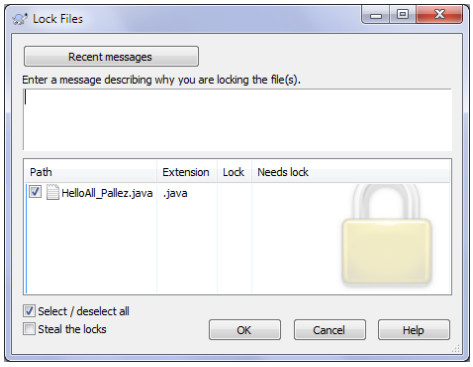
8. À partir de maintenant, il faut considérer que vous n'êtes plus le manager du projet de développement mais au contraire un simple programmeur qui a reçu une tâche bien précise de la part du manager. Demandez à l'enseignant responsable de votre groupe de TP de vous affecter une langue bien précise. Votre objectif, si vous l'acceptez, sera de modifier le fichier `HelloAll_Pallez.java` en programmant l'affichage de `Bonjour` dans la langue que le responsable vous a affecté en respectant les étapes suivantes :
   a) Faites un « `SVN Update` » pour être sur que vous allez travailler sur la dernière version du projet (vous pouvez faire un update sur n'importe quel fichier ou groupe de fichiers ou répertoires).
   b) Faites un « `Get Lock…` » pour verrouiller les fichiers sur lesquels vous allez travailler. Vous pouvez encore une fois saisir un message à destination des autres programmeurs pour expliquer l'objectif de votre verrouillage.
   c) Faites vos modifications. Comment apparaît(ssent) le(s) fichier(s) modifié(s) ?
   d) Faites un « `SVN Commit` » pour valider vos modifications. Avez-vous besoin de déverrouiller le fichier ?




```
public class HelloAll_Pallez {
    public static void main(String[] args) {
        String langue = args[0] ;
        if (langue=="FR")
            System.out.println("Bonjour") ;
        else if (langue=="US")
            System.out.println("Hello") ;
        else if (langue=="DE")
            System.out.println("Hallo") ;
        else if (langue=="IT")
            System.out.println("Bonjourno") ;
        else if (langue=="ES")
            System.out.println("Hola") ;
        else System.out.println("#!^¨£$%ù") ;
    }
}
```

Gestion des conflits

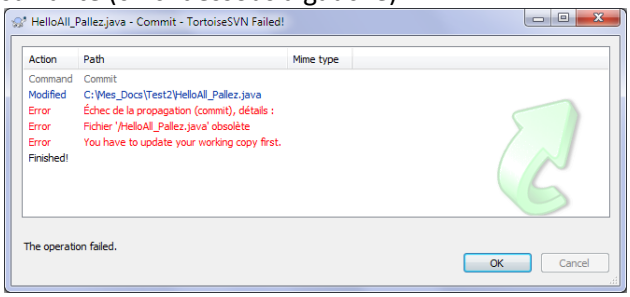
9. La question 8 correspond plutôt à une situation de travail asynchrone puisque vous bloquez l'accès à une ressource (un fichier) ; ainsi, pendant que vous travaillez, d'autres personnes ne peuvent pas y accéder. Néanmoins, dans certaines situations, il peut être indispensable que vous soyez plusieurs à travailler sur la même ressource (même fichier) mais pas forcément au même endroit dans le fichier. Cette situation correspond à un travail collaboratif synchrone. Mettez vous d'accord avec un autre binôme pour tester cette situation en modifiant simultanément `HelloAll_VotreNom.java` sans l'avoir verrouillé :

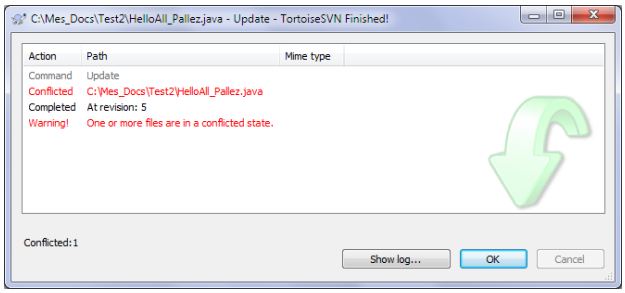


a) Le `binôme1` et le `binome2` font un update ;
b) Le `binôme1` modifie le fichier `HelloAll_VotreNom.java` ;
c) Le `binôme1` fait un commit ;
d) Le `binôme2` modifie le fichier `HelloAll_VotreNom.java` ;
e) Le `binôme2` fait un commit.

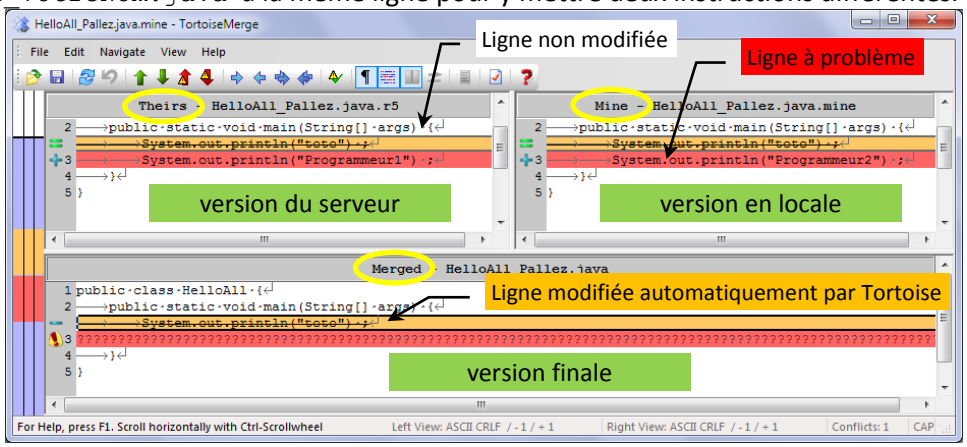
Remarque : Si par malheur un deuxième binôme n'est pas disponible, vous pouvez malgré tout simuler le fait que vous soyez deux programmeurs différents, en créant deux répertoires différents sur votre machine et en faisant un `checkout` sur le même dépôt. Ensuite, utilisez un répertoire pour simuler le travail du `binôme1` et l'autre répertoire pour simuler le travail du `binome2`.

10. Ainsi, lorsque le `binôme2` fait un commit, `TortoiseSVN` vous signale qu'il y a des conflits de la façon suivante (cf. ci-dessous à gauche) :

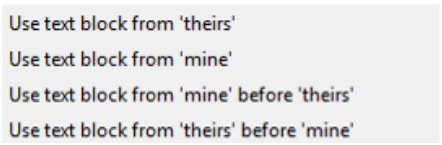




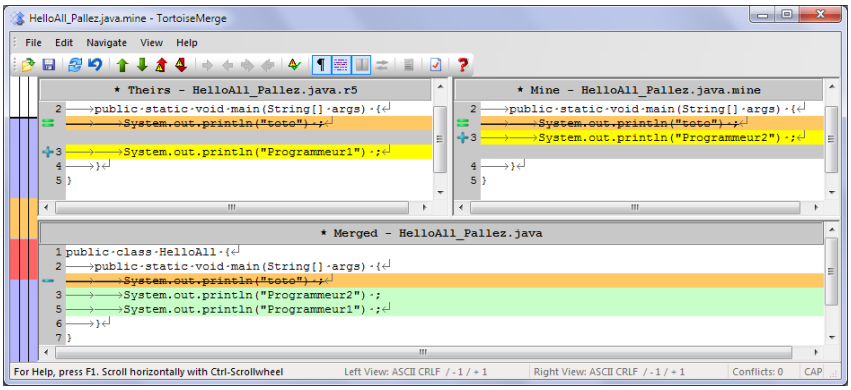
Pour résoudre le problème, `TortoiseSVN` vous demande de commencer par faire un update sur le fichier qui est en conflit. ***L'update ne modifiera pas vos dernières modifications*** ! Une fois l'update effectué, il se peut que les conflits soient résolus automatiquement ou bien que `TortoiseSVN` ne soit pas en mesure de régler automatiquement les conflits (cf. ci-dessus à droite). Afin que vous testiez cette nouvelle situation, répéter la *question 9* en modifiant simultanément le fichier `HelloAll_VotreNom.java` à la même ligne pour y mettre deux instructions différentes.



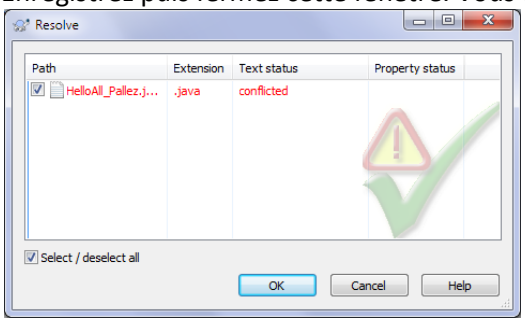
11. Pour résoudre la situation présentée précédemment, il faut que vous gériez manuellement les conflits en utilisant le menu « `Edit Conflicts…` » sur le ou les fichiers en conflit. L'écran ci-dessus apparaît :
`TortoiseSVN` vous présente alors l'outil `TortoiseMerge` qui permet de comparer des fichiers. Il va vous falloir comparer le fichier contenu *officiellement* dans le dépôt sur le serveur (en haut à gauche) nommé `Theirs`, le fichier tel qu'il est sur votre machine (en haut à droite) nommé `Mine`, et le fichier que vous allez réellement envoyer sur le serveur contenant les conflits résolus (en bas) nommé `Merged`.
Pour résoudre les conflits manuellement, vous devez alors vous positionnez sur la ou les lignes qui posent soucis dans la version finale et cliquez avec le bouton droit pour choisir ce que vous souhaitez finalement faire (cf. ci-contre).

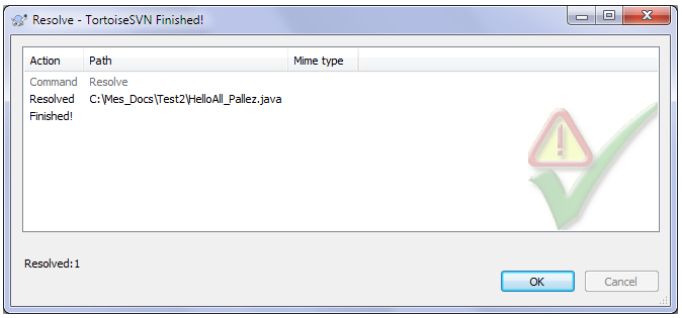


Une fois les conflits résolus, vous obtenez un écran similaire à celui-ci :



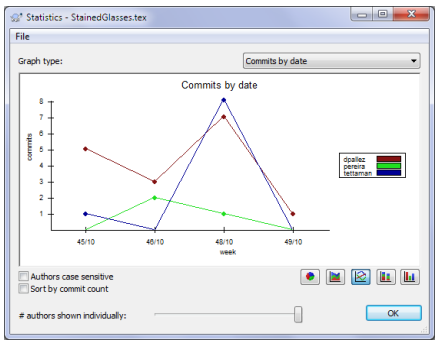
Enregistrez puis fermez cette fenêtre. Vous obtenez les fenêtres suivantes :





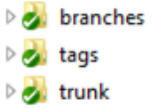
Vérifiez l'état de votre fichier ! Que vous reste-t-il à faire ? Faites-le !!!

12. Dans la question 8 se trouve le code source du projet final (5 langues différentes sont gérées). Plutôt que chaque binôme implémente une version *complète* de ce projet, partagez vous le travail en divisant le groupe de TP en 5 binômes de telle sorte que chaque binôme s'occupe d'implémenter une et une seule langue (votre enseignant définira les groupes). Vous verrez de cette façon l'intérêt de travailler à plusieurs même de manière synchrone. La ressource à partager sera appelée `HelloAll.java`.
13. Sur votre WC (`Working Copy`), faites un `switch` sur la révision n°1. Que constatez-vous ? À quoi correspond cette opération ?
14. Pour revenir à la toute dernière version, il suffit simplement de faire un `switch` sur la `HEAD` revision. En réalité, vous venez de référencer deux versions développées à des moments différents du même projet.
15. Les participants au projet possèdent un moyen de savoir qui a travaillé sur quelle ressource. Pour cela, faites un update du répertoire de votre projet (`TestSVN`) et utilisez le bouton « `Show log` » et vous verrez qui a effectué quelle révision et pourquoi si des commentaires ont été ajoutés. Ensuite, cliquez sur le bouton « `Statistics` » pour avoir un résumé du travail effectué par le groupe. Un écran similaire à celui ci-contre vous sera affiché. Vous pouvez changer les informations affichées sur le graphique.



Gestion de versions

16. Comme vous le savez surement, un bon logiciel doit évoluer dans le temps pour toujours satisfaire au mieux les besoins des utilisateurs. Le génie logiciel suggère de développer les projets par étapes. Ainsi, une *version* correspond à un ensemble de codes sources permettant de générer une version *stable* (ou encore version *béta*) utilisable par le client. Le principe consiste donc à marquer, étiqueter et sauvegarder un moment de l'historique de développement du logiciel. Par conséquent, un dépôt est communément structuré de la manière suivante par analogie à la notion d'arbre :
    a) Un répertoire correspondant à la dernière version en cours de développement du logiciel (« `trunk` » en anglais, *tronc* en français) ;
    b) Un répertoire stockant les différentes versions *stables* ou étiquettes (« `tag` » en anglais) qui ont été enregistrées pendant tout le développement du logiciel ;

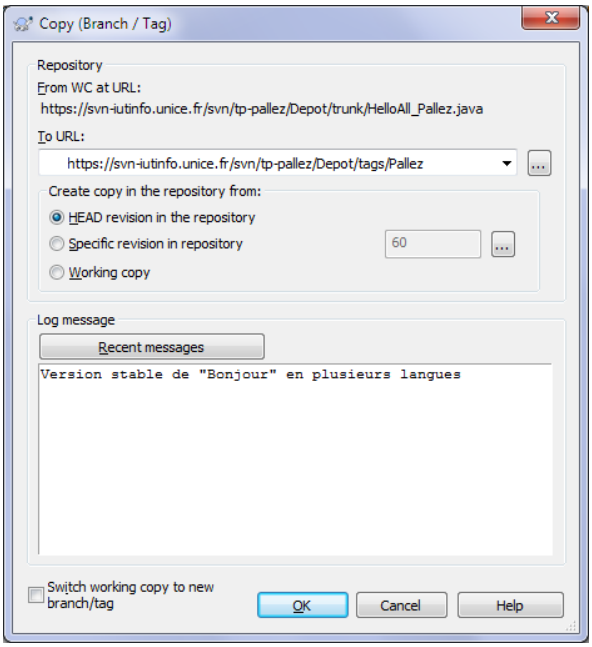


c) Un répertoire contenant toutes les versions simultanées (« `branches` » en anglais) en cours de développement (cf. question 18 pour en savoir plus).

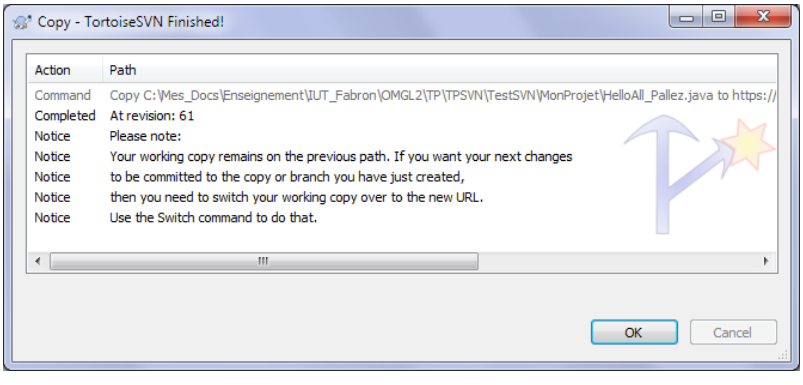
Ainsi, au fur et à mesure du développement du logiciel, le tronc (`trunk`) contiendra toujours la dernière version en cours de développement. Dès que cette version correspond à une version stable, elle peut être sauvegardée et taguée dans `tags`. Le logiciel va encore évoluer (suppression de bogues) donc le tronc va évoluer (grandir) pour aboutir sur une nouvelle version stable qui pourra elle aussi être taguée à nouveau. Il est clair que les développeurs ne devront pas référencer toutes les versions contenues sur le dépôt (versions stables et dernière version) mais faire référence à une seule version à la fois sur leur machine en local appelée `Working Copy (WC)` dans `TortoiseSVN`. L'intérêt de taguer les versions consistent à pouvoir naviguer et donc référencer et exécuter les différentes versions contenues dans le dépôt. Cette navigation se fait grâce à la commande `switch` vue aux questions 13 et 14.

Sur votre espace personnel, créez un nouveau répertoire `Projet` qui fait référence au **tronc** du répertoire `Depot` de votre projet (cf. question 4) `https://svn-iutinfo.unice.fr/svn/groupeX/Depot/trunk`.

17. Comme dans la question 12, vous avez fini une version stable et exécutable du projet qui consiste à dire bonjour en plusieurs langues, copiez le fichier `HelloAll.java` dans votre `WC` (`Projet`) et « taguez » cette copie de travail en utilisant le menu `Branch/Tag` :



   a) Dans le champ « `To URL` », il faut préciser le répertoire du dépôt qui correspond aux versions sauvegardées (« `tags` ») suivi d'un nom d'un répertoire correspond au nom de cette version (utilisez votre nom comme nom de version, cf. ci-contre).
   b) Ensuite, il est nécessaire de préciser ce qu'on sauvegarde :
      i. Soit la dernière version du `trunk`, et dans ce cas, il faut cocher le bouton `HEAD revision in the repository` ;
      ii. Soit une révision spécifique et dans ce cas, il faut cocher `Specific revision in repository` et choisir le numéro de révision. Ce choix est équivalent au précédent si le numéro choisi est le numéro de la dernière révision (c'est l'option sélectionnée par défaut) ;
      iii. Soit le répertoire de travail en local qui n'est pas forcément mis à jour sur le dépôt et dans ce cas, il faut choisir `Working copy`.



Le résultat de cette opération est représenté dans la figure ci-contre.

Gestion simultanée de plusieurs versions d'un même projet (branches)

18. Un de vos collaborateurs vous signale une manière plus intelligente de gérer la localité (cf. ci-contre). En bon manager, vous ne souhaitez pas perdre la version actuelle qui fonctionne et qui est en cours d'utilisation par les clients mais qui est également en cours de modification par un autre groupe de développeurs qui est en train d'ajouter d'autres langues. Cependant, vous souhaitez quand même tester cette nouvelle idée. Pour cela, vous créez une branche à votre développement logiciel en utilisant le même menu que précédemment. Une branche est équivalente à une étiquette, néanmoins, une branche est principalement utilisée pour ajouter et mettre en œuvre de nouvelles fonctionnalités à la version courante alors qu'une étiquette est utilisée pour conserver une version stable. Une fois la fonctionnalité développée, testée, validée, `TortoiseSVN` permet de ramener une branche (`branch`) sur le tronc (`trunk`).

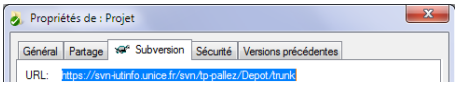
HelloAll_Pallez.java

```
import java.util.Locale ;
import java.util.ResourceBundle ;
public class HelloAll {
    public static void main(String[] args) {
        Locale local_current = Locale.getDefault() ;
        ResourceBundle myRessources = ResourceBundle.getBundle("Resources", local_current);
        System.out.println (myRessources.getString("MsgHello")) ;
    }
}
```

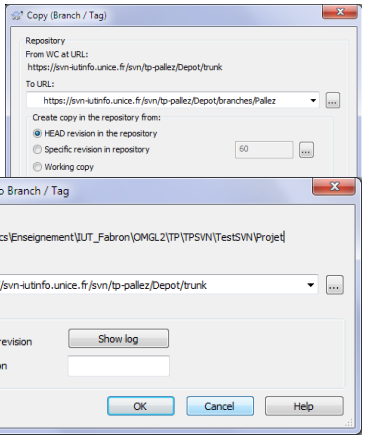
Resources_fr_FR.properties

```
MsgHello = Bonjour
MsgBye = Au revoir
```

   a) Faites un update de votre `WC` qui référence normalement le tronc du projet sur la forge de l'IUT. Pour en être sûr, affichez les propriétés du répertoire projet et vérifiez que l'URL corresponde avec l'URL de la figure ci-contre.

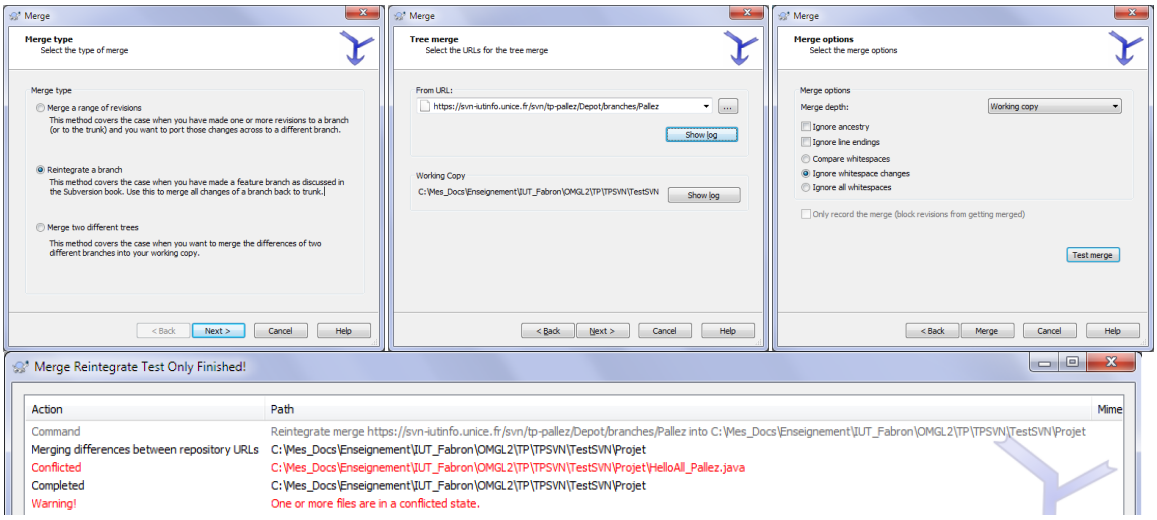


   b) Sur ce même répertoire (`Projet`), créez une nouvelle branche en choisissant le répertoire `branches` du dépôt suivi d'un répertoire correspondant à votre nom (cf. ci-contre : `Depot/branches/ Pallez`) ; choisissez bien la version `HEAD` ou `WC` (elles sont identiques à priori puisque vous avez fait un update en 18.a) et cochez la case « `switch working copy …` » pour qu'à partir de maintenant, chaque fois que vous ferez un update, *seule* la branche que vous venez de créer soit mis à jour au lieu du tronc.
   Le résultat de cette opération se trouve dans la figure ci-dessous :



   c) Comme précédemment, mettez vous d'accord pour que chaque binôme mette en œuvre une langue en particulier en utilisant le principe des classes `Locale` et `RessourceBundle` de java.

19. Une fois les développements des branches terminés, il est nécessaire de « recoller » la branche sur le tronc. Il faut, pour cela, fusionner la branche et le tronc en local et ensuite mettre à jour le dépôt sur le serveur :
   a) Faites un commit sur votre branche créée dans la question 18 (`Depot/branches/Pallez`) pour être sûr que le dépôt contienne la dernière version de votre branche ;
   b) Comme vous souhaitez réintégrer la branche au tronc, il faut d'abord référencer le tronc ; pour cela utiliser l'option « `Switch` » de `TortoiseSVN` en utilisant l'URL du tronc (cf. ci-contre) ;
   c) Sélectionner le dossier `Projet` dans votre `WC` et utilisez le menu « `Merge` » de `TortoiseSVN` afin de fusionner le tronc (qui a peut être évolué entre temps) et votre branche. Au 3ème écran (cf. ci-dessous), il est fortement conseillé de tester la fusion avec le bouton « `Test Merge` » avant de l'appliquer réellement :



d) Comme vous avez fusionné deux projets qui ont évolué simultanément, il y a de fortes probabilités pour que soyez obligé de gérer manuellement les conflits (cf. question 10). Sachez que la fusion de projets (`merge`) est peu utilisée car elle demande souvent de gérer manuellement beaucoup de conflits (surtout si le projet contenu dans le tronc est conséquent).

Pour aller plus loin

20. Les différentes manipulations faites dans ce TP peuvent être réalisées en ligne de commande. Ouvrez une invite de commandes Windows en exécutant `cmd` et utilisez la commande `svn`. Refaites l'ensemble du TP uniquement en ligne de commandes.

Sources utilisées pour réaliser ce TP

http://kevin.fardel.perso.esil.univmed.fr/documentation/TutorielTortoiseSVN.pdf
http://tortoisesvn.net/docs/nightly/TortoiseSVN_fr/
http://svnbook.red-bean.com/nightly/fr/