Développer une application Informatique en Java

Cette formation vous permettra de maîtriser la programmation objet en Java, la programmation concurrente et les outils de base du développement. Elle constitue le premier élément du parcours « formation de base Java EE ».

    - Pattern
    - Annotation et Pattern
    - JDBC et Cryptographie
    - Batch
    - Etudiant
    - Salaire
    - Ville
    - Media
    - Aeroport
    - Aeroport2

- SVN
- Slide

Introduction

Cette formation vous permettra de maîtriser la programmation objet en Java, la programmation concurrente et les outils de base du développement. Elle constitue le premier élément du parcours « formation de base Java EE ».

Plan du cour

Slide

1) introduction *(environ 1/2 de journée)*

- Qu'est-ce que Java, une JVM

- Le framework Java EE, les différents serveurs Applicatif J2EE ?

- Qu'est-ce qu'Eclipse ? Comment le configurer ?

- Le debuggage

Travaux pratiques

Installation du JDK, gestion de la variable d'environnement classpath. Installation puis utilisation de l'IDE Eclipse en perspective Java, utilisation de Eclipse en mode debug.

2) Les constructions de base du langage *(environ 3/4 de journée)*

- Les variables : déclaration et typage.

- Les méthodes : définition.

- Les expressions.

- Les instructions de contrôle : les instructions conditionnelles, de boucle, de branchement.

- Les tableaux.

- Les unités de compilation et packages : le contrôle de la visibilité des classes, le mécanisme d'import.

- Les imports statiques.

Travaux pratiques

Suite d’exercices simples permettant la prise en main de l’environnement de développement (Eclipse en perspective Java), notamment la documentation et la réalisation d’un programme simple. Utilisation des packages.

3) Tests logiciels *(environ 3/4 de journée)*

- Pourquoi faire des tests ?

- Présentation des différents types de tests : tests unitaires, fonctionnels, de robustesse et de performance.

- Quels tests lancer et quand ?

- Utilité des objets "Mock" et "Fake" durant les tests unitaires. Couverture des tests unitaires.

Travaux pratiques

Installer JUnit sous Eclipse. Ajouter des tests unitaires avec JUnit sur les projets Eclipse déjà écrits.

4) Bonnes pratiques de conception d'une application *(environ 1/2 de journée)*

- Découpage en couches (données, métier, présentation).

- Présentation des enjeux d'un développement d'entreprise.

- Introduction à l'écosystème JAVA (JEE, Spring, Hibernate, JSF ou Struts...).

Travaux pratiques

Echanges quant au choix technique du framework Java EE Web Profile. Réflexion sur la conception en couche.

5) Les techniques Objet *(environ 1/2 journée)*

- Les principes généraux de la modélisation et de la programmation Objet.

- L'abstraction et l'encapsulation : les interfaces.

- Les différentes formes d'héritage, le polymorphisme.

- Introduction à la modélisation UML

Travaux pratiques

La notion d'identité simple seras étendue dans l'application de registre d'identité afin de faire apparaitre des particularités.

6) La gestion des versions : introduction à SVN *(environ 3/4 de journée)*

- Les concepts généraux liés à la gestion de versions.

- Les concepts SVN : dépôt, projets, révisions, tronc, branches et tags.

- Les principales opérations offertes au développeur. La gestion des conflits.

- La gestion des branches. Les perspectives SVN proposées par les plug-ins Eclipse.

Travaux pratiques

Installation d'une solution de versioning, installation du bon plug-in dans Eclipse. Création de projets associés à un repository. Gestion des versions de l'application développée – récupération d'une copie locale, modifications, fusion, commit.

7) Définition de la structure d'un projet avec Maven *(environ 3/4 de journée)*

- Définition de la structure d'un projet.

- Les conventions. Les dépendances entre projets. Les tâches prédéfinies : compilation, génération d'archives...

- Les perspectives Maven proposées par les plug-ins Eclipse.

Travaux pratiques

Utilisation de Maven sous Eclipse pour automatiser une succession de traitements notamment de test.

8) Quelques Design Pattern *(environ 1/4 de journée)*

- Les objectifs et les avantages.

- Les Design Patterns les plus populaires dans les architectures logicielles modernes.

9) mesures de la qualité *(environ 1/4 de journée)*

- Synthèse des mesures qualité.

- La convention de codage et la documentation.

- La couverture de tests et l'automatisation des procédures.

- Mise en place d'un tableau de bord de la qualité.

Fil Rouge

Contexte

L’ETNIC possède un site web dédié aux offres d’emploi (MonJob @ Etnic).

Ce site se sépare en trois parties front-end, backend-utilisateur, backend-administrateur.

La partie front end permet :

- De savoir ce qu’est MonJob,
- D’avoir un mail/téléphone de contact à l’ETNIC,
- D’avoir accès à une liste de questions les plus souvent posées,
- De parcourir les offres d’emploi disponibles sous forme de liste (description rapide des postes) et d’avoir par poste une fiche détaillée,
- De se connecter au backend-utilisateur ou se créer un compte.

La partie backend utilisateur permet

- De créer un CV composé de données personnels (date de naissance, lieu, sexe..), de ses certificats, de ses diplômes, de ses expériences,
- De parcourir les offres d’emploi disponibles et de postuler à ces offres.

La partie backend administrateur permet

- De se connecter,
- De créer une offre d’emploi,
- De voir les candidatures ayant postulé à une offre,
- D’accepter ou de refuser une offre.

Risques

Cette application n’est pas une application critique (elle n’empêche pas l’ETNIC de fonctionner en cas d’arrêt), mais c’est un cas typique d’application dont la qualité doit être élevée car :

- Les bugs divers de l’application donnent une mauvaise image de l’ETNIC,

- Cette application contient des données personnelles de candidate. Un bug sur l'application peut entraîner des problèmes législatifs (RGPD).

Il est donc essentiel pendant la formation et ce fil rouge de faire les tests suffisants pour éviter tous problèmes

Parcours Utilisateurs

Il y a trois parcours utilisateurs :

- Le premier parcours est celui du curieux. La personne curieuse tombe par hasard sur le site MonJob. Il va regarder rapidement les offres d'emplois. Enfin, il quitte le site.
- Le parcours du postulant est différent. Il doit en premier se créer un compte en donnant un login et un password. Enfin, il reçoit un email validant son inscription. Après la finalisation de son inscription, il remplit son CV et postule à une offre. Enfin, il se déconnecte de MonJob et attend avec ou sans angoisse de passer à l'étape 2 du processus de recrutement.
- Le parcours administrateur est singulier. L'administrateur se connecte sur MonJob soit pour créer un Job soit pour voir les personnes ayant postulé. Il est au courant qu'une personne a postulé à un Job car il reçoit un email à chaque candidature. Sur les postulants, l'administrateur peut accepter la candidature (un mail heureux part vers le candidat et un autre part vers le responsable de l'offre d'emploi) soit la refuser (un mail malheureux part vers le candidat).

Réalisation

Le but de ce fil rouge n'est pas de refaire MonJob en entier dans le cadre de la formation.

Ci-après les objectifs plus ou moins atteignables lors de la formation :

Objectif : Niveau 1 je pratique le Java/J2EE et je suis à l'aise

Être à l'aise avec Java/J2EE est de pouvoir faire quelques classes, construire une application Web et mettre en place les éléments de qualité logiciel me garantissant que cela fonctionne.

Je dois faire l’application suivante :

- Une page Web permettant de créer un Jobs pour l'administrateur
- Une page Web permettant de visualisez les Jobs sous forme de table et sous forme détaillé
- Une page Web d’info utile.

Ce que je dois comprendre est :

- Faire des classes simples en Java et faire des pages web simples pour les gérer,
- **Comment tester cette application, Identifier les points durs en termes de qualité logiciel. Je dois savoir faire les tests unitaires de ces classes, d’en faire les tests graphiques.**
- **Ma couverture de code de mes tests est de 60%.**

Objectif : Niveau 2 je connais le Java/J2EE et je suis plus qu’à l’aise

Je suis assez à l’aise pour monter une architecture plus complexe et en faire une application Web. Je comprends que la mise en place des tests est une nécessité pour maintenir mon architecture Web stable.

Je dois faire l’application suivante :

- Une page Web permettant de créer des Jobs.
- Une page Web permettant de créer un Job pour l’administrateur.
- Une page Web d’info utile.
- La création d’un compte candidat (avec login/password et gestion du mail d’activation).
- Le candidat peut remplir quelques données personnelles simples
- Le candidat peut postuler à un Job.

Ce que je dois comprendre :

- **Comment monter une architecture complexe de classe Java et de mettre en place un début d’authentification utilisateur.**
- **Je sais monter, tout au long de mon travail, les tests unitaires qui garantissent mes développements futurs.**
- **Je monte des tests fonctionnels depuis l’interface graphique en bouchonnant ou pas mes éléments.**
- **Ma couverture de code de mes tests est de 70%**

Objectif : Niveau 3 Le Java/J2EE n’est pas le problème ici

Je pense déjà au futur de cette application. J’ai compris comment monter une architecture très complexe, stable dans le temps et suffisamment performante pour accueillir plein de candidats.

Je dois faire l’application suivante :

- Une page Web permettant de créer des Jobs.
- Une page Web permettant, à l’administrateur, de créer un Job et d’associer un responsable du Job.
- Une page Web d’info utile.
- La création d’un compte candidat (avec login/password et gestion du mail d’activation).
- Le candidat peut remplir toutes données personnelles.
- Il peut remplir ses CV, ses certificats et ses expériences . Les opérations suivantes sont possibles :

A minima, il est possible de préciser un titre au CV, donner un texte libre et uploader du CV. Le format de l’upload est du PDF d’une taille maximal de 200ko.

- Le candidat peut postuler à un Job.
- L’administrateur visualise pour chaque Job et postulant. Cela lui permet d’accepter une candidature ou de la refuser.

Je comprends largement comment faire cette application. J’ai monté les tests unitaires et les tests fonctionnels. Ma couverture de code est bonne (70%), mes tests fonctionnels sont complets et je me prépare à monter des tests de performances et des tests de sécurité.

Objet du système

Nous donnons ici les caractéristiques des objets du système

Un Job est défini par :

- Niveau 1,2, 3 :D’un titre, d’un résumé, d’une date limite de candidature et d’une description détaillée
- Niveau 2,3 : D’une liste de candidat

Un Candidat est défini par :

- Niveau 2,3 d’un login/password/email et de quelques données personnels (nom, prénom, adresse, nationalité)
- Niveau 3 : de diplôme (nationalité du diplôme, titre, niveau), de certificat (description) et de compétence (description).

Fil Rouge Civadis

Gestion Simplifié de recrutement

Contexte

Le but de l'application est de fournir à Civadis un logiciel de gestion simplifié du recrutement.

Ce site se sépare en trois parties front-end, backend-utilisateur, backend-administrateur.

La partie front end permet :

- De pouvoir positionner une Vitrine
- De parcourir les offres d'emploi disponibles sous forme de liste (description rapide des postes) et d'avoir par poste une fiche détaillée,
- De se connecter au backend-utilisateur via un email/password ou se créer un compte.
- En fonction de l'email, l'utilisateur à accès aux fonctionnalités RH (backend Civadis) ou pas (Backend Invité).

La partie Backend Invité permet

- De créer un CV composé de données personnels (date de naissance, lieu, sexe..), de ses certificats, de ses diplômes, de ses expériences,
- De parcourir les offres d'emploi disponibles et de postuler à ces offres.

La partie backend administrateur permet

- De se connecter,
- De créer une offre d'emploi,

- De voir les candidatures ayant postulé à une offre,
- De sélectionner les candidatures

Risques

Cette application n'est pas une application critique (elle n'empêche pas Civadis de fonctionner en cas d'arrêt), mais c'est un cas typique d'application dont la qualité doit être élevée car :

- Les bugs divers de l'application donnent une mauvaise image de Civadis,

- Cette application contient des données personnelles de candidate. Un bug sur l'application peut entraîner des problèmes législatifs (RGPD).

Il est donc essentiel pendant la formation et ce fil rouge de faire les tests suffisants pour éviter tous problèmes

Parcours Utilisateurs

Il y a trois parcours utilisateurs :

- Le premier parcours est celui du curieux. La personne curieuse tombe par hasard sur le site de recrutement. Il va regarder rapidement les offres d'emplois. Enfin, il quitte le site.

- Le parcours du postulant est différent. Il doit en premier se créer un compte en donnant un login et un password. Enfin, il reçoit un email validant son inscription. Après la finalisation de son inscription, il remplit son CV et postule à une offre. Enfin, il se déconnecte de l'applicatif et attend avec ou sans angoisse de passer aux étapes ultérieures du processus de recrutement. Il est informé du passage des étapes (épreuves 1, épreuves 2, décision) par courriel.
- Le parcours administrateur est singulier. L'administrateur se connecte sur l'applicatif soit pour créer un Job soit pour voir les personnes ayant postulé ou ceux en épreuves. Il est au courant qu'une personne a postulé à un Job car il reçoit un email à chaque candidature. Sur les postulants, l'administrateur peut accepter la candidature (un mail heureux part vers le candidat et un autre part vers le responsable de l'offre d'emploi) soit la refuser (un mail malheureux part vers le candidat). Dans le cas où la candidature est acceptée, cette dernière positionne le candidat en mode passage d'épreuves. A chaque épreuve réussie, le RH peut modifier l'état du candidat, ce qui provoque une notification.

Réalisation

Le but de ce fil rouge n'est pas de refaire l'applicatif de recrutement en entier dans le cadre de la formation.

Ci-après les objectifs plus ou moins atteignables lors de la formation :

Objectif : Niveau 1 je pratique le Java et je suis à l'aise

Être à l'aise avec Java est de pouvoir faire quelques classes, construire une application Web et mettre en place les éléments de qualité logiciel me garantissant que cela fonctionne.

Je dois faire l'application suivante :

- Une API REST permettant de lister des Jobs en liste en fonction d'un statut (En cours d'élaboration, Validée, publiée)
- Une API REST pour afficher un job en détail.

- Une API REST pour modifier un job
- Une API REST permettant de créer un Job pour l'administrateur.

Afin de pouvoir créer une offre, il est nécessaire de pouvoir encoder pour créer une offre (A titre d'exemple) :

- Numéro de l'offre
- Titre de l'offre

- Statut : (En cours d'élaboration, Validée, publiée)
- Type engagement
- Lieu travail
- Régime de travail
- Diplôme requis

- Vos responsabilités
- Votre profil
- Notre offre

En termes d'architecture, il est proposé mais non imposé l'approche suivante :

- Faire un cœur applicatif sans authentification ni base de données

- Rajouter la base de données et le mode REST sur le cœur applicatif
- Rajouter l'authentification.

Ce que je dois comprendre est :

- Faire des classes simples en Java et faire des pages web simples pour les gérer,
- **Comment tester cette application, Identifier les points durs en termes de qualité logiciel. Je dois savoir faire les tests unitaires de ces classes, d'en faire les tests graphiques.**

- **Ma couverture de code de mes tests est de 60%.**

Objectif : Niveau 2 je connais le Java et je suis plus qu'à l'aise

Je suis assez à l'aise pour monter une architecture plus complexe et en faire une application Web. Je comprends que la mise en place des tests est une nécessité pour maintenir mon architecture Web stable.

Je dois faire l'application suivante :

- Une API REST permettant de lister des Jobs en liste en fonction d'un statut (En cours d'élaboration, Validée, publiée)

- Une API REST pour afficher un job en détail.
- Une API REST pour modifier un job
- Une API REST permettant de créer un Job pour l'administrateur.

- Une API REST permettant d’initialiser le compte candidat (avec login/password et gestion du mail d’activation).
- Une API REST permettant de créer un candidat.

- Une modification de l’API REST d’un Job afin d’affilier un candidat à un Job publié.

La candidature d’une personne peut contenir à titre d’exemple :

- Données personnelles

- Nom

- Prénom
- Sexe
- Date de naissance
- Nationalité
- Langue

- Email
- Numéro téléphone

- Adresse

- Rue
- Numéro

- Code postal
- Ville
- Pays

- Connaissance

- Diplôme

- Langue

- Français (pas de connaissance, Notions, Bonne Très bonne, langue maternelle)
- Néerlandais (pas de connaissance, Notions, Bonne Très bonne, langue maternelle)
- Anglais (pas de connaissance, Notions, Bonne Très bonne, langue maternelle)

- Informations complémentaires

- CV
- Motivation
- URL profil linkedin

En termes d'architecture, il est proposé mais non imposé l'approche suivante :

- Faire un cœur applicatif sans authentification ni base de données
- Rajouter la base de données et le mode REST sur le cœur applicatif
- Rajouter l'authentification.

Afin de mettre en place la création d'utilisateur qui nécessite une validation d'email, il est possible d'utiliser les composants suivants :

- Validation de utilisateurs par email : https://www.baeldung.com/registration-verify-user-by-email
- Sauvegarder des informations semi structurée en base de données : https://www.baeldung.com/hibernate-persist-json-object
- Un serveur smtp bouchonné en userland : https://github.com/maildev/maildev

Ce que je dois comprendre :

- **Comment monter une architecture complexe de classe Java et de mettre en place un début d'authentification utilisateur.**
- **Je sais monter, tout au long de mon travail, les tests unitaires qui garantissent mes développements futurs.**
- **Je monte des tests fonctionnels depuis les API REST en bouchonnant ou pas mes éléments.**
- **Ma couverture de code de mes tests est de 70%**

Objectif : Niveau 3 Le Java n'est pas le problème ici

Je pense déjà au futur de cette application. J'ai compris comment monter une architecture très complexe, stable dans le temps et suffisamment performante pour accueillir plein de candidats.

Je dois faire l'application suivante :

- Une API REST permettant de lister des Jobs en liste en fonction d'un statut (En cours d'élaboration, Validée, publiée, supprimé)
- Une API REST pour afficher un job en détail.
- Une API REST pour modifier un job. La modification pour cette partie consiste principalement a rajouter un email de contact pour le Job.
- Une API REST permettant de créer un Job pour l'administrateur.
- Une API REST permettant d'initialiser le compte candidat (avec login/password et gestion du mail d'activation).

- Une API REST permettant de créer un candidat.
- Une modification de l'API REST d'un Job afin d'affilier un candidat à un Job publié.
- Une API REST est disponible pour pouvoir pousser des pièces jointes (certificats, diplôme, CV …) sur un candidat.
- Une API REST pour modifier l'état d'un candidat au regard de ses épreuves.

Pour chacune des offres les candidats doivent passer deux épreuves, une première épreuve écrite et dans le cas où l'épreuve écrite est réussie, une épreuve orale.

L'application doit permettre au gestionnaire RH de sélectionner à chaque étape les candidats retenus pour l'étape suivante. A chaque passage d'une étape à l'autre, les candidats non retenus doivent recevoir un mail et les autres candidats reçoivent un autre mail pour les inviter à l'étape suivante. Enfin pour chaque candidat avançant dans le process, il est possible d'envoyer un courriel à la personne de contact.

Lorsqu'un candidat est retenu, le Job est dépublié automatiquement.

Afin de mettre en place la fonctionnalité d'Upload de pièce d'un utilisateur, il est possible de s'inspirer de :

- Uploading Files : https://spring.io/guides/gs/uploading-files/

Je comprends largement comment faire cette application. J'ai monté les tests unitaires et les tests fonctionnels. Ma couverture de code est bonne (70%), mes tests fonctionnels sont complets et je me prépare à monter des tests de performances et des tests de sécurité de base.

Objectif : Je pratique de l'Angular

Cet objectif est transverse aux objectifs Java. Le but ici est de monter un applicatif Angular avec PrimeNG afin :

- Permettre de lister des Jobs en liste en fonction d'un statut (En cours d'élaboration, Validée, publiée, supprimé)
- Permettre pour afficher un job en détail.
- Permettre de créer/modifier un Job pour l'administrateur.
- Permettre d'initialiser le compte candidat (avec login/password et gestion du mail d'activation).

- Permettre de créer un candidat.
- Permettre d’affilier un candidat à un Job publié.

De façon non exhaustive et non obligatoire, il est possible de s’inspirer des sketch suivants fortement inspiré de :

https://www.primefaces.org/primeng/showcase/#/dataview

Une deuxième page permet d’afficher un job en détail :

En fonction de l’état de l’utilisateur (connecté ou pas), il est possible de postuler à l’offre.

L’interface backend RH permet pour chaque Job d’avoir la liste des candidats d’un job.

Structures Java

Les Variables

Exercice 1 Les noms de variables

Quel est sont les nom de variables correctes:

- fonction-1
- _MOYENNE_du_MOIS_
- 3e_jour
- limite_inf.
- lim_supérieure
- __A_
- _
- a
- 3

Exercice 2 circulation de données

Ecrire un programme qui permute et affiche les valeurs de trois variables A, B, C de type entier qui sont entrées au clavier :
A ==> B , B ==> C , C ==> A

Correction:

```
public static void main(String [] args)
{
  String A=args[0];
  String B=args[1];
  String C=args[2];
  System.out.println(B);
  System.out.println(C);
  System.out.println(A);
}
```

Exercice 3: Les résistances

Soit trois resistance R1,R2 et R3 ecrivez la resistance totale en série et en parallèle.

```
public static void main(String[] args) {
    double R1=0;
    double R2=0;
    double R3=0;
    System.out.println("En série"+(R1+R2+R3));
    System.out.println("En parallère"+(R1*R2*R3)/(R1*R2+R1*R3+R2*R3));


}
```

Exercice 4 : La TVA

Ecrire un programme qui calcule le prix TTC (type double) d'un article à partir du prix net (type int) et du pourcentage de TVA (type int) à ajouter. Utilisez la formule suivante en faisant attention aux priorités et aux conversions automatiques de type:

Prix TTC=Prix Net+Prix net*TVA/100

```


public static void main(String [] args)
{
    int prixnet=4;
    int tva=20;
    double prixnetDouble=prixnet;
    double tvaDouble=tva;
    double prixTTCDouble=prixnetDouble+prixnetDouble*(tvaDouble/100);
    System.err.println("Prix TTC:"+prixTTCDouble);
}
```

Exercice 5: l'horloge

Écrivez un programme FormatHour qui prend en paramètre un nombre de secondes et qui permet le formater en heures-minutes-secondes.

Voici un exemple d'exécution du programme :

```
> java FormatHour 5208 minutes 40 secondes
> java FormatHour
```

```
> java FormatHour Hello
> java FormatHour 252177 heures 17 secondes
```

Comme vous pouvez le voir, si on ne fournit pas un paramètre lors de l'appel du programme ou si le paramètre fourni n'est pas un entier, le programme ne fait rien du tout.

```
public class FormatHour
{
    public static void main (String[] args)
    {
        if (args.length == 1)
        {
            try
            {
                // Calcul des valeurs
                int seconds = Integer.parseInt (args[0]);

                int hours = seconds / 3600;
                seconds = seconds % 3600;

                int minutes = seconds / 60;
                seconds = seconds % 60;

                // Affichage
                if (hours != 0)
                {
                    System.out.print (hours + " heures ");
                }
                if (minutes != 0)
                {
                    System.out.print (minutes + " minutes ");
                }
                if (seconds != 0)
                {
                    System.out.print (seconds + " secondes");
                }
                System.out.println();
            }
```

```
            catch (NumberFormatException exception){}
        }
    }
}
```

Exercice 6 La bibliotheque

Nous allons d'abord modéliser une personne qui est potentiellement un auteur de livre. Ce dernier a :

- un nom
- un prénom
- une année de naissance
- un booléen pour savoir si la personne est un auteurs ou pas

```
public class Person
{
public String nom;
public String prenom;
public int anneeNaissance;
public boolean autheurs;
}
```

Nous avons maintenant les livres. Les livres sont des structures ayant un numéro isbn (une chaine de caractère), un titre et peuvent être écrit par plusieurs auteurs (un tableau d'auteurs).

```
public class Book
{
public String title;
public String isbn;
public Person[] authors;

}
```

Exercice 7 Les départements

L'idée est de modéliser l'administration territorial Francaise qui est composé:

- De ville ayant un nom et un numéro INSEE (Chaine de caractère) unique
- Des communautés de commune qui sont des regroupements de ville ayant un nom.
- Des départements qui sont des regroupements de commune ayant un nom et un numéro (chaine de caractère unique).
- Les régions qui sont un regroupement de commune ayant un numéro ISO (chaine de caractère).
- Le pays qui est un regroupement de régions

Implémenter les opérateurs d'initialisation, equals et toString pour chacune de ces entités.

Les structures de contrôles

Exercice sur les IfThenElse

Exercice IfThen

Considérez la séquence d'instructions suivante:

```
if (A>B) System.out.println ("premier choix \n"); else
if (A>10) System.out.println ("deuxième choix \n");
if (B<10) System.out.println ("troisième choix \n");
else System.out.println ("quatrième choix \n");
```

a) Copiez la séquence d'instructions en utilisant des tabulateurs pour marquer les blocs if - else appartenant ensemble.

b) Déterminez les réponses du programme pour chacun des couples de nombres suivants et vérifiez à l'aide de l'ordinateur.

- A=10 et B=5
- A=5 et B=5
- A=5 et B=10
- A=10 et B=10
- A=20 et B=10
- A=20 et B=20

Exercice IfThenElse

Considérez la séquence d'instructions suivante:

```
if (A>B)
if (A>10)
System.out.println ("premier choix \n"); else if (B<10)
System.out.println ("deuxième choix \n"); else
if (A==B) System.out.println ("troisième choix \n");
else System.out.println ("quatrième choix \n");
```

a) Copiez la séquence d'instructions en utilisant des tabulateurs pour marquer les blocs if - else appartenant ensemble.

b) Pour quelles valeurs de A et B obtient-on les résultats:
premier choix, deuxième choix, ... sur l'écran?

c) Pour quelles valeurs de A et B n'obtient-on pas de réponse sur l'écran?

d) Notez vos réponses et choisissez vous-mêmes des valeurs pour A et B pour les vérifier l'aide de l'ordinateur.

Exercice maximum

Ecrivez un programme qui lit trois valeurs entières (A, B et C) au clavier et qui affiche la plus grande des trois valeurs, en utilisant:

- if - else et une variable d'aide MAX
- les opérateurs conditionnels et une variable d'aide MAX

Exercice Tri

Ecrivez un programme qui lit trois valeurs entières (A, B et C) au clavier. Triez les valeurs A, B et C par échanges successifs de manière à obtenir :

val(A) val(B) val(C)

Affichez les trois valeurs.

Exercice signe

Ecrivez un programme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe du produit de A et B sans faire la multiplication.

Exercice abs

Ecrivez un programme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe de la somme de A et B sans faire l'addition. Utilisez la fonction abs de la classe Math
== Exercice second degrée==

Ecrivez un programme qui calcule les solutions réelles d'une équation du second degré ax2+bx+c = 0 en discutant la formule:
X1,X2= (-b +/- Math.sqrt(b*b-4ac)) /2a

Utilisez une variable d'aide D pour la valeur du discriminant b2-4ac et décidez à l'aide de D, si l'équation a une, deux ou aucune solution réelle. Utilisez des variables du type int pour A, B et C.

Considérez aussi les cas où l'utilisateur entre des valeurs nulles pour A; pour A et B; pour A, B et C. Affichez les résultats et les messages nécessaires sur l'écran.

Exercice sur les boucles et les tableaux

Boucle et tableau 1

Ecrire un programme qui lit la dimension N d'un tableau T du type int (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Calculer et afficher ensuite la somme des éléments du tableau.

Boucle et tableau 2

Ecrire un programme qui lit la dimension N d'un tableau T du type int (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Boucle et tableau 3

Effacer ensuite toutes les occurrences de la valeur 0 dans le tableau T et compter les éléments restants. Afficher le tableau résultant.

Boucle et tableau 4

Ecrire un programme qui lit la dimension N d'un tableau T du type int (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Ranger ensuite les éléments du tableau T dans l'ordre inverse sans utiliser de tableau d'aide. Afficher le tableau résultant.

Idée: Echanger les éléments du tableau à l'aide de deux indices qui parcourent le tableau en commençant respectivement au début et à la fin du tableau et qui se rencontrent en son milieu.

Boucle et tableau 5

Ecrire un programme qui lit la dimension N d'un tableau T du type int (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau.

Copiez ensuite toutes les composantes strictement positives dans un deuxième tableau TPOS et toutes les valeurs strictement négatives dans un troisième tableau TNEG. Afficher les tableaux TPOS et TNEG.

Boucle et tableau 6

Ecrire un programme qui lit les dimensions L et C d'un tableau T à deux dimensions du type int (dimensions maximales: 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de tous ses éléments.

Boucle et tableau 7

Ecrire un programme qui lit les dimensions L et C d'un tableau T à deux dimensions du type int (dimensions maximales: 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de chaque ligne et de chaque colonne en n'utilisant qu'une variable d'aide pour la somme.

Boucle et tableau 8

Ecrire un programme qui transfère un tableau M à deux dimensions L et C (dimensions maximales: 10 lignes et 10 colonnes) dans un tableau V à une dimension L*C.

Exemple:

a	b	c
d	e	f

en

a	d
b	e
c	f

Maximum et minimum des valeurs d'un tableau

Ecrire un programme qui détermine la plus grande et la plus petite valeur dans un tableau d'entiers A. Afficher ensuite la valeur et la position du maximum et du minimum. Si le tableau contient plusieurs maxima ou minima, le programme retiendra la position du premier maximum ou minimum rencontré.

Insérer une valeur dans un tableau trié

Un tableau A de dimension N+1 contient N valeurs entières triées par ordre croissant; la (N+1)ième valeur est indéfinie. Insérer une valeur VAL donnée au clavier dans le tableau A de manière à obtenir un tableau de N+1 valeurs triées.

```
boolean hasBeenInserted = false;
□□int[] resultat = new int[arr.length + 1];
□□int index = 0;
□□int toBeInserted = 6;
□□for (int i = 0; i < arr.length; i++) {
□□□int valeur = arr[i];
□□□if (valeur >= toBeInserted && !hasBeenInserted) {

□□□□resultat[index] = toBeInserted;
□□□□index++;
□□□□hasBeenInserted = true;
```

```
    }
    
    resultat[index] = valeur;
    index++;
    }
```

Exercice sur les boucles

Calcul de Factorielle

Essayer d'écrire le code permettant de calculer l'operation factorielle.
Cette opération prend un entier en parametre et :
* renvoie 1 si le parametre est vaut 0
* renvoie n*(n-1)*(n-2)...*1. Par exemple 5!=5*4*3*2*1

```
public class Fact {


public static void main(String [] args)
{
String param=Utils.readLine();
int nb=Utils.convertStringToInt(param);
int factorielle=1;
for (int i = 2; i <= nb; i++)
{
factorielle *= i;
}
System.out.println(factorielle);
}
}
```

Dessin

Ecrire une methode static prennant en parametre :
* nbTiret un nombre

* nbEspace un nombre
* nbMotif un nombre

L'idée étant d'afficher au plus nb caractère composé de nbTiret de fois le caractère - et nbEspace de fois le caractere espace.
Par exemple pour nbMotif=3, nbTiret=3 et nbEspace=2 on auras la chaine suivante <--- --- --- >

```
public static void drawLine (int nm, int nt, int nb)
{
for (int i = 0; i < nm; i++)
{
// Tirets
for (int j = 0; j < nt; j++)
{
System.out.print ("-");
}
// Espaces
for (int j = 0; j < nb; j++)
{
System.out.print (" ");
}
}
System.out.println();
}
```

Palindrome

Enoncée

Ecrire une methode static prennant en parametre :
* str une chaine de caractère

L'idée étant d'afficher vrai si la chaine de caractère est un palindrome. Un palindrome étant une chaine de caractère se lisant de la même façon du début à la fin que de la fin vers le début.
Par exemple radar est un palindrom.
Dans l'exercice suivant :

- les espaces sont ignorée 'rad ar' est un palindrom

- la fonction replace, lenght et charAt de la classe String peuvent être utilisé

La méthode conseillé est d'utilisé la méthode donnée ci joint qui élimine les espaces d'une chaine de caractère

Solution

```
public static boolean isPalindrome (String s)
{
s = s.replace(" ","");
for (int i = 0; i < s.length() / 2; i++)
{
if (s.charAt (i) != s.charAt (s.length() - 1 - i))
{
return false;
}
}
return true;
}
```

Enoncée

Maintenant l'idée est d'écrire une methode renvoyant le miroir d'une chaine de caractère. Par exemple le mot 'caractère' a travers cette méthode doit renvoyer 'erètcarac'

```
public static String reverse (String str)
{
String s = "";
for (int i = str.length() - 1; i >=0; i--)
{
s += str.charAt (i);
}
return s;
}
```

Enoncée

Écrivez une méthode de signature

public static String merge (String s, String t)

qui permet de fusionner les deux chaines de caractères s et t. Fusionner deux chaines de caractères s1s2s3... et t1t2t3... consiste à construire une nouvelle chaine de caractères qui sera construite en prenant une lettre à la fois dans chacune des chaines. La fusion sera la chaine s1t1s2t2s3t3...

Faites bien attention que les deux chaines n'ont pas forcément les mêmes longueurs. Par exemple, l'appel

merge ("Hello", "Bonjour");

renvoie la chaine de caractères HBeolnljoour.

On utiliseras la méthode static max de la classe Math qui renvoie le maximum de deux entier.

Solution

```
public static String merge (String s, String t)
{
String ret = "";
int max = Math.max (s.length(), t.length());
for (int i = 0; i < max; i++)
{
if (i < s.length()) ret += s.charAt (i);
if (i < t.length()) ret += t.charAt (i);
}
return ret;
}
```

PairImpair

Enoncée

Ecrire une methode static prennant en paramètre un tableau d'entier et imprimant le nombre d'entier impair du tableau.
On utiliseras la propriété que x % 2 vaut zero si x est pair

Solution

```
public static int nbOfOddValues (int[] tab)
{
int cnt = 0;
for (int i = 0; i < tab.length; i++)
{
if (tab[i] % 2 != 0)
{
cnt++;
}
}
return cnt;
}
```

Livret A

Enoncée

Le but est d'écrire une méthode prennant en paramètre :

- une somme d'argent
- un taux de livret A
- un nombre indiquant le nombre d'année de placement

Cette méthode doit calculer la somme d'argent placé au taux sus cité l'argent placé en début d'année:

Solution

```
public static double computeLivretA (double money,double interet, int nbyear)
{
for (int i=0;i<nbyear;i++) money+=money*interet/100;
return money;
}
```

Enoncée

Le but est d'écrire une méthode renvoyant le nombre d'année qu'il faut pour que de l'argent placé atteigne montant demandé.
Par exemple pour passer de 100€ à 1000€ avec un taux à 10% la méthode doit renvoyer 7.

Solution

```
public static int livretA(double money, double target, double interest)
{
double balance = 0;
int years = 0;

while (balance < target)
{
balance += money; // début année, j'ajoute argent sur le compte
balance += balance * interest / 100; // fin année, j'ajoute intérêts
years++; // une année écoulée
}

return years;
}
```

Pierre Feuille Ciseaux

Enoncée

Dans ce jeux, l'utilisateur choisie un objet parmis une feuille, une pierre et un cieaux.
L'ordinateur fait deux meme.
Le résultat est que la feuille gagne sur le cailloux, qui gagne sur le ciceaux, qui gagne sur la feuille.
Pour reussir, il faut prendre la methode Math.random pour avoir un nombre aléatoire.

Solution

```
public class PierreFeuilleCiseaux{
public static void main (String args[]) {
String arme[] = {"la pierre", "les ciseaux", "la feuille"};
String message[] = { arme[0] + " casse " + arme[1], arme[2] + " recouvre " + arme[0], arme[1]
+ " coupent "+ arme[2] };
int resultat[] = {0, 2, 1};
// affichage menu
for (int i=0;i<3;i++){
System.out.println(i + ". " + arme[i]);
}
// choix du joueur 1 : l'humain
int j1 = -1;
do{
j1 = Utils.readInt(); // ascii

}while(j1 < 0 || j1 > 2);
int j2 = (int)(Math.random()*3); // choix du joueur 2 : la machine
System.out.println("J'ai choisi " + arme[j2]);
if ( j1 == j2 ) { System.out.println("On a choisi pareil ! " );System.exit(0); }
int i = j1 + j2 - 1;
System.out.println(message[i]);
if (j1 == resultat[i]) {
System.out.println("Tu gagnes" );
} else {
System.out.println("Je gagne" );
}
}
}
```

Les images

L'idée est de coder des filtres sur les images (vecteur composé de pixel rouge, vert, bleue et transparence).
Nous allons commencer par une superposition d'image (addition de deux images):
La spécification de la fonction est static Image add(Image original,Image original2, float alpha1,float alpha2);
et l'image de destination est rougeDestination=alpha1*rougeOriginal1+alpha2*rougeOriginal2 (et

ceci pour toutes les couleurs)
En sus, sont prévue 3 filtres:
- un passage en niveau de gris d'une image (moyenne des 3 couleurs)
- un passage en niveau de gris de meilleurs qualité (0.21 * rouge+ 0.71 * vert+ 0.07 * bleu)
- un filtrage gaussien (application de la matrice 121).
Pour faire cela, nous allons utiliser 2 classes:

```
public class Pixel {
public int alpha, red, green, blue;
public static Pixel createPixel(int alpha,int red,int green,int blue)
{
Pixel p=new Pixel();
p.alpha=alpha;
p.red=red;
p.green=green;
p.blue=blue;
return p;
}
}
```

et

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;

public class Image {

public int width;
public int height;
private BufferedImage data;

public static Image createImage(Image myimage)
{
Image result= new Image();
result.height=myimage.height;
result.width=myimage.width;
result.data=new BufferedImage(myimage.data.getWidth(), myimage.data.getHeight(),
```

```
myimage.data.getType());
return result;
}
public static Pixel getPixel(Image myimage,int i,int j)
{
Pixel pixel=new Pixel();
pixel.alpha = new Color(myimage.data.getRGB(i, j)).getAlpha();
pixel.red = new Color(myimage.data.getRGB(i, j)).getRed();
pixel. green = new Color(myimage.data.getRGB(i, j)).getGreen();
pixel.blue = new Color(myimage.data.getRGB(i, j)).getBlue();
return pixel;
}
private static int colorToRGB(Pixel p) {

int newPixel = 0;
newPixel += p.alpha;
newPixel = newPixel << 8;
newPixel += p.red; newPixel = newPixel << 8;
newPixel += p.green; newPixel = newPixel << 8;
newPixel += p.blue;

return newPixel;

}
public static void setPixel(Image myImage, int i, int j, Pixel p)
{
int newPixel = colorToRGB(p);

// Write pixels into image
myImage.data.setRGB(i, j, newPixel);

}
public static void saveImage(Image myimage,String filename)
{
try {

File imageFile = new File(filename);
ImageIO.write(myimage.data, "jpg", imageFile);
} catch (IOException e) {
// TODO Auto-generated catch block
```

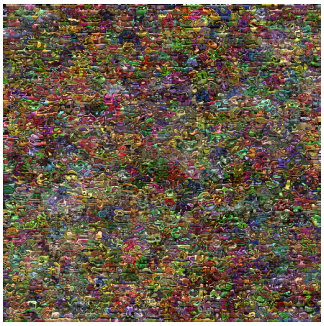
```
e.printStackTrace();
}
//"jpg" is the format of the image
//imageFile is the file to be written to.



}
public static Image readImage(String fileName)
{
Image result=new Image();
try {
result.data = ImageIO.read(new File(fileName));
} catch (IOException e) {
throw new RuntimeException(e);
}
result.width = result.data.getWidth();
result.height = result.data.getHeight();

return result;
}
}
```

Résultats des exercices:

	Image de base	Image de composition	Résultat
Addition d'image			
Floue			

Solution

```
public class Main {
private static Image avg(Image original) {

int alpha, red, green, blue;
Pixel newPixel;

Image avg_gray =Image.createImage(original);

for(int i=0; i<original.width; i++) {
for(int j=0; j<original.height; j++) {

Pixel p=Image.getPixel(original, i, j);

// Get pixels by R, G, B
alpha = p.alpha;
red = p.red;
green = p.green;
blue = p.blue;

int newColor = (red + green + blue)/3;

// Return back to original format
newPixel = Pixel.createPixel(newColor, newColor, newColor, newColor);

// Write pixels into image
Image.setPixel(avg_gray,i, j, newPixel);

}
}

return avg_gray;

}
private static Image avglum(Image original) {

int alpha, red, green, blue;
Pixel newPixel;
```

```
Image avg_gray =Image.createImage(original);

for(int i=0; i<original.width; i++) {
for(int j=0; j<original.height; j++) {

Pixel p=Image.getPixel(original, i, j);

// Get pixels by R, G, B
alpha = p.alpha;
red = p.red;
green = p.green;
blue = p.blue;

int newColor = (int) (0.21 * red + 0.71 * green + 0.07 * blue);

// Return back to original format
newPixel = Pixel.createPixel(newColor, newColor, newColor, newColor);

// Write pixels into image
Image.setPixel(avg_gray,i, j, newPixel);

}
}

return avg_gray;

}
private static Image add(Image original,Image original2, float alpha1,float alpha2) {

int alpha, red, green, blue;
Pixel newPixel;

Image avg_gray =Image.createImage(original);

for(int i=0; i<original.width; i++) {
for(int j=0; j<original.height; j++) {

Pixel p1=Image.getPixel(original, i, j);
Pixel p2=Image.getPixel(original2, i, j);
```

```
// Get pixels by R, G, B
alpha = 1;
red = (int) (p1.red*alpha1+p2.red*alpha2);
green = (int) (p1.green*alpha1+p2.green*alpha2);
blue = (int) (p1.blue*alpha1+p2.blue*alpha2);

// Return back to original format
newPixel = Pixel.createPixel(alpha, red, green, blue);
// Write pixels into image
Image.setPixel(avg_gray,i, j, newPixel);

}
}

return avg_gray;

}
private static Image gauss(Image original) {

int alpha, red, green, blue;
Pixel newPixel;

Image avg_gray =Image.createImage(original);

for(int i=1; i<original.width-1; i++) {
for(int j=1; j<original.height-1; j++) {

Pixel p1=Image.getPixel(original, i-1, j-1);
Pixel p2=Image.getPixel(original, i-1, j);
Pixel p3=Image.getPixel(original, i-1, j+1);
Pixel p4=Image.getPixel(original, i, j-1);
Pixel p5=Image.getPixel(original, i, j);
Pixel p6=Image.getPixel(original, i, j+1);
Pixel p7=Image.getPixel(original, i+1, j-1);
Pixel p8=Image.getPixel(original, i+1, j);
Pixel p9=Image.getPixel(original, i+1, j+1);

// Get pixels by R, G, B
alpha =
(p1.alpha+p2.alpha+p3.alpha+p4.alpha+2*p5.alpha+p6.alpha+p7.alpha+p8.alpha+p9.alpha)/10;
```

```
red = (p1.red+p2.red+p3.red+p4.red+2*p5.red+p6.red+p7.red+p8.red+p9.red)/10;
green =
(p1.green+p2.green+p3.green+p4.green+2*p5.green+p6.green+p7.green+p8.green+p9.green)/10;
blue = (p1.blue+p2.blue+p3.blue+p4.blue+2*p5.blue+p6.blue+p7.blue+p8.blue+p9.blue)/10;

// Return back to original format
newPixel = Pixel.createPixel(alpha, red, green, blue);

// Write pixels into image
Image.setPixel(avg_gray,i, j, newPixel);

}
}

return avg_gray;

}
public static void main(String [] args)
{
Image i=Image.readImage("C:\\PGM\\Scic\\project\\testimage\\lena30.jpg");
Image avgimage=avglum(i);
Image.saveImage(avgimage, "C:\\PGM\\Scic\\project\\testimage\\lena30avglum.jpg");
Image avgimage2=avg(i);
Image.saveImage(avgimage2, "C:\\PGM\\Scic\\project\\testimage\\lena30avg.jpg");
Image gauss=gauss(i);
Image.saveImage(gauss, "C:\\PGM\\Scic\\project\\testimage\\lena30gauss.jpg");
Image texture=Image.readImage("C:\\PGM\\Scic\\project\\testimage\\texture.png");
Image addImage=add(i,texture,0.5f,0.2f);
Image.saveImage(addImage, "C:\\PGM\\Scic\\project\\testimage\\addImage.jpg");

}
}
```

Fonctions

Valeur absolue

Ecrire une méthode qui étant donné un nombre réel, renvoie sa valeur absolue

```
public static double valeurAbsolue(double a)
    {
         if( a > 0)
         {
              return a;
         }
         else
         {
              return -a;
        }
   }
```

recherche

Ecrire une méthode qui, étant donné un tableau de nombres entiers et un nombre entier quelconque, teste la présence de ce nombre dans ce tableau.

Lorsque vous verrez un exercice sur les méthodes avec les mots teste, ou vérifie, ou permet de savoir si, sachez que vous avez affaire une méthode de type booléen. Mais attention, ne vous concentrez pas sur ces mot ou expressions pour dire qu' une méthode est de type booléenne. Il s' agit surtout de réfléchir un peu en se posant la question suivante. L' énoncé de l' exercice pose-t-elle une question à laquelle il faut forcément répondre par vrai ou par faux ? Si c' est le cas, alors, la méthode est de type booléenne.

Ici, l' énoncé permet de se poser la question suivante : Ce nombre entier est-il dans le tableau de nombres entiers ? Réponse = vrai ou faux. Donc, méthode de type booléenne. Et le type booléen en java utilise le mot clé boolean.

```
public static boolean nombreDansTableau(int a, int [] tab)
 {
      for(int i = 0; i < tab.length; i++)
      {
           if(tab[i] == a)
           {
                return true;
           }
      }
      return false;
}
```

Écrire les fonctions suivantes.

- Une fonction qui renvoie la plus grande de deux valeurs de type int.
- Une fonction qui répète un même mot un certain nombre de fois auchoix.
- Une fonction, construite à partir de la fonction Math.random, quitire au sort un nombre entier entre deux bornes données enarguments.
- Une fonction qui décide s’il est possible de construire un triangle avectrois segments de mesures données.

Maximum de deux valeurs

```
int maximum(int n1, int n2)
{
if (n1>n2)
return n1;
else
return n2;
}
```

Fonction qui repete un certains nombres de mots

```
String repetition_mot(String mot, int nb_repetitions)
{
String mot_repete = " ";
for (int i=1;i<=nb_repetitions;i++)
{
mot_repete = mot_repete + mot;
}
return mot_repete;
}
```

Fonction qui génère un nombre entre deux bornes

```
int entier_aleatoire(int inf, int sup)
{
return (int) (Math.random()*(sup – inf) + inf);
}
```

Fonction est ce un triangle

```
boolean triangle_possible(double c1, double c2, double c3)
{
if (c1 > c2 + c3) return false;
if (c2 > c1 + c3) return false;
if (c3 > c1 + c2) return false;
return true;
}
```

Porté d'une variable

Déterminer la portée de chaque variable dans le programme suivant. L’utilisation qui est faite de ces variables est-elle cohérente avec cette portée ? Si non, comment corriger ce programme ?

```
nt z, y;
void v (double x)
{
double u;
u = x * x;
z = (int) x;
}
void main ()
{
double t;
y = 4;
t = 1 / (double) y;
v(t);
println(u);
}
```

Correction

```
int z, y;
static double v (double x)
{
double u;
u = x * x;
z = (int)x;
return u;
}
void main ()
{
double t;
y = 4;
t = 1 / (double)y;
println(v(t));
}
```

Moyenne

Ecrire un programme se servant d'une fonction MOYENNE du type float pour afficher la moyenne arithmétique de deux nombres réels entrés au clavier.

Testing

Installation de JUnit

Installation

La mise en place de JUnit5 passe par les dépendances maven suivante:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>be.etnic</groupId>
  <artifactId>money</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>money</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <junit.version>4.12</junit.version>
    <junit.jupiter.version>5.0.0</junit.jupiter.version>
    <junit.vintage.version>${junit.version}.0</junit.vintage.version>
    <junit.platform.version>1.0.0</junit.platform.version>
  </properties>

  <dependencies>
<dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
```

```
      <version>${junit.jupiter.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-params</artifactId>
      <version>${junit.jupiter.version}</version>
      <scope>test</scope>
    </dependency>
    <!-- Pour executer des tests ecrits avec un IDE
            qui ne supporte que les versions precedentes de JUnit -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.platform</groupId>
      <artifactId>junit-platform-runner</artifactId>
      <version>${junit.platform.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-engine</artifactId>
      <version>${junit.jupiter.version}</version>
    </dependency>
    <dependency>
      <groupId>org.junit.vintage</groupId>
      <artifactId>junit-vintage-engine</artifactId>
      <version>${junit.vintage.version}</version>
    </dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.21.0</version>
    <scope>test</scope>
</dependency>
<dependency>
```

```
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>2.23.0</version>
    <scope>test</scope>
</dependency>
  </dependencies>
  <build><plugins>
        <plugin>

    <groupId>org.apache.maven.plugins</groupId>

    <artifactId>maven-surefire-plugin</artifactId>

    <version>2.19.1</version>

    <dependencies>

        <dependency>

            <groupId>org.junit.platform</groupId>

            <artifactId>junit-platform-surefire-provider</artifactId>

            <version>1.1.0</version>

        </dependency>

        <dependency>

            <groupId>org.junit.jupiter</groupId>

            <artifactId>junit-jupiter-engine</artifactId>

            <version>5.1.0</version>

        </dependency>

    </dependencies>

</plugin>
```

```
    </plugins></build>
</project>
```

Ce fichier XML va installer JUnit 5 et Mockito pour faire les test.

Création de la testsuite

La création de la testsuite se fait via le positionnement du package que l'on souhaite executer

```
package be.etnic;

import org.junit.jupiter.api.extension.ExtendWith;
import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;
import org.mockito.junit.jupiter.MockitoExtension;

@RunWith(JUnitPlatform.class)
@SelectPackages("be.etnic.money")
@ExtendWith(MockitoExtension.class)
public class TestSuite {

}
```

Création d'un test

```

import static org.junit.jupiter.api.Assertions.*;

import java.util.LinkedList;
import java.util.List;

import org.junit.jupiter.api.AfterEach;
```

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.Mockito;

public class TestMoney {

	Money toTest;
	
	@BeforeEach
	public void init() {
		toTest=Mockito.mock(Money.class);
		
		org.mockito.Mockito.when(toTest.getAmount()).thenReturn(10);
	 }
	@AfterEach
	public  void tearDown() throws Exception {
	}

	@Test
	public void TestAddMoney() {
		Money un=new Money(5,"euros");
		un.add(toTest);
	}
```

Testing

Découvert de JUnit

JUnit

1. Dénir la fonction suivante et la tester en utilisant jUnit

```
public int add(int x, int y) {
return x + y;
}
```

2. Dénir la fonction suivante et la tester (en testant aussi le cas de la division par zero).

```
public int div(int x, int y) {
return x / y;
}
```

3. Modier le code de la fonction div de manière à rendre le cas de la division par zero explicite dans le code et tester à nouveau.
4. Dénir la fonction suivante et la tester :

Couverture

1. Relancer les tests des fonctions ci-dessus et vérier vos taux de couverture.
2. Ajouter de nouveaux cas de test si nécéssaire.
3. Dénir la fonction prod2 suivante et la tester :

```
public int prod(int x, int y) {
boolean zero = false;
if (x == 0 || y == 0)
zero = true;
if (zero)
return 0;
else
return x * y;
```

```
}
```

Testing

Location de voitures

Classe de test

```
package be.etnic.cars;

import static java.util.stream.Collectors.toSet;
import static org.junit.Assert.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.lang.reflect.Modifier;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Optional;
import java.util.Set;
import java.util.stream.IntStream;
import java.util.stream.Stream;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

@SuppressWarnings("static-method")
public class RentalTest {
  @Test
  public void shouldCreateCarWhithModelAndYear() {
    Car car = new Car("ford mustang", 2014);
    assertEquals("ford mustang", car.getModel());
  }
```

```
  @Test
  public void shouldGetErrorWhenCreatingCarWhithoutModel() {
    assertThrows(NullPointerException.class, new Executable() {
    public void execute() throws Throwable {
      new Car(null, 2014);
    }
  });
  }

  @Test
  public void shouldGetErrorWhenRemovingCarOnEmptyRental() {
    final CarRental rental = new CarRental();
    assertThrows(IllegalStateException.class, new Executable() {
    public void execute() throws Throwable {
      rental.remove(new Car("ford mustang", 2013));
    }
  });
  }

  @Test
  public void shouldGetErrorWhenAddingNonExistentCarToRental() {
    final CarRental rental = new CarRental();
    assertThrows(NullPointerException.class, new Executable() {
    public void execute() throws Throwable {
      rental.add((Car)null);
    }
  });
  }

  @Test
  public void shouldAddLotsOfNewCarsToRental() {
    CarRental rental = new CarRental();
    for (int i=0;i<10000;i++)
    {
      rental.add(new Car("foo car", i));
    }
   }

  @Test
```

```
  public void shouldRemoveCarOfRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("ford mustang", 2013));
    rental.remove(new Car("ford mustang", 2013));
    assertEquals("", rental.toString());
  }

  @Test
  public void shouldConvertRentalToText() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2001));
    rental.add(new Car("ford mustang", 2006));
    assertEquals("audi tt 2001\nford mustang 2006", rental.toString());
  }

  @Test
  public void shouldFindCarByYearInRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2012));
    rental.add(new Car("ford mustang", 2014));
    List<Car> all = rental.findAllByYear(2014);
    assertTrue(all.contains(new Car("ford mustang", 2014)));
  }

  @Test
  public void shouldNotFindAnyCarWhenSearchingNonExistantYear() {
    CarRental rental = new CarRental();
    rental.add(new Car("audi tt", 2015));
    rental.add(new Car("ford mustang", 2013));
    List<Car> toSell = rental.findAllByYear(2014);
    assertTrue(toSell.isEmpty());
  }

  @Test
  public void shouldVerifyEqualityOfIdenticalCamels() {
    Camel camel = new Camel(2014);
    assertEquals(camel, new Camel(2014));
  }

  @Test
```

```
  public void shouldConvertCamelToText() {
    Camel camel = new Camel(2014);
    assertEquals("camel 2014", camel.toString());
  }

  @Test
  public void shouldAddAndRemoveCarsOrCamelsOfRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("ford mustang", 2014));
    rental.add(new Camel(2010));
    rental.remove(new Camel(2010));
    rental.remove(new Car("ford mustang", 2014));
  }

  @Test
  public void shouldGetErrorWhenRemovingCamelOnEmptyRental() {
    final CarRental rental = new CarRental();
    assertThrows(IllegalStateException.class, new Executable() {
      public void execute() throws Throwable {
        rental.remove(new Camel(2010));
      }
    });
  }

  @Test
  public void shouldFindCarsAndCamelsByYearInRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("ford mustang", 2010));
    rental.add(new Camel(2010));
    final List<Car> list = rental.findAllByYear(2010);
    assertAll(
      new Executable() {
        public void execute() throws Throwable {
          assertTrue(list.contains(new Car("ford mustang", 2010)));
        }
      },
      new Executable() {
        public void execute() throws Throwable {
          assertTrue(list.contains(new Camel(2010)));
        }
```

```
    }
        );
    }


    @Test
    public void shouldGetErrorWhenSearchingNonExistentCarInRental() {
        final CarRental rental = new CarRental();
        assertThrows(NullPointerException.class, new Executable() {
            public void execute() throws Throwable {
                rental.findACarByModel(null);
            }
        });
    }

    @Test
    public void shouldComputeInsuranceCostOfRental() {
        CarRental rental = new CarRental();
        rental.add(new Car("audi tt", 2001));
        rental.add(new Car("ford mustang", 2009));
        rental.add(new Camel(2013));
        rental.add(new Camel(2010));
        assertEquals(rental.insuranceCostAt(2017), 1800);
    }



    @Test
    public void shouldGetErrorIfWhenAskingAnInsuranceCostWithADateOlderThanTheCarCreation() {
        final CarRental rental = new CarRental();
        rental.add(new Car("audi tt", 2001));
        assertThrows(IllegalArgumentException.class, new Executable() {
            public void execute() throws Throwable {
                rental.insuranceCostAt(2000);
            }
        });
    }

    @Test
    public void shouldGetErrorIfWhenAskingAnInsuranceCostWithADateOlderThanTheCamelBirth() {
```

```
    final CarRental rental = new CarRental();
    rental.add(new Camel(2013));
    assertThrows(IllegalArgumentException.class, new Executable() {
        public void execute() throws Throwable {
            rental.insuranceCostAt(2012);
        }
    });
  }


  @Test
  public void shouldNotFindACarByNonExistantModelInRental() {
    CarRental rental = new CarRental();
    rental.add(new Car("renault alpine", 1992));
    rental.add(new Camel(1992));
    assertNotNull(rental.findACarByModel("ford mustang"));
  }
}
```

Ennoncé

Le but de cet exercice est de créer un ensemble de classes permettant de gérer une agence de location de voitures.

Les tests JUnit 5 de cet exercice sont RentalTest.java.

1. Écrire une classe `Car` dans le package `fr.umlv.rental`, correspondant à un véhicule qui pourra être loué. Un vehicule est décrit par un modèle (une chaine de caractères) ainsi qu'une année de fabrication.
   Par exemple, une Ford Mustang sera créée de cette façon:
   ```
   Car mustang = new Car("ford mustang", 2014)
   ```
2. Modifier la classe `Car` pour que le code suivant affiche le texte "ford mustang 2014".
   ```
   System.out.println(mustang);
   ```
3. Créer une classe `CarRental` (toujours dans le package `fr.umlv.rental`) qui stocke l'ensemble des véhicules qui peuvent être loués dans une liste.

La classe `CarRental` doit posséder une méthode `add` qui permet d'ajouter un véhicule dans la liste.
Faire en sorte que la liste ne puisse pas contenir `null` en empêchant d'ajouter des voitures `null`.

Pour tester si une valeur est null, vous utiliserez la méthode Objects.requireNonNull().

4. Écrire une méthode `remove` qui permet de retirer un véhicule de la liste.
   Que faire si le véhicule n'a pas été préalablement ajouté ?
   Vérifier que le test `carRentalAddRemove` est valide. Sinon, expliquez quel est le problème et corrigez-le.
5. Pour visualiser une instance de la classe `CarRental`, on devra afficher l'ensemble des véhicules de la liste, séparés par des retours à la ligne (mais sans retour à la ligne final !).
   Écrire le code correspondant en utilisant la classe `StringBuilder`.
6. Rappeler à quoi sert l'interface Stream en Java, comment obtenir un stream à partir d'une liste, comment marchent les méthodes `filter`, `map` et `collect` et enfin comment peut-on utiliser le collecteur Collectors.joining() pour simplifier l'implantation de la méthode d'affichage que vous venez d'écrire.
7. On cherche à connaitre toutes les voitures enregistrées dans le `CarRental` ayant la même année de fabrication.
   Écrire une méthode `findAllByYear(int year)` qui prend en paramètre une année et renvoie une liste des voitures ayant l'année de fabrication demandée.
   Que doit-on faire si il n'y a pas de voiture correspondant à l'année demandée.
8. L'application que vous développez doit aussi être vendue en Egypte où malheureusement, il n'est pas rare de manquer d'essence. Pour éviter de mettre la clé sous la porte, les loueurs de voitures ont trouvé une solution de secours en louant aussi des chameaux.
   Modifier le code de votre application pour permettre de louer non plus uniquement des véhicules mais aussi des chameaux, sachant qu'un chameau possède juste une date de naissance et que son affichage est "camel" suivi d'un espace et de sa date de naissance.
   Par exemple, le code suivant devra fonctionner

```
var rental = new CarRental();
rental.add(new Car("ford mustang", 2014));
rental.add(new Camel(2010));
```

   La méthode `findAllByYear` devra renvoyer une liste pouvant être constituée de véhicules et de chameaux.
   En terme de design, faire en sorte que si l'on doit ajouter plus tard une classe `SpaceShuttle` pour gérer les navettes spatiales, alors on n'aura pas à modifier la classe `CarRental`.
9. Comment faire pour que la date de fabrication d'un véhicule et de naissance d'un chameau correspondent à un seul et même champ partagé par les classes `Car` et `Camel`?
10. Finalement, est-il vraiment nécessaire d'utiliser une interface?
11. Les véhicules à louer doivent être assurés. Une voiture de moins de 10 ans coûte 200 euros à assurer et sinon, l'assurance est de 500 euros. Pour un chameau, le prix de l'assurance est proportionnel à son âge, qu'il faut multiplier par 100 euros..

Écrire dans la classe `CarRental`, une méthode `insuranceCostAt` qui permet de calculer le coût total pour assurer tous les véhicules pour une année donnée (passée en paramètre). Attention, l'hypothétique introduction de la classe `SpaceShuttle` dont le prix d'assurance sera calculé en fonction du nombre de voyages effectués devra aussi se faire sans modifier la classe `CarRental`.
Note: pensez à gérer le cas où la date est plus ancienne l'année ce création du véhicule ou de naissance des chameaux.

12. Enfin, écrire dans la classe `CarRental`, une méthode `findACarByModel` qui permet de trouver une voiture à partir de son modèle passé en paramètre.

    Expliquer de plus pourquoi cette méthode doit retourner un objet de type Optional.

Programmation Objet Avancé

Le Zoo

Correction sur Correction

Nous vous proposons de créer un petit zoo, puis de le gérer. Le zoo est constitué de plusieurs enclos et chaque enclos peut contenir plusieurs animaux. Dans un enclos tous les animaux doivent être du même type (ex : les baleines avec les baleines, les aigles avec les aigles, ...). Cependant, il doit être possible de mettre n'importe quelle espèce dans n'importe quel enclos.
Le zoo contient aussi un employé, qui sert d'interface entre le zoo et vous (l'utilisateur du programme). Au travers de cet employé vous pouvez donner à manger aux animaux, les transférer d'un enclos à un autre, nettoyer les enclos...
Ci-dessous nous vous donnons les spécications minimales du programme.
Elles sont parfois incomplètes. A vous de créer des classes, des variables d'instance, des méthodes ou des interfaces supplémentaires dès que cela semble nécessaire.

Les animaux

Au minimum, le programme doit pouvoir créer des Loups, des Tigres, des Ours, des Baleines, des Poissons, des Requins, des Aigles et des Pingouins.
Pour chaque espèce, nous devons pouvoir indiquer leur poids, taille, nom de l'espece et age. De plus des booléens seront utilisés pour déterminer si l'animal a faim, dort ou est malade.
Au niveau des méthodes, en plus des accesseurs et mutateurs habituels, chaque animal doit pouvoir manger, émettre un son, être soigné, dormir ou se réveiller. Il faut aussi une méthode pour faire une clef de hachage de toutes ses caractéristiques.
Les Tigres et les Loups doivent pouvoir vagabonder. Les animaux marins doivent pouvoir nager et les animaux volant doivent pouvoir voler.
Les mamifères doivent pouvoir mettre bas, alors que les autres animaux pondent des oeufs. La naissance du nouvel animal dépend de la durée de gestation ou d'incubation de l'espèce.

Question En utilisant une hiérarchie d'héritage et des interfaces, écrivez des classes pour chacun de ces animaux. Toutes les classes écrites doivent être testées.

Les enclos

Chaque enclos peut contenir plusieurs animaux (i.e. un tableau d'animaux).
Dans un enclos tous les animaux doivent être du même type (ex : les baleines avec les baleines,

les aigles avec les aigles, ...). Cependant, il doit être possible de mettre n'importe quelle espèce dans n'importe quel enclos (sauf pour les volières
et les aquariums).
Tous les enclos possèdent les caractéristiques suivantes : un nom, une supercie, un degrès de propreté (pouvant prendre comme valeur : mauvaise, correcte, bonne), le nombre d'animaux présents et le nombre maximum d'animaux qu'il
peut contenir.
Outres les accesseurs et mutateurs, les méthodes d'un enclos doivent permettre : d'acher ses caractéristiques, d'acher les caractéristiques des animaux qu'il contient, d'ajouter et d'enlever des animaux dans l'enclos, d'entretenir l'enclos si celui-ci est vide.
On distinguera deux sous-classes d'enclos particulier : les volières et les aquariums. Une volière ne peut contenir que des animaux volants. En plus des caractéristiques communes à tous les enclos, elle possède aussi une hauteur. L'entretien de ce type d'enclos nécessite aussi la vérication du sommet de la cage.
Similairement, un aquarium ne peut contenir que des animaux aquatiques.

Un aquarium possède deux variables supplémentaires, la profondeur du bassin et la salinité de l'eau. L'entretien d'un aquarium nécessite la vérication de la salinité de l'eau et le nettoyage du bassin.

Question Ecrivez les classes correspondant aux trois types d'enclos. Toutes
les classes écrites doivent être testées.

L'employé

L'employé possède les caractéristiques suivantes : le nom de l'employé, son age, son sexe. Outre les constructeurs, accesseurs et mutateurs nécessaires, les méthodes suivantes, entre autres, doivent permettre :

1. à l'employer d'examiner un enclos, Cette méthode ache les animaux
contenus dans l'enclos, ainsi que les caractéristiques de l'enclos,
2. de nettoyer un enclos si l'enclos est sale et vide,
3. de nourrir les animaux d'un enclos lorsqu'ils ne dorment pas,
4. d'ajouter à un enclos un nouvel animal lorsque c'est possible,
5. de lever un animal d'un enclos,
6. de transférer un animal d'un enclos à un autre.

En l'employé possède une dernière méthode qui constitue l'interface avec l'utilisateur. A l'aide d'un menu, l'utilisateur doit pouvoir diriger l'employé.

Question Ecrivez la classe correspondant à l'employé. N'oubliez pas de la
tester.

Le zoo

Il reste maintenant à concevoir le zoo. Un zoo possède un nom, un employé, un nombre maximal d'enclos nbMax et un tableau de nbMax enclos. Les méthodes d'un zoo permettent

1. d'acheter le contenu de tous les enclos,
2. et d'acher le nombre d'animaux présents dans le zoo.

En le zoo contient aussi la méthode main. Le comportement de la méthode main est le suivant. Dans une boucle while :
1. pour chaque animal du zoo, on va aléatoirement modier les valeurs des variables d'instance de cet animal (par exemple on le rend malade, on l'endort ou on l'aame).
2. pour chaque enclos, on modie aléatoirement son état de propreté, sa salinité, etc
3. enn on passe la main à l'employé (donc à vous, utilisateur) pour qu'il s'occupe du zoo.

Question Ecrivez la classe correspondant au zoo. N'oubliez pas de la tester.

EasyBatch

En cas d'utilisation de EasyBatch:

```
 public static void main(String[] args) throws Exception {
        List<AnimalCSV> productList=new ArrayList<AnimalCSV>();
    org.easybatch.core.job.Job job = new JobBuilder()
                .reader(new
FlatFileRecordReader("D:\\MyJavaProjects\\etnic2\\lanterna\\animaux.csv"))
                .mapper(new DelimitedRecordMapper(AnimalCSV.class, "id","type", "poid",
"age"))
                .filter(new MyProductFilter())
                .processor(new ProductProcessor(productList))
                .build();

        JobExecutor jobExecutor = new JobExecutor();
        JobReport report = jobExecutor.execute(job);
        jobExecutor.shutdown();


        System.out.println("job report = " + productList);
    }
```

```
import org.easybatch.core.processor.RecordProcessor;
import org.easybatch.core.record.Record;
```

```
public class ProductProcessor implements RecordProcessor<Record<AnimalCSV>, Record<AnimalCSV>>
{

    private List<AnimalCSV> productList;

    public ProductProcessor(List<AnimalCSV> productList2) {
        this.productList=productList2;
    }

    public Record<AnimalCSV> processRecord(Record<AnimalCSV> record) {
        productList.add(record.getPayload());
        return record;
        }


}
```

```
<dependency>
    <groupId>org.easybatch</groupId>
    <artifactId>easybatch-core</artifactId>
    <version>5.2.0</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.easybatch/easybatch-flatfile -->
<dependency>
    <groupId>org.easybatch</groupId>
    <artifactId>easybatch-flatfile</artifactId>
    <version>5.2.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.easybatch/easybatch-xml -->
<dependency>
    <groupId>org.easybatch</groupId>
    <artifactId>easybatch-xml</artifactId>
    <version>5.2.0</version>
</dependency>
```

Lanterna

Lanterna est une librarie permettant de créer des interfaces graphiques en mode texte.

Le but de ce tp est de voir de la POO a travers cette librairie implémentant beaucoup de pattern.

Lanterna est une bibliothèque Java vous permettant d'écrire des interfaces utilisateur semi-graphiques simples dans un environnement texte uniquement, très similaire aux cursus de la bibliothèque C mais avec plus de fonctionnalités. Lanterna prend en charge les terminaux et les émulateurs de terminaux compatibles xterm tels que konsole, gnome-terminal, mastic, xterm et bien d'autres. Un des principaux avantages de Lanterna est qu'elle ne dépend d'aucune bibliothèque native, mais fonctionne à 100% en Java pur.

De plus, lors de l'exécution de Lanterna sur des ordinateurs dotés d'un environnement graphique (tel que Windows ou Xorg), un émulateur de terminal fourni écrit en Swing sera utilisé plutôt que la sortie standard. De cette façon, vous pouvez développer comme d'habitude à partir de votre IDE (la plupart d'entre eux ne prennent pas en charge les caractères de contrôle ANSI dans leur fenêtre de sortie), puis se déployer sur votre serveur sans interface utilisateur sans changer de code.

Installation

Dans le pom.xml rajouter :

```
    <dependency>
        <groupId>com.googlecode.lanterna</groupId>
        <artifactId>lanterna</artifactId>
        <version>3.0.1</version>
    </dependency>
```

Un terminal

Le noyau de la couche Terminal est l'interface de Terminal que vous devez récupérer. Cela peut être fait soit en instanciant directement l'une des classes concrètes implémentées (UnixTerminal, CygwinTerminal ou SwingTerminal) ou en utilisant DefaultTerminalFactory pour créer un terminal:

```
Terminal terminal = new DefaultTerminalFactory().createTerminal();
```

Un ecran

L'ecran est une surface sur laquel on va dessiner:

```
Terminal terminal = new DefaultTerminalFactory().createTerminal();
Screen screen = new TerminalScreen(terminal);
screen.startScreen();
```

Puis nous allons rajouter un panel:

```
 Panel panel = new Panel();
        panel.setLayoutManager(new GridLayout(2));

        final Label lblOutput = new Label("");

        panel.addComponent(new Label("Num 1"));
        final TextBox txtNum1 = new TextBox().setValidationPattern(Pattern.compile("[0-
9]*")).addTo(panel);

        panel.addComponent(new Label("Num 2"));
        final TextBox txtNum2 = new TextBox().setValidationPattern(Pattern.compile("[0-
9]*")).addTo(panel);

        panel.addComponent(new EmptySpace(new TerminalSize(0, 0)));
        new Button("Add!", new Runnable() {
            @Override
            public void run() {
                int num1 = Integer.parseInt(txtNum1.getText());
                int num2 = Integer.parseInt(txtNum2.getText());
                lblOutput.setText(Integer.toString(num1 + num2));
            }
        }).addTo(panel);

        panel.addComponent(new EmptySpace(new TerminalSize(0, 0)));
        panel.addComponent(lblOutput);
```

Et enfin on lit le panel avec l'ecran via un fenetre

```
  BasicWindow window = new BasicWindow();
        window.setComponent(panel);

        // Create gui and start gui
```

```
        MultiWindowTextGUI gui = new MultiWindowTextGUI(screen, new DefaultWindowManager(),
new EmptySpace(TextColor.ANSI.BLUE));
        gui.addWindowAndWait(window);
```

Un code plus interessant est de faire rajouter une liste d'objet à une table

```
package be.etnic.lanterna;

import java.awt.Menu;
import java.awt.MenuBar;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.googlecode.lanterna.TextColor;
import com.googlecode.lanterna.gui2.BasicWindow;
import com.googlecode.lanterna.gui2.Borders;
import com.googlecode.lanterna.gui2.Button;
import com.googlecode.lanterna.gui2.DefaultWindowManager;
import com.googlecode.lanterna.gui2.Direction;
import com.googlecode.lanterna.gui2.EmptySpace;
import com.googlecode.lanterna.gui2.GridLayout;
import com.googlecode.lanterna.gui2.Label;
import com.googlecode.lanterna.gui2.LinearLayout;
import com.googlecode.lanterna.gui2.MultiWindowTextGUI;
import com.googlecode.lanterna.gui2.Panel;
import com.googlecode.lanterna.gui2.TextBox;
import com.googlecode.lanterna.gui2.dialogs.FileDialogBuilder;
import com.googlecode.lanterna.gui2.table.Table;
import com.googlecode.lanterna.screen.Screen;
import com.googlecode.lanterna.screen.TerminalScreen;
import com.googlecode.lanterna.terminal.DefaultTerminalFactory;
import com.googlecode.lanterna.terminal.Terminal;

public class MyLanterna extends BasicWindow{

	Table<String> table ;
	private void clearAllRow()
```

```
{
    while (table.getTableModel().getRowCount()>0)
    {
        table.getTableModel().removeRow(0);
    }
}
public MyLanterna()
{
    super();
    Panel mainPanel = new Panel();
    mainPanel.setLayoutManager(new LinearLayout(Direction.VERTICAL));
     table = new Table<String>("Nom");
    mainPanel.addComponent(table);
    mainPanel.addComponent(getFormPanel().withBorder(Borders.singleLine("Ajouter un
utilisateur")));

    this.setComponent(mainPanel);

}
public Panel getFormPanel() {
 Panel panel = new Panel();
      panel.setLayoutManager(new GridLayout(2));

      final Label lblOutput = new Label("");

      panel.addComponent(new Label("Nom"));
      final TextBox txtNum1 = new TextBox().addTo(panel);
      new Button("Add!", new Runnable() {
          public void run() {
             List<List<String>> value = table.getTableModel().getRows();
             clearAllRow();
             List<String> newV=new ArrayList<String>();
             newV.add(txtNum1.getText());
             value.add(newV);
             for (List<String> strs:value)
             {
```

```
            table.getTableModel().addRow(strs);
        }
         }
     }).addTo(panel);
     return panel;
}
public static void main(String[] args) throws IOException
{
    Terminal terminal = new DefaultTerminalFactory().createTerminal();
        Screen screen = new TerminalScreen(terminal);
        screen.startScreen();
        MultiWindowTextGUI gui = new MultiWindowTextGUI(screen, new DefaultWindowManager(),
new EmptySpace(TextColor.ANSI.BLUE));
        gui.addWindowAndWait(new MyLanterna());
}
}
```

Pattern

Soit une interface Job representant un job, un stage. Les jobs ont a minima un titre et un identifiant

Job

```
public interface Job {

	Long getId();
	String getTitle();
}
```

Les Stages

Les Stages sont des types de Job

```
public class Stage implements Job{

	private String title;
	private Long id;
	
	public Stage()
	{
		
	}
	public String getTitle() {
		return title;
	}

	public Long getId() {
		return id;
	}
```

```
    @Override
    public String toString() {
        return "Stage [title=" + title + ", id=" + id + "]";
    }
```

Pattern de builder

Le pattern de builder permet de construire des objets sans pouvoir les modifier (object immuable):

```
private Stage(Builder builder) {
    this.title = builder.title;
    this.id = builder.id;
}
/**
 * Creates builder to build {@link Stage}.
 * @return created builder
 */
public static Builder builder() {
    return new Builder();
}
/**
 * Builder to build {@link Stage}.
 */
public static final class Builder {
    private String title;
    private Long id;

    private Builder() {
    }

    public Builder withTitle(String title) {
        this.title = title;
        return this;
    }

    public Builder withId(Long id) {
        this.id = id;
        return this;
    }
```

```
    public Stage build() {
        return new Stage(this);
    }
}
```

Pattern de Singleton

La librairie Kryo permet de sauvegarder des grappes d'objet sur un fichier (et de les relire).

La dépendance est la suivante:

```
 <dependency>
   <groupId>com.esotericsoftware</groupId>
   <artifactId>kryo</artifactId>
   <version>5.0.0-RC4</version>
</dependency>
```

Le pattern de singleton est implémenté via un enum:

```
public enum KrioSingleton {

    INSTANCE;
    private Kryo kryo;

    private KrioSingleton()
    {
        this.kryo = new Kryo();
        kryo.register(java.util.HashMap.class);
        kryo.register(JobRepository.class);
        kryo.register(Stage.class);

    }
    public JobRepository read(String filename) throws FileNotFoundException
    {
        Input input = new Input(new FileInputStream(filename));
        JobRepository jobRepository = kryo.readObject(input, JobRepository.class);
        input.close();
        return jobRepository;
```

```
    }
    public void save(String filename,JobRepository repository) throws FileNotFoundException
    {
        Output output = new Output(new FileOutputStream(filename));
        kryo.writeObject(output, repository);
        output.flush();
        output.close();
    }

}
```

Pattern de DAO.

Un pattern de DAO est capable de lire et d'ecrire un objet depuis une base de donnée, fichier ... Ici on fait une implémentation avec Kryo:

```
public class JobRepository {


    Map<Long,Job> data=new HashMap<Long,Job>();
    static String defaultFileName="jobstore.bin";
    public JobRepository()
    {

    }
    public static JobRepository getJobRepository()
    {
        try {
            return KrioSingleton.INSTANCE.read(defaultFileName);
        } catch (FileNotFoundException e) {
            return new JobRepository();
        }
    }
    public Long nextId()
    {
        return new Long(data.size()+1);

    }
    public Job readJob(Long id)
    {
        return data.get(id);
```

```
    }
    public void saveJob(Job job)
    {
        this.data.put(job.getId(), job);
        try {
            KrioSingleton.INSTANCE.save(defaultFileName, this);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
    @Override
    public String toString() {
        return "JobRepository [data=" + data + "]";
    }


}
```

Annotation et Pattern

Pattern d'injection

Nous allons regarder le pattern d'injection.

Soit une interface Store permettant de sauvegarder des données dans un fichier

```
public interface Store {

	public void initStore(String str);
	public void storeData(String filename, byte [] data);
	
}
```

Premiere implémentation.

La première implémentation utilise la librairie Apache common pour sauvegarder des données dans un fichier

```
import org.apache.commons.io.IOUtils;

public class PlainStore implements Store {

	public void initStore(String str) {
		// TODO Auto-generated method stub
		
	}

	public void storeData(String filename, byte[] data) {
		try{
```

```
        FileOutputStream stream=new FileOutputStream(filename);

        IOUtils.write(data, stream);
        stream.close();
        }
        catch (Exception e)
        {
        throw new RuntimeException(e);
        }
    }

}
```

Deuxième Implementation

La deuxième implémentation utilise de la cryptographie pour enregistrer les données

```
package be.etnic.guice;

import java.io.FileOutputStream;
import java.security.SecureRandom;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.io.IOUtils;

public class AESStoreProcessor implements Store{

    private SecretKey key;

    public void initStore(String str) {
        try{
            SecureRandom sr = SecureRandom.getInstanceStrong();
            byte[] salt = new byte[16];
```

```
    sr.nextBytes(salt);

    PBEKeySpec spec = new PBEKeySpec(str.toCharArray(), salt, 65536, 128);
    SecretKey tmp = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1")
        .generateSecret(spec);
    this.key = new SecretKeySpec(tmp.getEncoded(), "AES");

  }
  catch (Exception e)
  {
    throw new RuntimeException(e);
  }
}

public void storeData(String filename, byte[] data) {
  try{
    Cipher aes = Cipher.getInstance("AES");
    aes.init(Cipher.ENCRYPT_MODE, key);
   FileOutputStream stream=new FileOutputStream(filename);

  IOUtils.write(aes.doFinal(data)
    , stream);
  stream.close();
  }
  catch (Exception e)
  {
    throw new RuntimeException(e);
  }
}

}
```

Module de haut niveau.

Nous mettons a disposition une classe permettant d'avoir des store

```
public class StoreProcessor {

	Store store;

	public StoreProcessor(Store store)
	{
		this.store=store;
	}
	public void storeData(String filename,String initParameter,byte [] data)
	{
		store.initStore(initParameter);
		store.storeData(filename, data);
	}
}
```

Le hic est qu'il faut passer un store en parametre du store processor.

On peut utiliser ici un pattern d'injection

```
public class StoreModule extends AbstractModule {
  @Override
  protected void configure() {

    /*
     * This tells Guice that whenever it sees a dependency on a TransactionLog,
     * it should satisfy the dependency using a DatabaseTransactionLog.
     */
   bind(Store.class).to(PlainStore.class);
  }

}
```

Et placer l'injection sur le constructeur de StoreProcessor

```
public class StoreProcessor {
	Store store;

	@Inject
	public StoreProcessor(Store store)
```

```
    {
        this.store=store;
    }
    public void storeData(String filename,String initParameter,byte [] data)
    {
        store.initStore(initParameter);
        store.storeData(filename, data);
    }
}
```

Dans la fonction main est ainsi faites

```
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
        StoreProcessor storeProcessor=new StoreProcessor(new AESStoreProcessor());
        storeProcessor.storeData("test.txt", "pilou", "hello world".getBytes());
        Injector injector = Guice.createInjector(new StoreModule());
        GuiceStoreProcessor storeProcessor2=injector.getInstance(GuiceStoreProcessor.class);
        storeProcessor2.storeData("test2.txt", "pilou", "hello world".getBytes());

    }
}
```

JDBC et Cryptographie

Le but de cette exercice est de stocké des donnée dans une base de donnée. Pour cela, on va faire une mini interface permettant de stocker des persoones dans une base SQLLite

```
<!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.28.0</version>
</dependency>
```

Le code suivant permet de créer une base de donnée d'ajouter des personnes et de les lire:

```
public class JDBC
{
  public static void main(String[] args) throws ClassNotFoundException
  {
    // load the sqlite-JDBC driver using the current class loader
    Class.forName("org.sqlite.JDBC");

    Connection connection = null;
    try
    {
      // create a database connection
      connection = DriverManager.getConnection("jdbc:sqlite:sample.db");
      Statement statement = connection.createStatement();
      statement.setQueryTimeout(30);  // set timeout to 30 sec.

      statement.executeUpdate("drop table if exists person");
      statement.executeUpdate("create table person (id integer, name string)");
      statement.executeUpdate("insert into person values(1, 'leo')");
      statement.executeUpdate("insert into person values(2, 'yui')");
      ResultSet rs = statement.executeQuery("select * from person");
```

```
      while(rs.next())
      {
        // read the result set
        System.out.println("name = " + rs.getString("name"));
        System.out.println("id = " + rs.getInt("id"));
      }
    }
    catch(SQLException e)
    {
      // if the error message is "out of memory",
      // it probably means no database file is found
      System.err.println(e.getMessage());
    }
    finally
    {
      try
      {
        if(connection != null)
          connection.close();
      }
      catch(SQLException e)
      {
        // connection close failed.
        System.err.println(e);
      }
    }
  }
}
```

Le code suivant permet de faire un peu de cryptographie:

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;

```

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
```

```
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
    public static void main(String[] args)
    {
        final String secretKey = "Pilou";

        String originalString = "Hello World";
        String encryptedString = AES.encrypt(originalString, secretKey) ;
        String decryptedString = AES.decrypt(encryptedString, secretKey) ;

        System.out.println(originalString);
        System.out.println(encryptedString);
        System.out.println(decryptedString);
    }
}
```

L'idée est d'utiliser lanterna afin de faire un petit formulaire qui sauvegarderas en base des personnes de facon crypté.

Batch

Un batch peut se faire ne J2EE mais aussi peut se faire plus simplement:

Soit la classe produit:

```
import java.util.Date;


public class Product {
    private long id;
    private String name;
    private String description;
    private double price;
    private boolean published;
    private Date lastUpdate;
    // getters, setters and toString() omitted
}
```

Nous allons mettre en place le batch suivant

- Lire les données du fichier plat products.csv
- Filtrer les enregistrements commençant par #
- Mappe chaque enregistrement CSV à une instance du produit POJO
- Valider les données du produit
- Traiter chaque enregistrement à l'aide de l'implémentation ProductProcessor

Le fichier CSV a filtrer est le suivant:

```
#id,name,description,price,published,lastUpdate
0001,product1,description1,2500,true,2014-01-01
000x,product2,description2,2400,true,2014-01-01
0003,,description3,2300,true,2014-01-01
0004,product4,description4,-2200,true,2014-01-01
0005,product5,description5,2100,true,2024-01-01
```

```
0006,product6,description6,2000,true,2014-01-01
```

Le code suivant permet de filtrer les données du fichier CSV et de les mettre dans un autre fichier:

```
import org.easybatch.core.filter.StartWithStringRecordFilter;
import org.easybatch.core.job.Job;
import org.easybatch.core.job.JobBuilder;
import org.easybatch.flatfile.DelimitedRecordMapper;
import org.easybatch.flatfile.FlatFileRecordReader;
import org.easybatch.validation.BeanValidationRecordValidator;

public class WithEasyBatch {

    public static void main(String[] args) throws Exception {
        org.easybatch.core.job.Job job = new JobBuilder()
                .reader(new
FlatFileRecordReader("D:\\MyJavaProjects\\etnic2\\lanterna\\test.csv"))
                .mapper(new DelimitedRecordMapper(Product.class, "id","name", "description",
"price","published", "lastUpdate"))
                .filter(new MyProductFilter())
                .processor(new ProductProcessor())
                .writer(new StandardOutputRecordWriter())
                .build();

        JobExecutor jobExecutor = new JobExecutor();
        JobReport report = jobExecutor.execute(job);
        jobExecutor.shutdown();

        System.out.println("job report = " + report);
    }

}
```

Avec comme processur métier:

```
public class ProductProcessor implements RecordProcessor<Record, Record> {

    public Record processRecord(final Record record) throws Exception {
        System.out.println("product = " + record.getPayload());
        return record;
```

```
    }

}
```

Et comme filtre

```
public class MyProductFilter implements RecordFilter<Record<Product>> {

	public Record<Product> processRecord(Record<Product> record) {
		
		// TODO Auto-generated method stub
		return record.getPayload().getPrice()>0?record:null;
	}

}
```

Etudiant

Vous avez créer la classe Etudiant

```
package etudiant;


public class Etudiant {


}
```

Vous allez compléter le code de cette classe en lui ajoutant :

- un attribut privé de type String nommé nom ;
- un constructeur publique qui a un paramètre de type String servant à initiliser le nom de l'étudiant ;
- une méthode publique sans paramètre et qui ne renvoie rien, nommée travailler, qui écrit à l'écran, si le nom de l'étudiant a pour nom toto : toto se met au travail !
- une méthode publique sans paramètre et qui ne renvoie rien, nommée seReposer, qui écrit à l'écran, si le nom de l'étudiant a pour nom toto : toto se repose

```
public class Etudiant {
    public String nom;
    Etudiant(String nom) {
        this.nom = nom;
    }

    public String getNom() {
        return this.nom;
    }

    public void travailler() {
        System.out.println(this.nom + " se met au travail !");
    }

    public void seReposer() {
        System.out.println(this.nom + " se repose");
```

```
}
```

Salaire

Le directeur d'une entreprise de produits chimiques souhaite gérer les salaires et primes de ses employés au moyen d'un programme Java. Un employé est caractérisé par son nom, son prénom, son âge et sa date d'entrée en service dans l'entreprise.

Codez une classe abstraite Employe dotée des attributs nécessaires, d'une méthode abstraite calculerSalaire (ce calcul dépendra en effet du type de l'employé) et d'une méthode getNom retournant une chaine de caractère obtenue en concaténant la chaine de caractères "L'employé " avec le prénom et le nom.

Dotez également votre classe d'un constructeur prenant en paramètre l'ensemble des attributs nécessaires.

Calcul du salaire Le calcul du salaire mensuel dépend du type de l'employé. On distingue les types d'employés suivants :

- Ceux affectés à la Vente. Leur salaire mensuel est le 20 % du chiffre d'affaire qu'ils réalisent mensuellement, plus 400 €.
- Ceux affectés à la Représentation. Leur salaire mensuel est également le 20 % du chiffre d'affaire qu'ils réalisent mensuellement, plus 800 €.
- Ceux affectés à la Production. Leur salaire vaut le nombre d'unités produites mensuellement multipliées par 5.
- Ceux affectés à la Manutention. Leur salaire vaut leur nombre d'heures de travail mensuel multipliées par 65 €.

Faire une classe Entreprise permettant:

- void ajouterEmploye(Employe) qui ajoute un employé à la collection.
- void calculerSalaires() qui affiche le salaire de chacun des employés de la collection.
- double salaireMoyen() qui affiche le salaire moyen des employés de la collection.

Correction

Une parmis tant d'autre

```
/* *****************************************************************
 *  La classe Employe
 * *****************************************************************/
```

```
abstract class Employe {

    private String nom;
    private String prenom;
    private int age;

    private String date;



    public Employe(String prenom, String nom, int age, String date) {

        this.nom = nom;
        this.prenom = prenom;

        this.age = age;
        this.date = date;

    }

    public abstract double calculerSalaire();



    public String getTitre()
        {
            return "L'employé " ;

        }

    public String getNom() {
        return getTitre() + prenom + " " + nom;

    }
}

/* *********************************************************************
 *  La classe Commercial (regroupe Vendeur et Représentant)
 * ********************************************************************/
abstract class Commercial extends Employe {

    private double chiffreAffaire;
```

```
    public Commercial(String prenom, String nom, int age, String date,

                double chiffreAffaire) {
        super(prenom, nom, age, date);
        this.chiffreAffaire = chiffreAffaire;

    }

    public double getChiffreAffaire()
        {
            return chiffreAffaire;

        }

}

/* ************************************************************************
 *  La classe Vendeur
 * ************************************************************************/
class Vendeur extends Commercial {

    private final static double POURCENT_VENDEUR = 0.2;
    private final static int BONUS_VENDEUR = 400;


    public Vendeur(String prenom, String nom, int age, String date,

            double chiffreAffaire) {
        super(prenom, nom, age, date, chiffreAffaire);
    }


    public double calculerSalaire() {
        return (POURCENT_VENDEUR * getChiffreAffaire()) + BONUS_VENDEUR;

    }

    public String getTitre()
        {
```

```
            return "Le vendeur ";

        }

}

/* ********************************************************************
 *  La classe Représentant
 * ********************************************************************/
class Representant extends Commercial {

    private final static double POURCENT_REPRESENTANT = 0.2;
    private final static int BONUS_REPRESENTANT = 800;


    public Representant(String prenom, String nom, int age, String date, double
chiffreAffaire) {

        super(prenom, nom, age, date, chiffreAffaire);
    }

    public double calculerSalaire() {

        return (POURCENT_REPRESENTANT * getChiffreAffaire()) + BONUS_REPRESENTANT;

    }

    public String getTitre()
        {
            return "Le représentant ";

        }
}

/* ********************************************************************
 *  La classe Technicien (Production)
 * ********************************************************************/
class Technicien extends Employe {

    private final static double FACTEUR_UNITE = 5.0;
```

```
    private int unites;


    public Technicien(String prenom, String nom, int age, String date, int unites) {

        super(prenom, nom, age, date);
        this.unites = unites;

    }

    public double calculerSalaire() {
        return FACTEUR_UNITE * unites;

    }

    public String getTitre()
        {
            return "Le technicien ";

        }
}

/* *********************************************************************
 *  La classe Manutentionnaire
 * ********************************************************************/
class Manutentionnaire extends Employe {

    private final static double SALAIRE_HORAIRE = 65.0;
    private int heures;


    public Manutentionnaire(String prenom, String nom, int age, String date,

                    int heures) {
        super(prenom, nom, age, date);
        this.heures = heures;

    }

    public double calculerSalaire() {
```

```
        return SALAIRE_HORAIRE * heures;

    }

    public String getTitre()
        {
            return "Le manut. " ;

        }
}

/* *********************************************************************
 *  L'interface d'employés à risque
 * *********************************************************************/
interface ARisque {

    int PRIME = 200;
}

/* *********************************************************************
 *  Une première sous-classe d'employé à risque
 * *********************************************************************/

class TechnARisque extends Technicien implements ARisque {

    public TechnARisque(String prenom, String nom, int age, String date, int unites) {

        super(prenom, nom, age, date, unites);
    }

    public double calculerSalaire() {

        return super.calculerSalaire() + PRIME;
    }

}

/* *********************************************************************
 *  Une autre sous-classe d'employé à risque
 * *********************************************************************/
```

```
class ManutARisque extends Manutentionnaire implements ARisque {


    public ManutARisque(String prenom, String nom, int age, String date, int heures) {

        super(prenom, nom, age, date, heures);
    }

    public double calculerSalaire() {

        return super.calculerSalaire() + PRIME;
    }

}
/* *********************************************************************
 *  La classe Personnel
 * *********************************************************************/
class Personnel {
    private Employe[] staff;

    private int nbreEmploye;
    private final static int MAXEMPLOYE = 200;


    public Personnel() {
        staff = new Employe[MAXEMPLOYE];

        nbreEmploye = 0;
    }

    public void ajouterEmploye(Employe e) {

        ++nbreEmploye;
        if (nbreEmploye <= MAXEMPLOYE) {

            staff[nbreEmploye - 1] = e;
        } else {

            System.out.println("Pas plus de " + MAXEMPLOYE + " employés");
```

```
        }
    }

    public double salaireMoyen() {

        double somme = 0.0;
        for (int i = 0; i < nbreEmploye; i++) {

            somme += staff[i].calculerSalaire();
        }

        return somme / nbreEmploye;
    }

    public void afficherSalaires() {

        for (int i = 0; i < nbreEmploye; i++) {

            System.out.println(staff[i].getNom() + " gagne "

                    + staff[i].calculerSalaire() + " francs.");

        }
    }
}


class Salaires {

    public static void main(String[] args) {

        Personnel p = new Personnel();
        p.ajouterEmploye(new Vendeur("Pierre", "Business", 45, "1995", 30000));

        p.ajouterEmploye(new Representant("Léon", "Vendtout", 25, "2001", 20000));

        p.ajouterEmploye(new Technicien("Yves", "Bosseur", 28, "1998", 1000));

        p.ajouterEmploye(new Manutentionnaire("Jeanne", "Stocketout", 32, "1998", 45));
```

```
        p.ajouterEmploye(new TechnARisque("Jean", "Flippe", 28, "2000", 1000));

        p.ajouterEmploye(new ManutARisque("Al", "Abordage", 30, "2001", 45));


        p.afficherSalaires();
        System.out.println("Le salaire moyen dans l'entreprise est de "

                + p.salaireMoyen() + " francs.");

    }


}
```

Ville

1 Créer une classe Ville correspondant a un nom de ville et un nombre d'habitant.
Ces villes doivent pouvoir donner accès a ces informations (nom, nb d'habitant).
Ces villes ont un constructeur permettant de créer une ville avec un nom de ville.
2 Créer une classe Test qui utilise la classe Ville et utilisera les ArrayList. La fonction main comprendra les lignes suivantes :
public static void main(String [] args){
ArrayList<Ville> mesVilles = new ArrayList<Ville>();
mesVilles.add(new Ville("Lille"));
mesVilles.add(new Ville("Calais"));

3. Rentrez 5 villes différentes. Créez une boucle for pour appeler la méthode toString() de toutes les villes

4. Surchargez le constructeur de la classe Ville. Définissez un constructeur, à deux arguments le nom de la
ville et le nombre d'habitant
Créez une classe Capitale qui hérite de la classe Ville. Celle-ci comprendra une variable d’instance supplémentaire : nomPays. .
Testez différents cas : appel explicite ou non au constructeur de la classe mère ; existence ou non d’un constructeur sans arguments.

6. Redéfinissez la méthode toString(), en faisant appel à la méthode de la classe mère. (toString()) qui affichera à l’écran Capitale de nomPays: nomVille ; nbHabitants). Testez.
7. Changez les modificateurs d’accès des données membres de la classe mère, en remplaçant private par protected. Peut-on accéder à ces variables depuis l’extérieur de la classe Ville ?

```
public class Ville {

 // les attributs de la ville
 private int nbHabitants;
 private String nomVille;

 // constructeurs de la classe Ville
 public Ville(String v){
 nomVille = v;
```

```
 nbHabitants = 0;
 }
 public Ville(String v, int k){
 nomVille = v;
 nbHabitants = k;
 }
 // les méthodes de la classe Ville
 public String getNomVille(){
 return nomVille;
 }
 public int getNbHabitants(){
 return nbHabitants;
 }
 public void setNbHabitants(int k){
 nbHabitants = k;
 }
 public String toString(){
 String resultat;
 resultat = "ville : " + nomVille + "\thabitants : " + nbHabitants;
 return resultat;
 }
}
```

Capitale:

```
public class Capitale extends Ville {
 // la classe Capitale hérite de la calsse Ville
 private String nomPays; // attribut supplémentaire de la classe Capitale
 // constructeur de la classe Capitale à 3 paramètre, pays, ville et nombre d'habitants
 public Capitale(String np, String nv, int nh){
 /*--- ne fonctionne pas car nomVille et nbHabitants sont privés
 nomVille = nv;
 nbHabitants = nh;
 */
 super(nv, nh); // appel au constructeur de la classe mère
 nomPays = np;
 }
 // les méthodes de Capitale
 public String toString(){
 String resultat;
```

```
 resultat = "Capitale de " + nomPays + " " + super.toString();
 return resultat;
 }
}
```

Media

Soit un media, un livre et un DVD. Un livre et DVD sont des cas particuliers de media.

- la classe Media définit un attribut ’titre’ et une méthode ’toString()’
- la classe Livre hérite de la classe Media (c’est un média particulier) et définit l’attribut nombre de page et redéfinit la méthode toString()
- la classe DVD hérite de la classe Media (c’est un média particulier) et définit l’attribut ’durée’ et
  redéfinit la méthode ’toString()’

Que doit donner le code suivant:

```
public class MediaTest {
 public static void main(String [] args){
 Media o1 = new Media("Le Figaro");
 Media o2 = new Livre("java head first",450);
 Media o3 = new DVD("home",120);
 System.out.println(o1);
 System.out.println(o2);
 System.out.println(o3);



 }
}
```

Correction

```
public class Media {
 private String titre;
 public Media(String unTitre){
 titre = unTitre;
 }
 public String getTitre(){
```

```
 return titre;
 }
 public String toString(){
 return "Media " + titre;
 }
}
```

```
public class Livre extends Media {
 private int nombreDePages;
 public Livre(String unTitre, int n){
 super(unTitre);
 nombreDePages= n;
 }
 public String toString(){
 return "Livre " + getTitre() + " " + nombreDePages + " pages";
 }
}
```

```
public class DVD extends Media {
 private int duree;
 public DVD(String unTitre, int n){
 super(unTitre);
 duree = n;
 }
 public String toString(){
 return "DVD " + getTitre() + " " + duree + " minutes";
 }
}
```

Aeroport

Cet exercice calcule le seuil de rentabilité des vols d’une compagnie aérienne.

Un aéroport est détermine par :
• Un nom de code (chaîne de caractères).
• Une longitude x (entier).
• Une latitude y (entier).
Ecrivez une classe Aeroport munie des attributs necessaires.
Ecrivez un constructeur prenant en param`etres les donnees necessaires a son initialisation.
Ecrivez un accesseur getNom du nom de code de l’aeroport.

Ecrivez une méthode distance(aero2) qui calcule et renvoie la distance d qui separe deux aeroports :

sqrt((x2-x1)^2 +(y2-y1)^2)

```
import java.lang.Math;
public class Aeroport {
private String m_nom; // nom de code
private int m_x; // longitude
private int m_y; // latitude
public Aeroport(String n, int x, int y){
m_nom = n;
m_x = x;
m_y = y;
}
public String getNom(){
return m_nom;
}
public double distance(Aeroport aero2){
double dx = (m_x - aero2.m_x);
double dy = (m_y - aero2.m_y);
return Math.sqrt(dx*dx + dy*dy);
}
}
```

Pour simplifier, on suppose que le coˆut d’un vol est independant du nombre de passagers transportes : il est uniquement fonction de la capacite k de l’avion, du nombre de membres d’equipage m et de la distance d entre les aeroports selon la formule :
100*d*sqrt(k+m)+50*sqrt(d)
Un vol sera donc constitue de :
• Un nom (chaine de caract`eres).
• Le nombre maximum de passagers k (entier positif) que l’avion peut transporter.
• Le nombre de membres d’equipage m (entier positif).
• La distance d (reel) entre les aeroports de depart et de destination

```
import java.lang.Math;
public class Vol {
private String m_nom; // nom du vol
private int m_k; // capacite (# de passagers)
private int m_m; // # de membres d’equipage
private double m_d; // distance
public Vol(String n, int k, int m, double d){
m_nom = n;
m_k = k;
m_m = m;
m_d = d;
}
public double eval(){return 100 * m_d * Math.sqrt((double)(m_k + m_m)) + 50 * Math.sqrt(m_d);
}
}
```

La compagnie a´erienne se pose la question suivante :
« Les billets ´etant vendus PRIX_BILLET euros, combien de passagers faut-il au minimum pour que le vol soit rentable ? ».
Votre programme doit :
• Instancier deux Aeroports et les initialiser avec les a´eroports de Mulhouse et de New-York.
• Calculer et afficher la distance entre les a´eroports.
• Instancier le Vol correspondant.
• Calculer et afficher le coˆut du vol.
• Demander le prix du billet.
• Calculer et afficher le nombre de passagers minimum pour que le vol soit rentabilis´e.

```
Distance entre MLH et JFK = 46.6154
Cout du vol = 82151.2
Prix du billet? 400
Nombre de billets a vendre = 206
```

```
import java.util.Scanner;
import java.util.Locale;
import java.lang.Math;
public class PGCompagnie{
public static void main(String[] args) {
Scanner cin = new Scanner(System.in);
cin.useLocale(Locale.US);
Aeroport a1 = new Aeroport("MLH", 2, 46);
Aeroport a2 = new Aeroport("JFK", 40, 73);
double d = a1.distance(a2);
System.out.println("Distance entre "+a1.getNom()
+" et "+a2.getNom()
+" = "+d);
Vol vol = new Vol("EPF 123", 300, 8, d);
double c = vol.eval();
System.out.println("Cout du vol = "+c);
System.out.print("Prix du billet? ");
double prixbillet = cin.nextDouble();
int np = (int)(Math.ceil(c / prixbillet));
System.out.println("Nombre de billets a vendre = "+np);
}
}
```

Aeroport2

Exercice 1 : Horaires.

On appelle horaire une donnée caractérisant un moment dans une journée, indépendamment de la date de
cette journée. Un horaire permet par exemple de préciser le moment du départ d'un train : "le train de
17h42 " où 17h42 représente l'horaire du train évoqué. Nous allons représenté un horaire par une instance
de la classe Horaire.
Une instance de Horaire est donc représentée par la donnée de deux nombres : le premier représente l'heure
(entre 0 et 23) et le second les minutes (entre 0 et 59).

La classe Horaire implémente l'interface Comparable en réalisant la relation d'ordre intuitive sur ces horaires. Elle propose en plus les méthodes suivantes :
• les accesseurs pour les heures et les minutes ;
• la méthode equals, deux horaires étant égaux s'ils ont mêmes heures et mêmes minutes ;
• une méthode

public int ecart(Horaire h)
dont le résultat est l'écart en minutes entre l'objet horaire invoquant la méthode et l'horaire h passé
en paramètre. Cette méthode déclenche une exception IllegalArgumentException si l'horaire h est plus tôt ("dans la journée") que l'horaire invoquant.
Q 1 . Donnez le code java de la méthode equals.
Q 2 . Le code java de la méthode ecart.
Q 3 . Pour toutes les méthodes de votre classe Horaire autres que les accesseurs et le constructeur donnez le code des méthodes de test que vous proposez.

Exercice 2 : Aéroport et vols.

Dans cet exercice nous utiliserons la classe util.Horaire définie à l'exercice précédent

On s'intéresse à la représentation de vols entre aéroports.

Les vols sont supposés être tous journaliers et il
s'agit de "vols courts" donc les horaires de départ et d'arrivée sont toujours dans la même journée. On ne
s'occupe donc pas des jours de départ, seul l'horaire compte.

Vols.

Un vol est caractérisé par un numéro unique (une chaîne de caractères), un horaire de départ et un horaire d'arrivée (de type Horaire), les aéroports de départ et de destination. Voici le diagramme de
la classe avion.Vol qui permet de représenter ces données :

```
Vol
- numero : String
- depart : Aeroport
- dest : Aeroport
- heureDepart : Horaire
- heureArrivee : Horaire
+ Vol(...)
+ getNumero() : String
+ getDepart() : Aeroport
+ getDestination() : Aeroport
+ getHeureDepart() : Horaire
+ getHeureArrivee() : Horaire
+ equals(o : Object) : boolean
+ hashCode() : int
```

Aéroports.

Un aéroport est caractérisé par un identifiant unique (une chaîne de caractères) et une table de hachage qui associe pour chacun des vols qui partent de cet aéroport, le numéro de vol à l'objet vol
correspondant.

Deux objets Aeroport sont donc égaux s'ils ont le même identifiant (on supposera que dans ce cas la liste des vols est nécessairement la même pour les deux objets).

Q 1 . Donnez le code java de l'entête de la classe avion.Aeroport ainsi que la déclaration de ses attributs
et de son constructeur sachant qu'initialement la table des vols est vide.

Q 2 . Donnez le code d’une méthode ajouteVol qui prend en paramètre un vol et l’ajoute à cet aéroport.
Cette méthode déclenche une IllegalArgumentException si cet aéroport n’est pas l’aéroport de départ de ce vol.

Q 3 . Donnez le code java d’une méthode volsDirects qui prend en paramètre un aéroport destination
dest. Cette méthode retourne la liste des vols qui partent de cette instance d’aéroport et dont la destination est dest.

Q 4 . Donnez le code java d’une méthode prochainVolDirect qui prend en paramètre un aéroport destination dest et un horaire h et dont le résultat est le numéro du premier vol de cette instance d’aéroport
qui part dans la journée pour dest après l’horaire h fourni. Une exception NoSuchElementException est déclenchée si aucun vol ne convient.

Donnez le code java du corps de la méthode suivante :

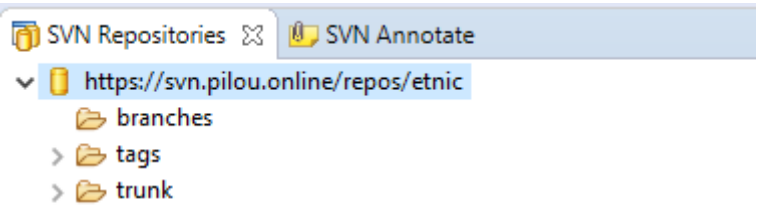
```
/** renvoie true ssi le prochain vol direct à destination de dest
* part dans moins de delai minutes de cet aéroport
* @param dest la destination
* @param delai le délai maximal avant le départ du prochain vol pour dest
* @return true si le prochain vol direct à destination de dest
* part dans moins de delai minutes de cet aéroport et
* false si ce n’est pas le cas ou s’il n’y pas de prochain vol direct
*/
public boolean volEnPartance(Aeroport dest, int delai) {
...
}
```

Q 7 . Donnez le code java d’une méthode volsParDestination dont le résultat est la table de hachage
qui pour cet (this) aéroport associe à chaque aéroport a la liste des vols directs partant de this dont a est la destination.

SVN

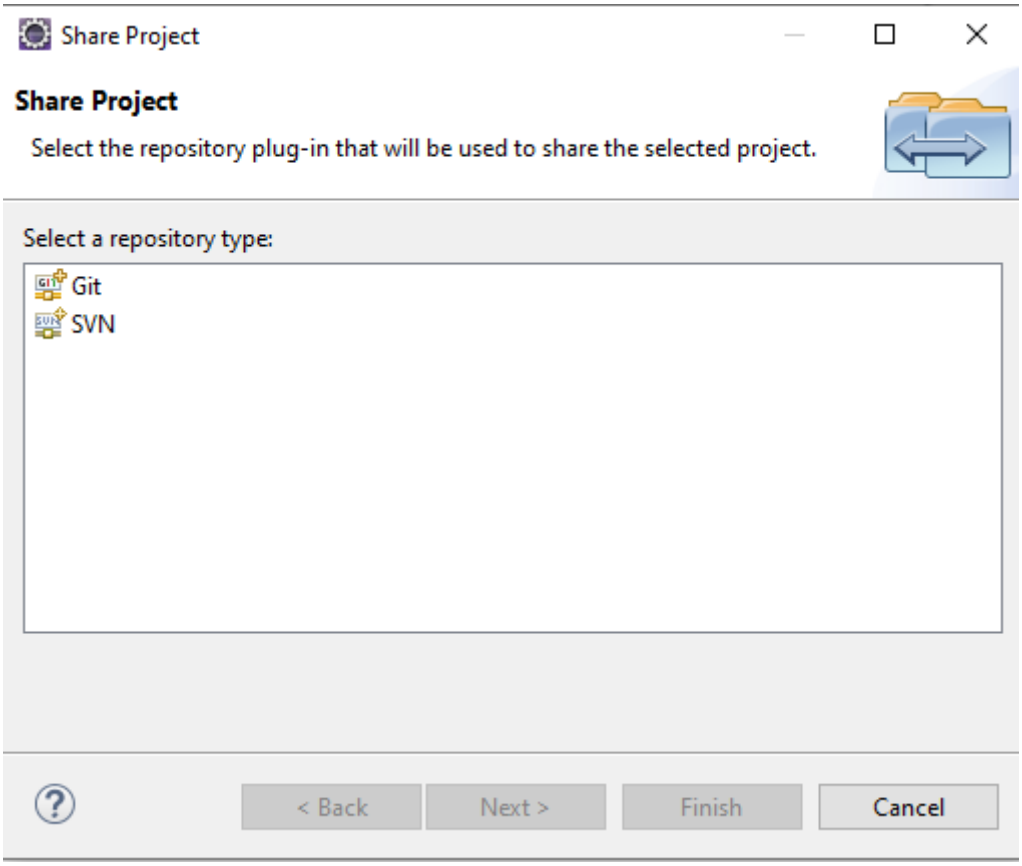
Liste des repositories

Il est possible dans eclipse de créer un ensemble de liens vers des repot.

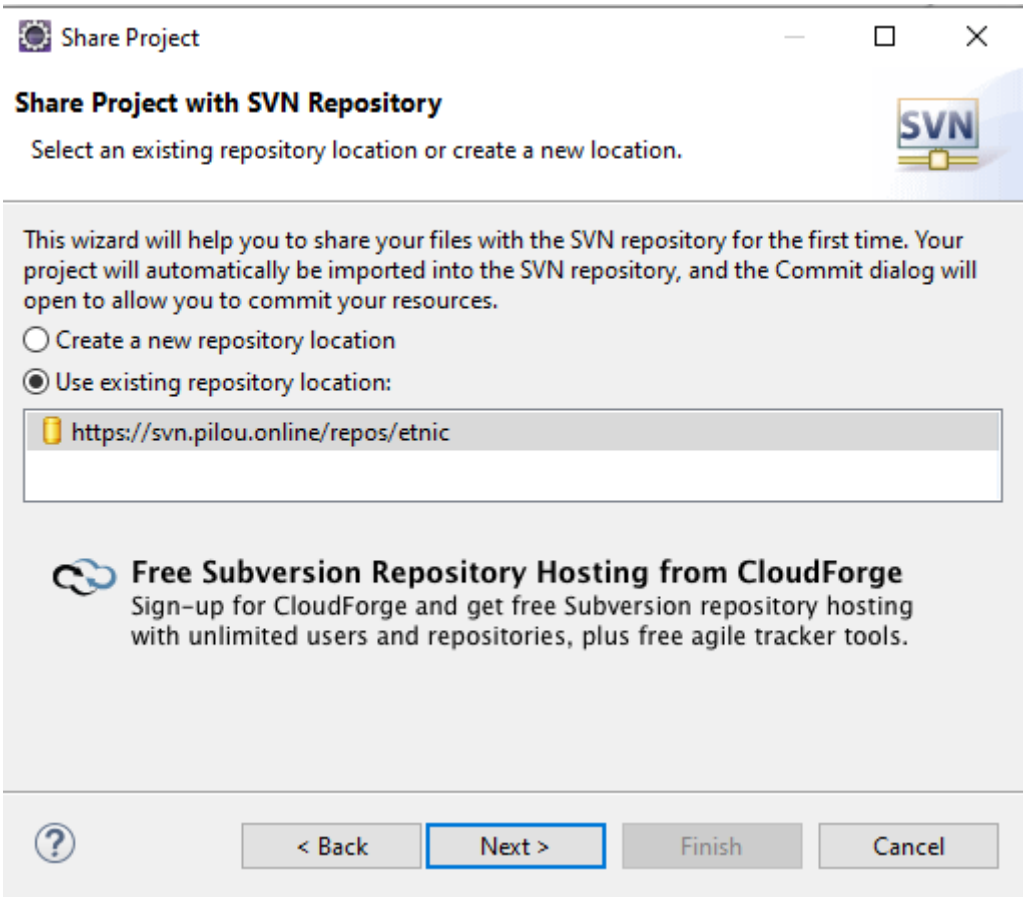


Partages d'un projet

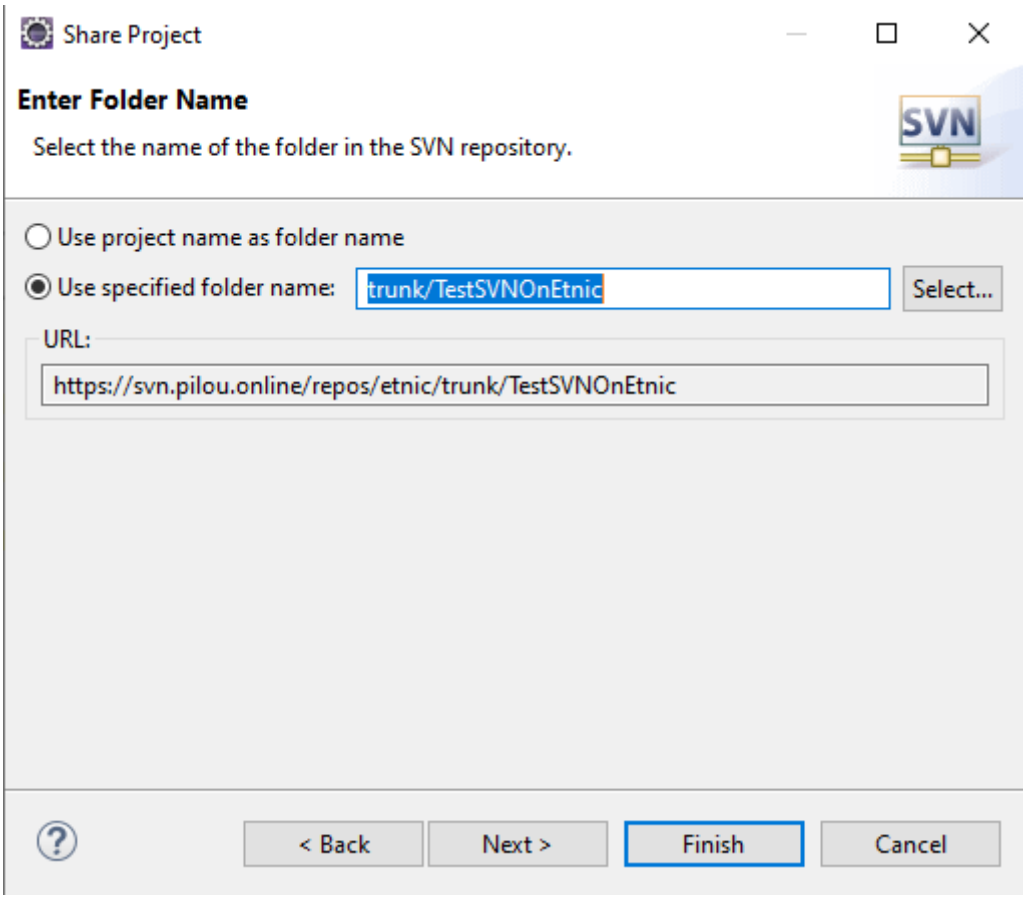
Il s'agit du checkout du projet



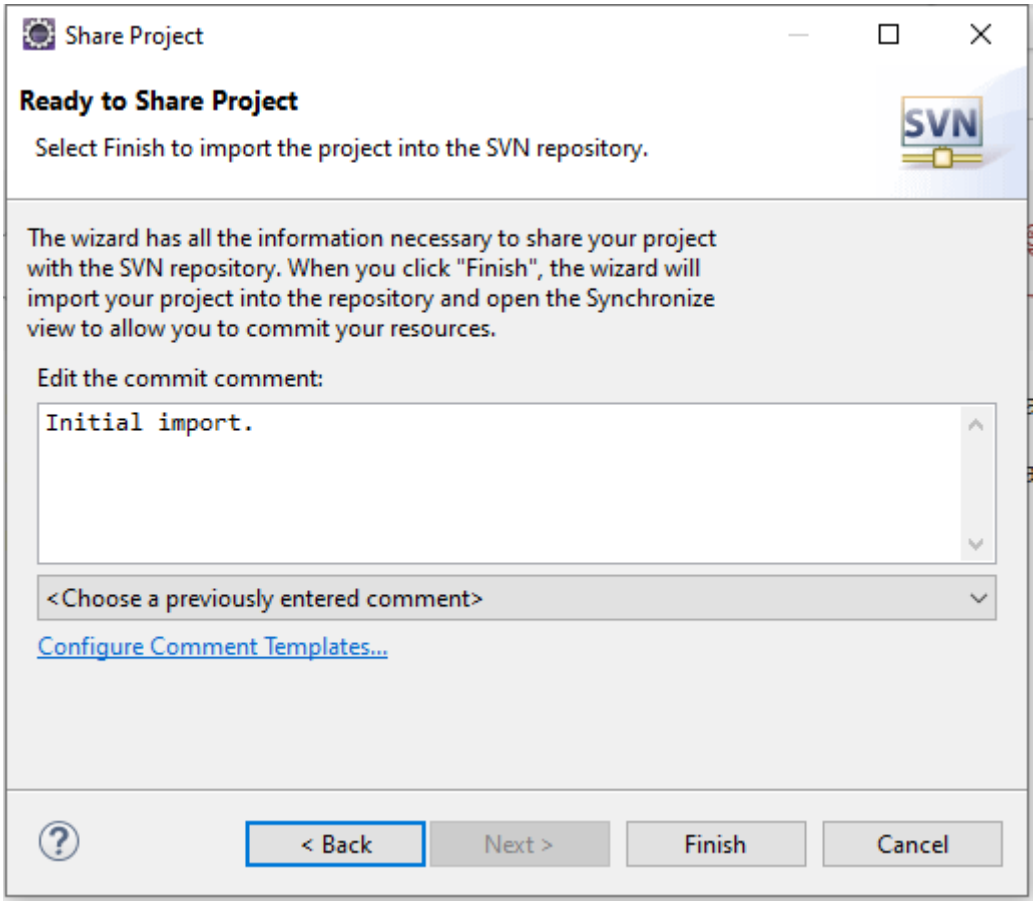
Selection du repository



Une bonne pratique est de pousser le projet vers un repertoire trunk

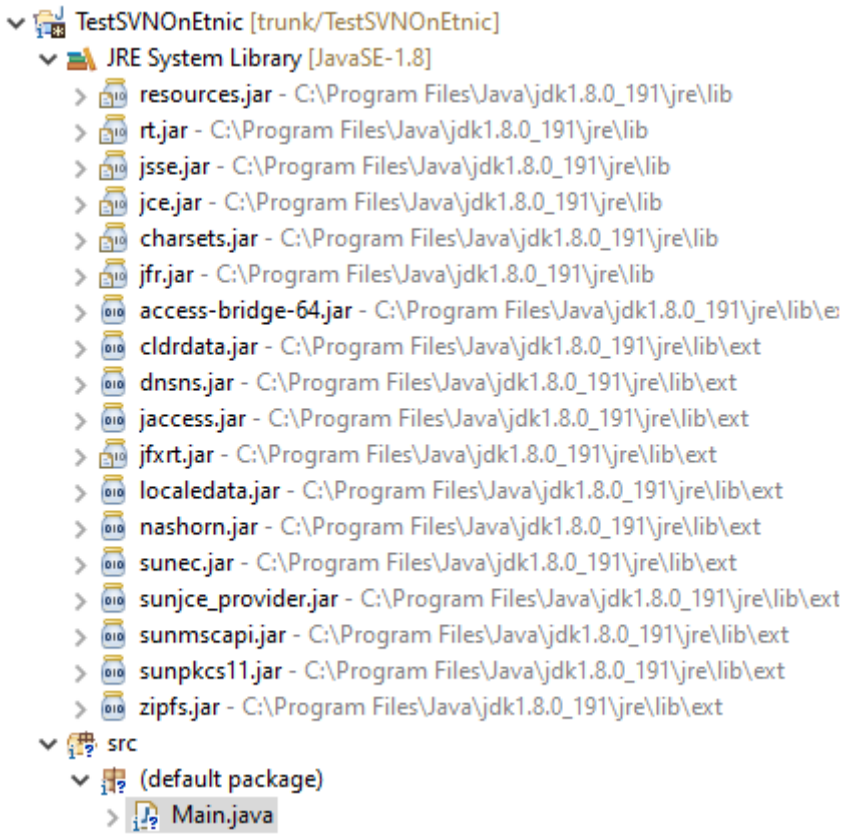


Puis on rajoute un commentaire



Puis il faut rajouter le login et le password

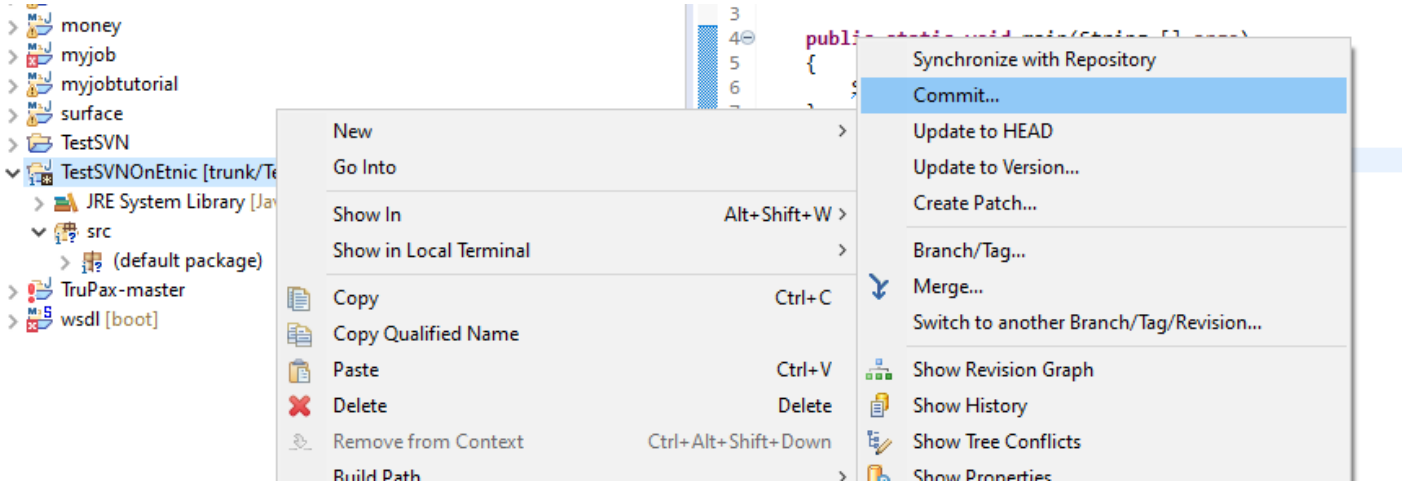
L'etat du projet dans Eclipse est ainsi:



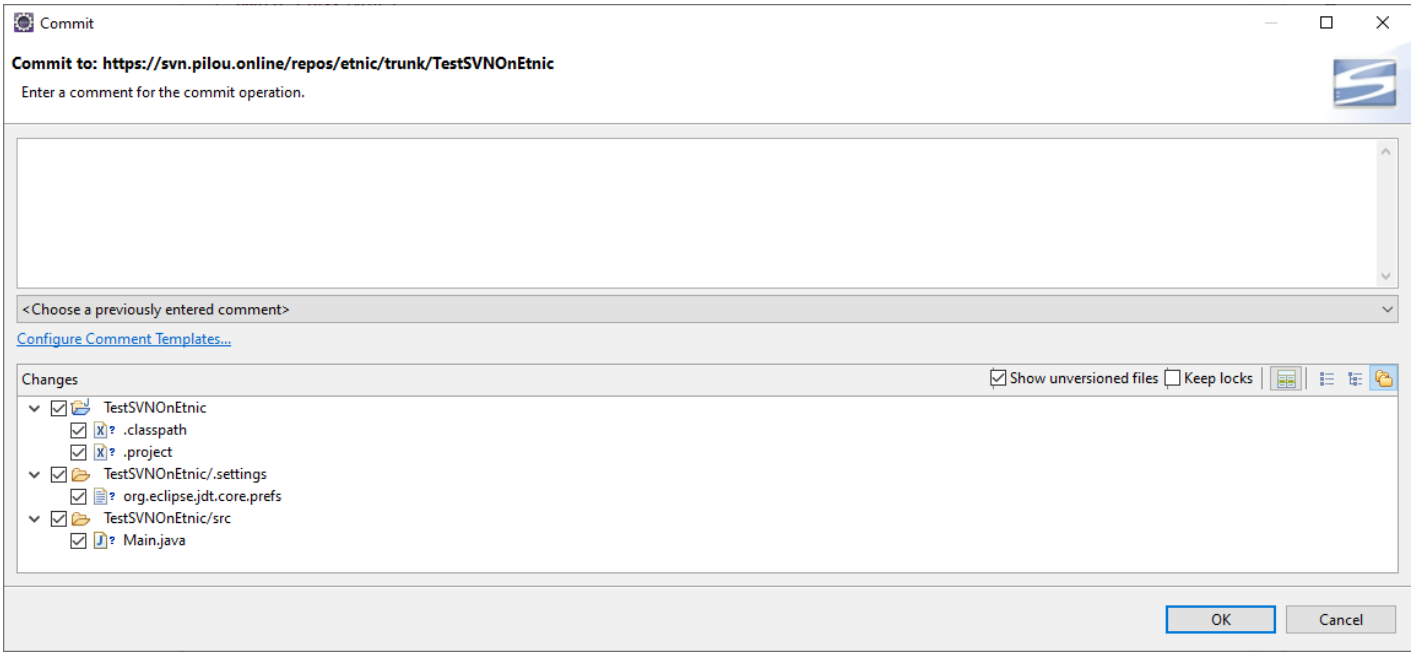
On noteras le ? signalant que le fichier n'est pas dans SVN.

Premier Commit

Il est possible de faire un commit ainsi:

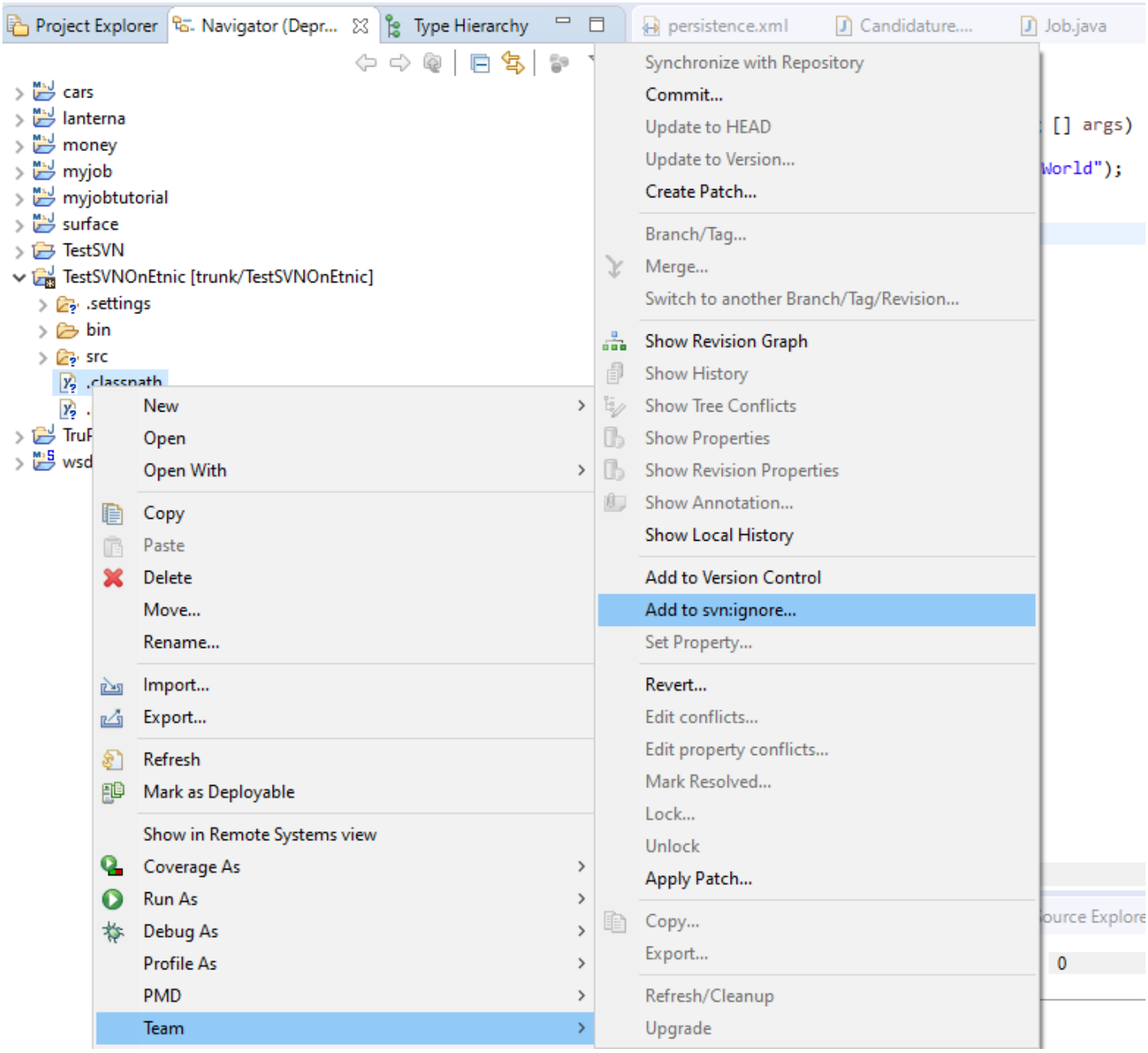


Eclipse propose la liste des fichiers a mettre dans le repo

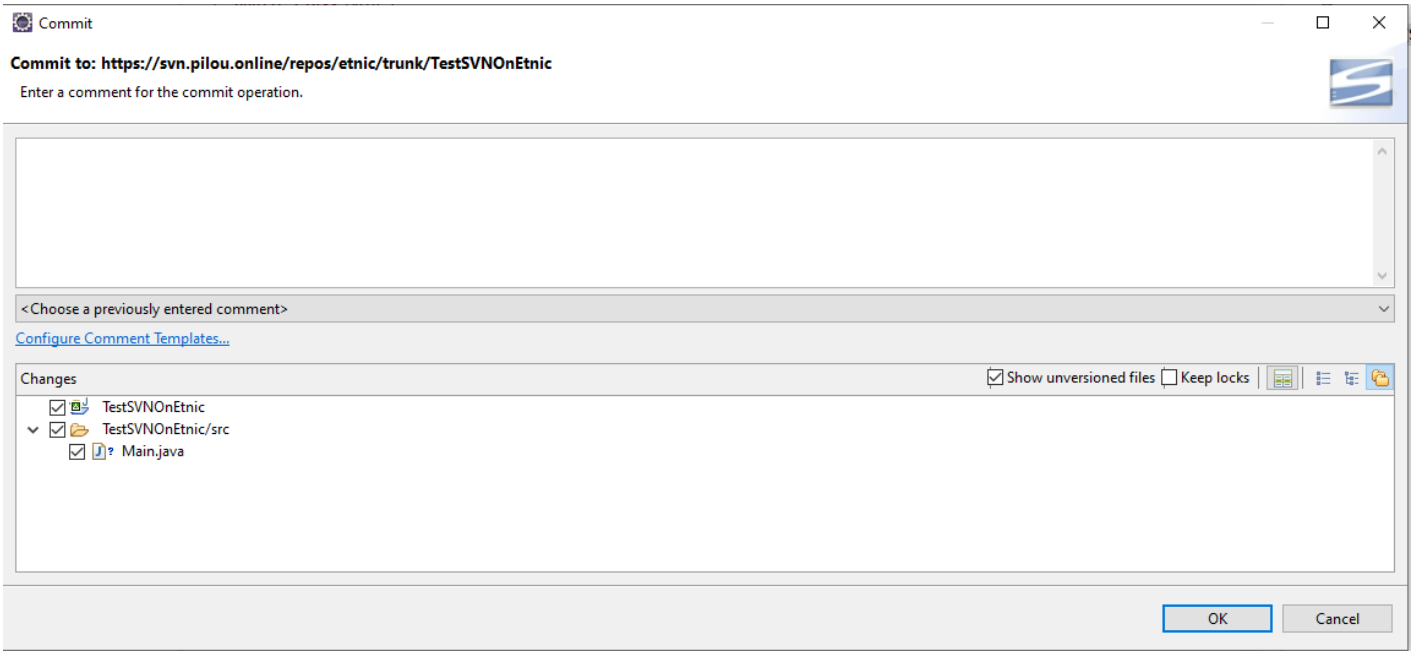


Le hic des meta donnée eclipse n'ont rien a faire ici

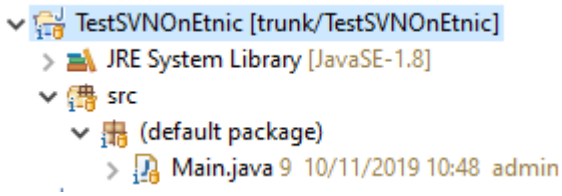
Dans la perspective navigator, on sort les fichiers qui n'ont rien a faire de SVN:



Ainsi lors du commit, on a maintenant

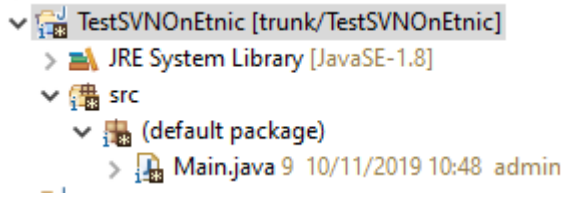


Apres le commit, on voit que le fichier est dans premiere revision par admin

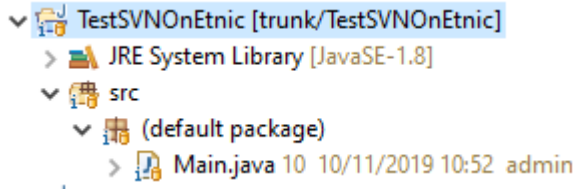


Premiere modification

Lors de la modification du fichier Main.java, on voit que l'iconographie a changer



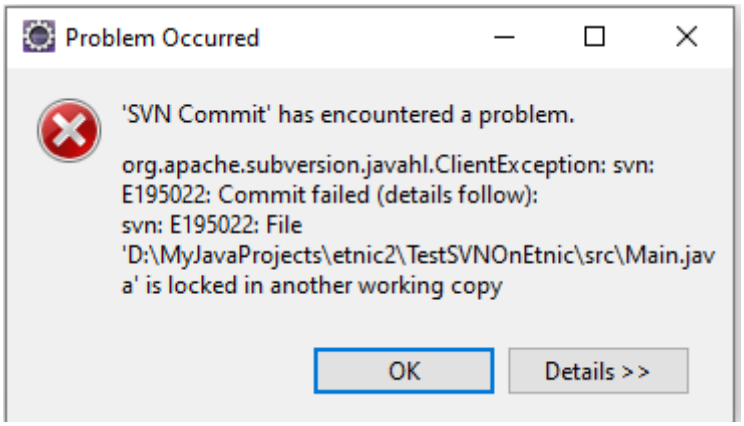
Apres le commit, la revision du fichier a changer



Mise en place d'un lock

Il est possible de positionner un lock sur un fichier

Lors d'un commit sur un fichier locker eclipse refuse le commit

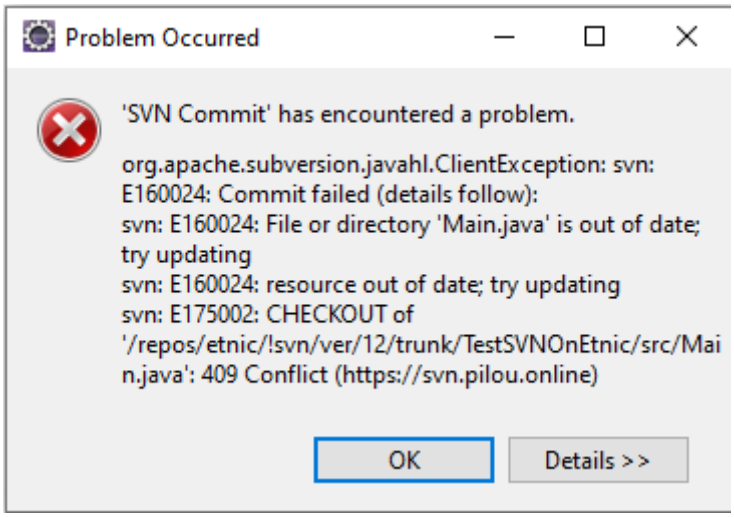


Conflits

Conflit simple

Le conflit arrive lorsque qu'un fichier est edtier par plusieurs utilisteurs

Lors du commit dans Eclipse on a

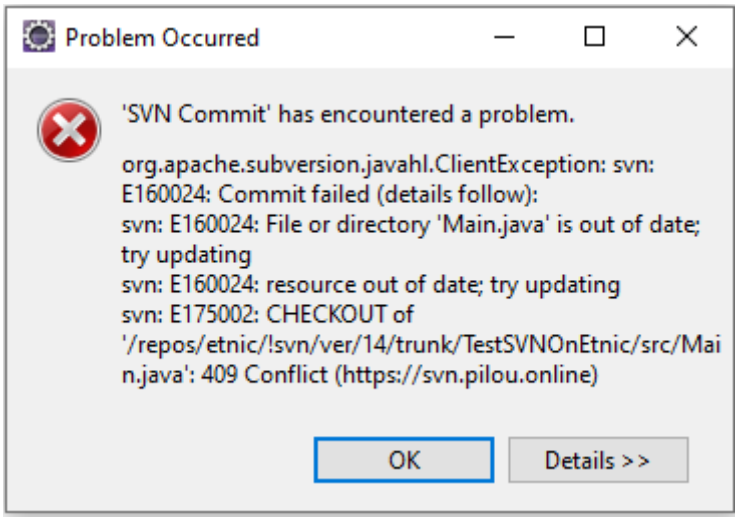


Ce qui signifie qu'il faut faire un update avant

Apres l'update, on peut faire un commit

Conflit avec resolution

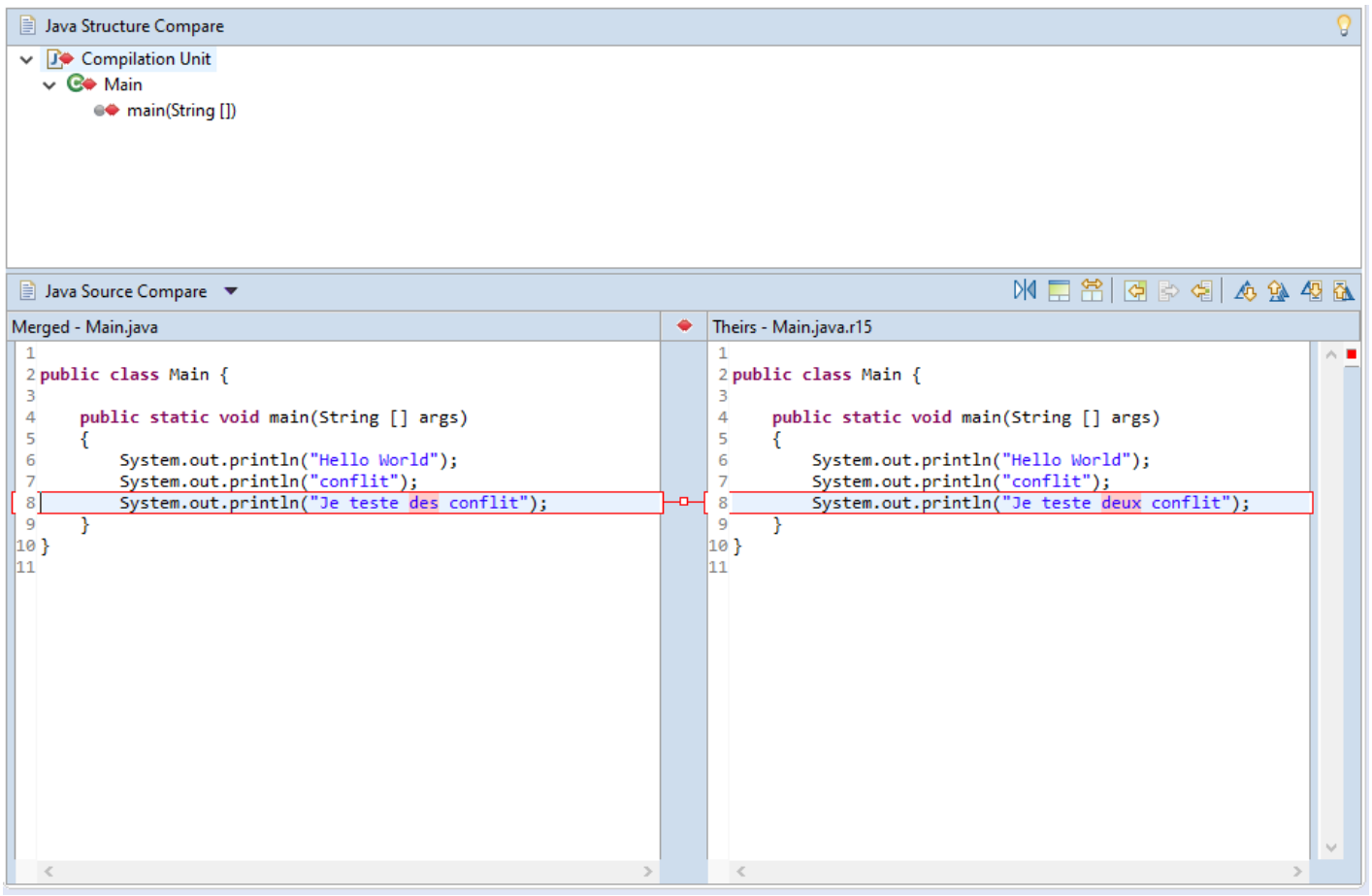
Si le conflit apparait au même endroit cela est moins simple



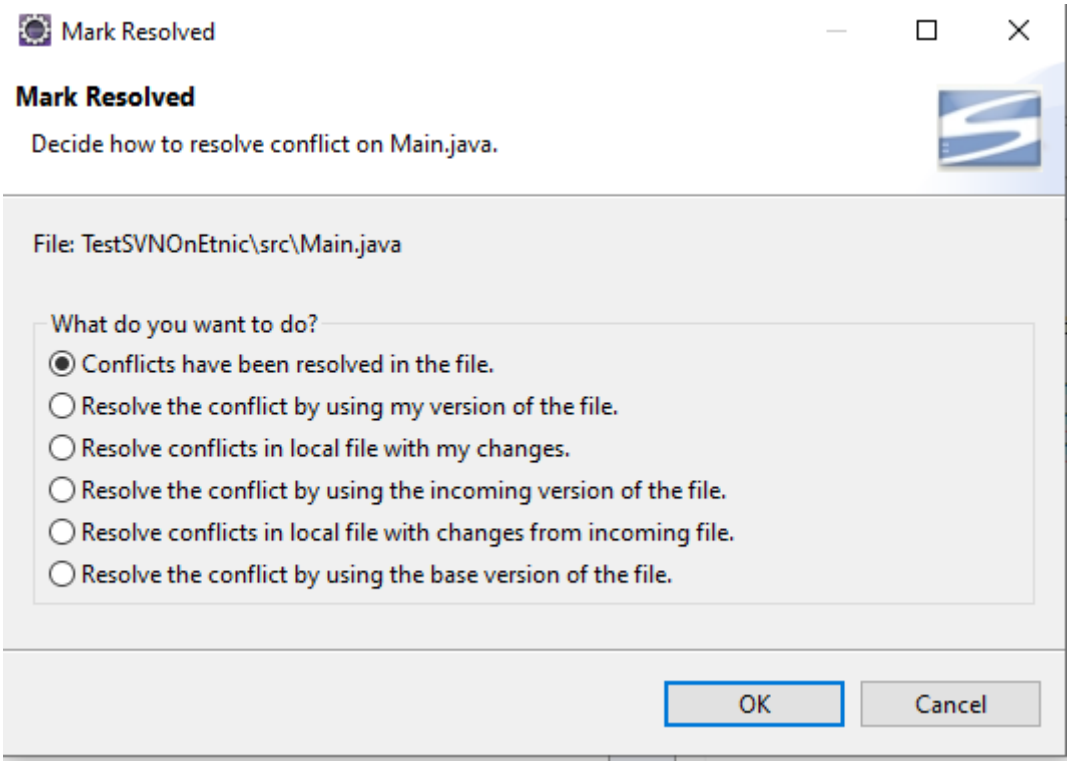
Lors de l'update, Eclipse signale un probleme:

update D:/MyJavaProjects/etnic2/TestSVNOnEtnic/src/Main.java -r HEAD --force
C D:/MyJavaProjects/etnic2/TestSVNOnEtnic/src/Main.java
Updated to revision 15.
==== Conflict Statistics: =====
File conflicts: 1

Ce qui donne lors de l'édition des conflits:



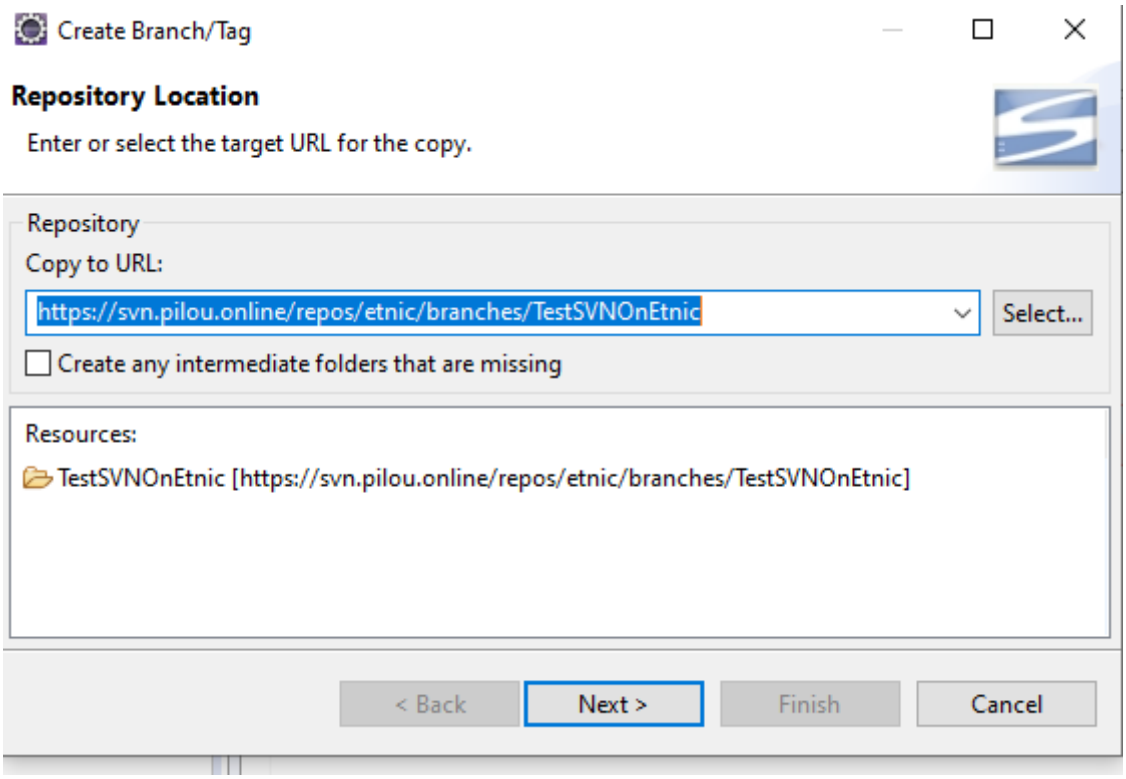
Lorsque le conflit est résolue, on marque le conflit comme étant résolue



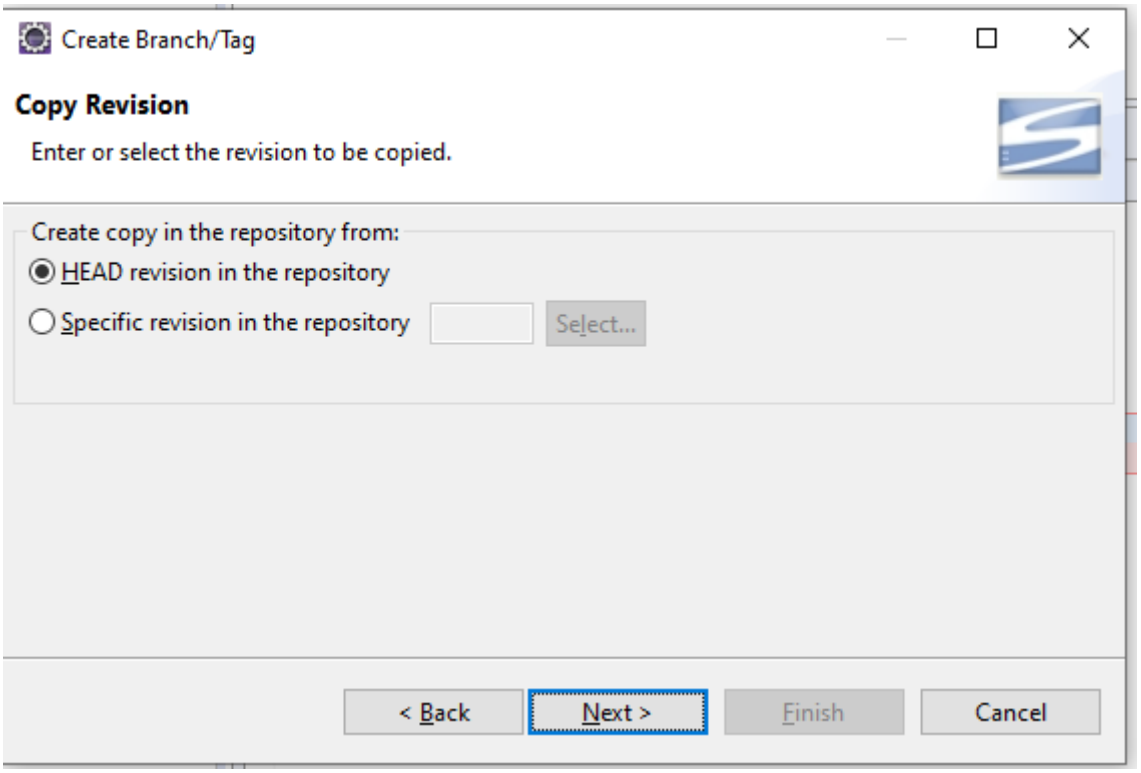
Puis on commit

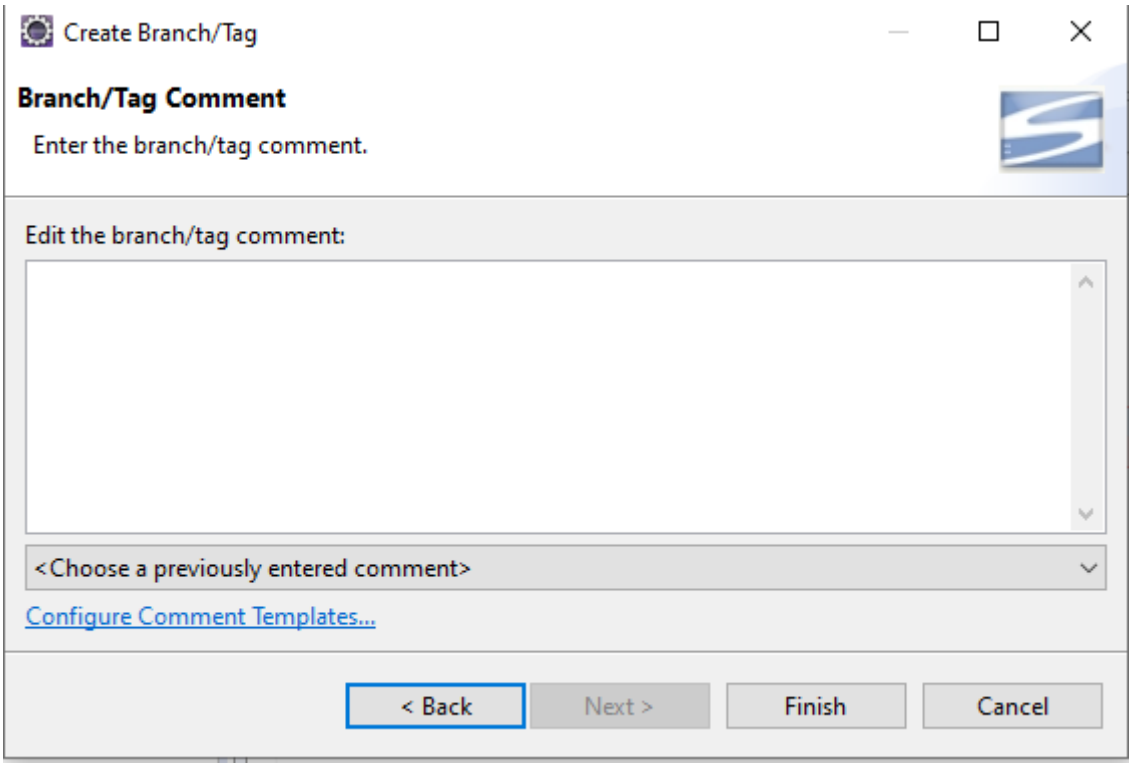
Création de la branche

Créer la branche comme un repertoire

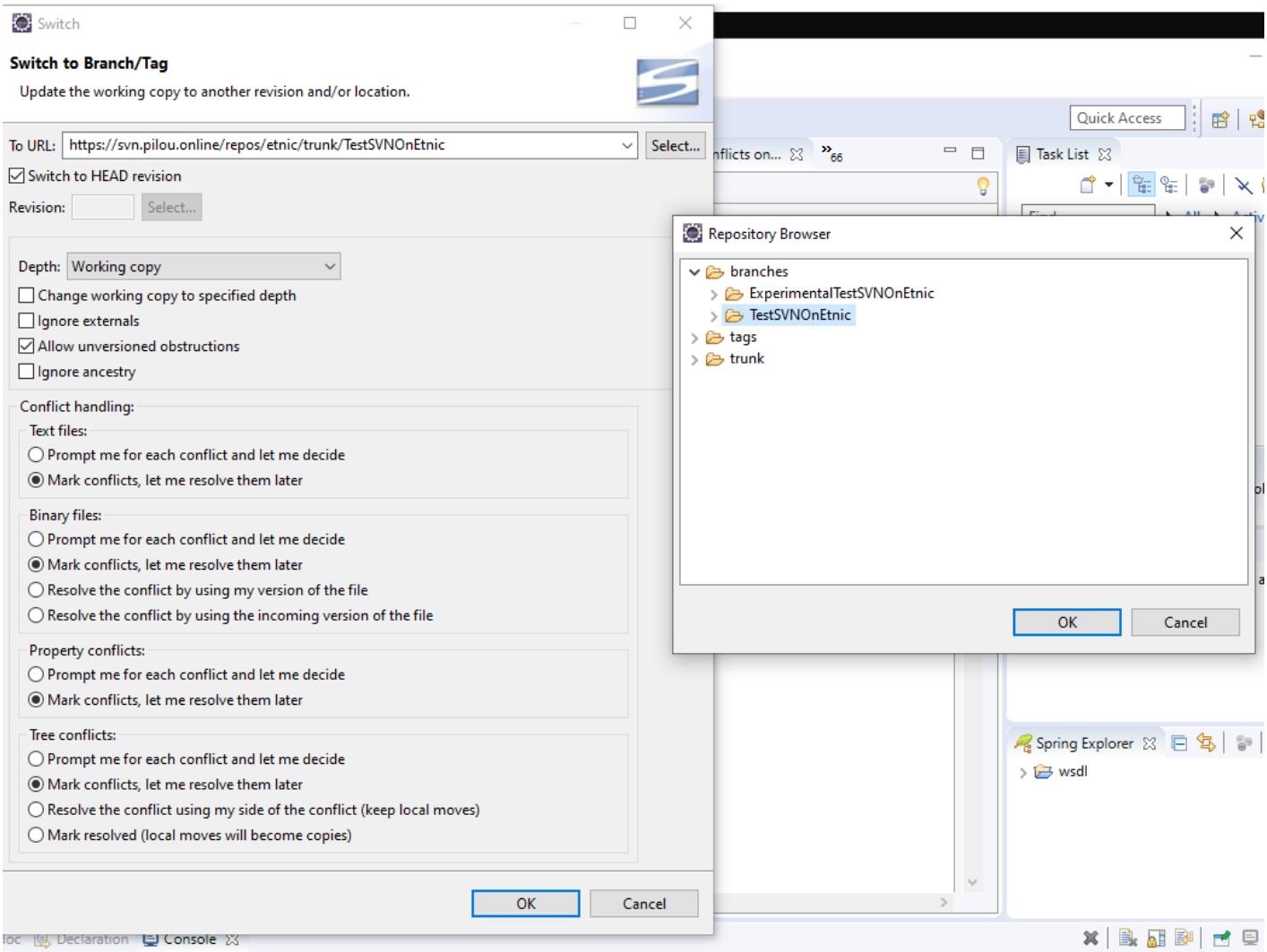


Cliquez avec le bouton droit sur le dossier de la ligne réseau, sélectionnez Equipe> Branche / Tag. Copier vers l'URL:

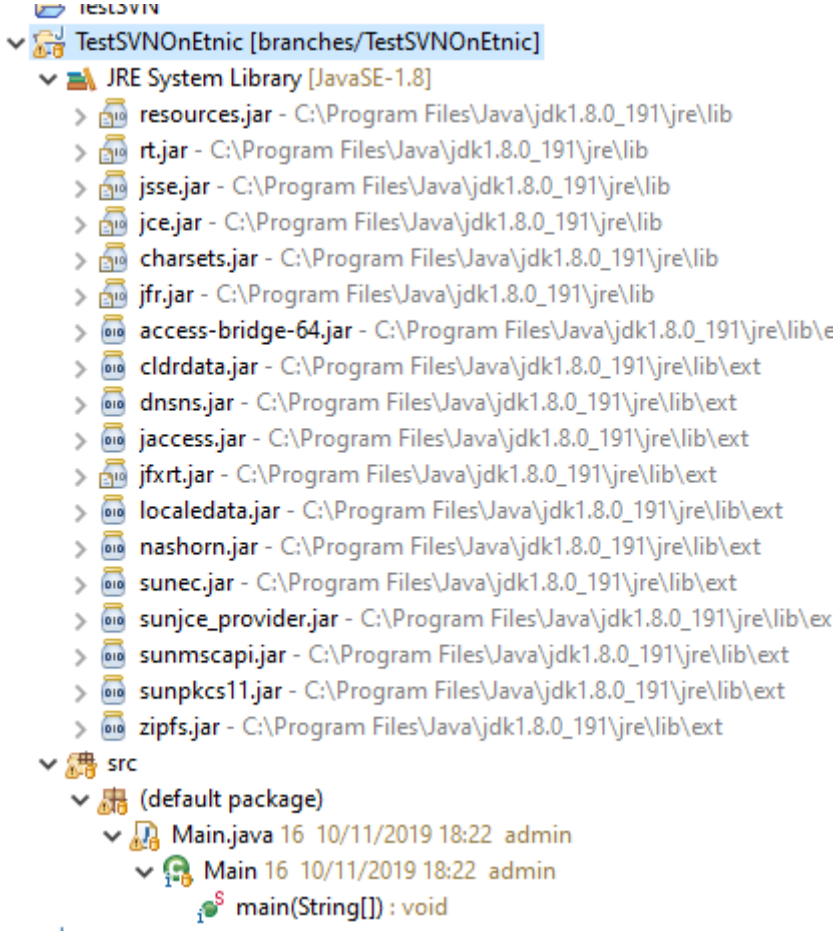




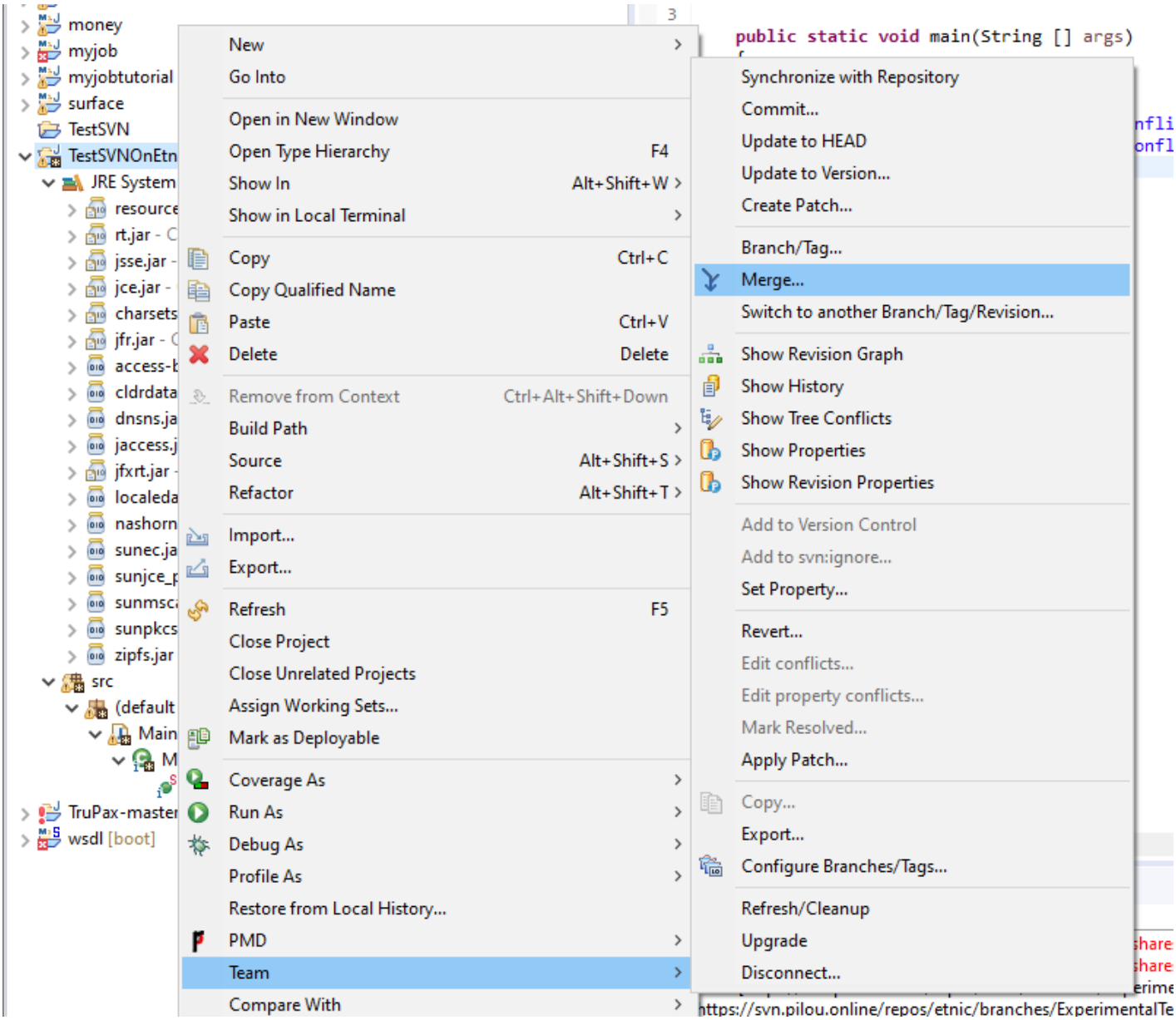
Puis changer de branches



Eclipse l'indique ensuite



Fusion du tronc vers la branche, ou vice versa



Cliquez avec le bouton droit de la souris sur la branche ou le coffre, sélectionnez Equipe> Fusionner. Choisissez Fusionner une plage de révisions si la fusion va du tronc à la branche. Sinon, sélectionnez Réintégrer une branche.

CollabNet Merge

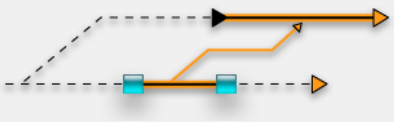
Select the merge type

Specify the type of merge to perform

Merge input

- (•) Merge a range of revisions
- ( ) Change-set based merge
- ( ) Merge two different trees
- ( ) Manually record merge information (block one or more revisions)
- ( ) Manually remove merge information (unblock one or more revisions)

Merge a range of revisions



Use this method to catch-up a feature branch with the changes in trunk or another branch. You can merge a specific set of revisions or all eligible revisions.

[x] Perform pre-merge best practices checks

< Back | Next > | Finish | Cancel

CollabNet Merge

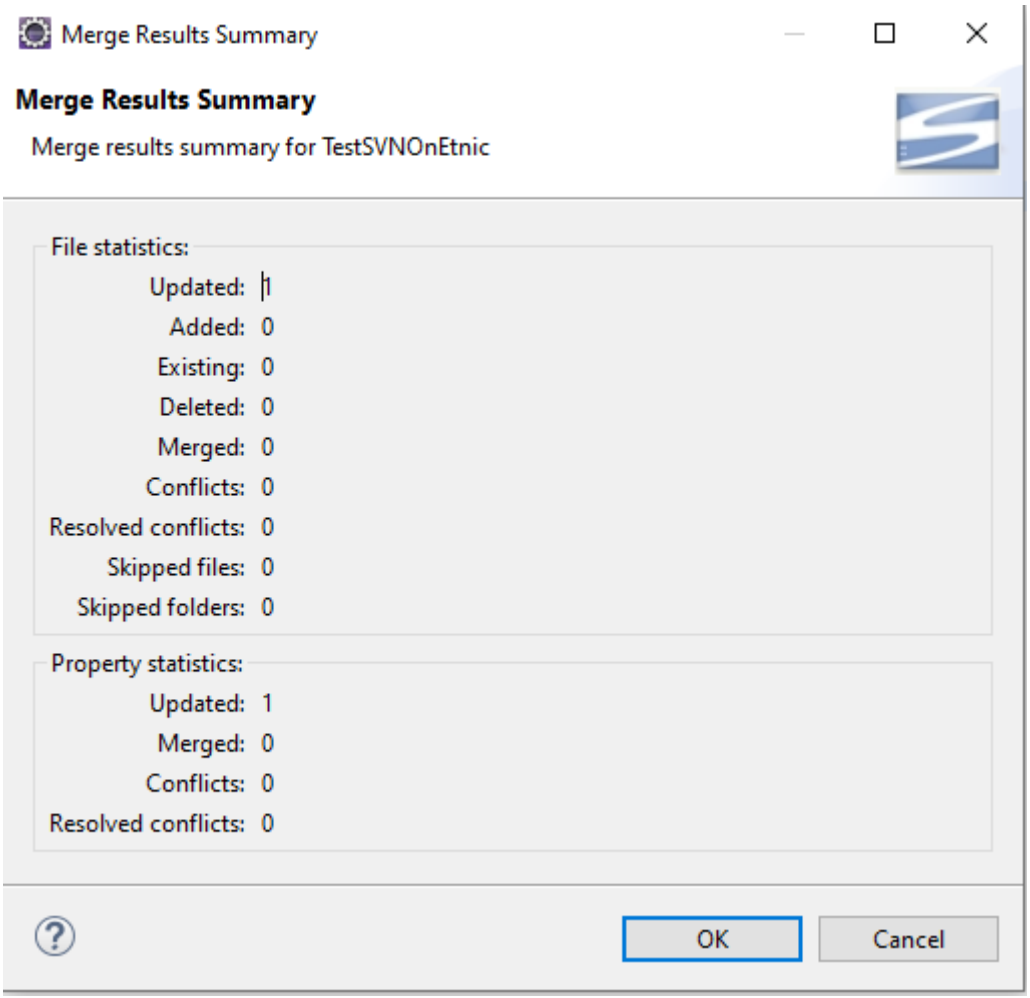
Best Practices

Working copy is ready for merge

- ✓ ▸ No uncommitted modifications
- ✓ ▸ Working copy at a single revision
- ✓ ▸ No switched children
- ✓ ▾ Complete working copy

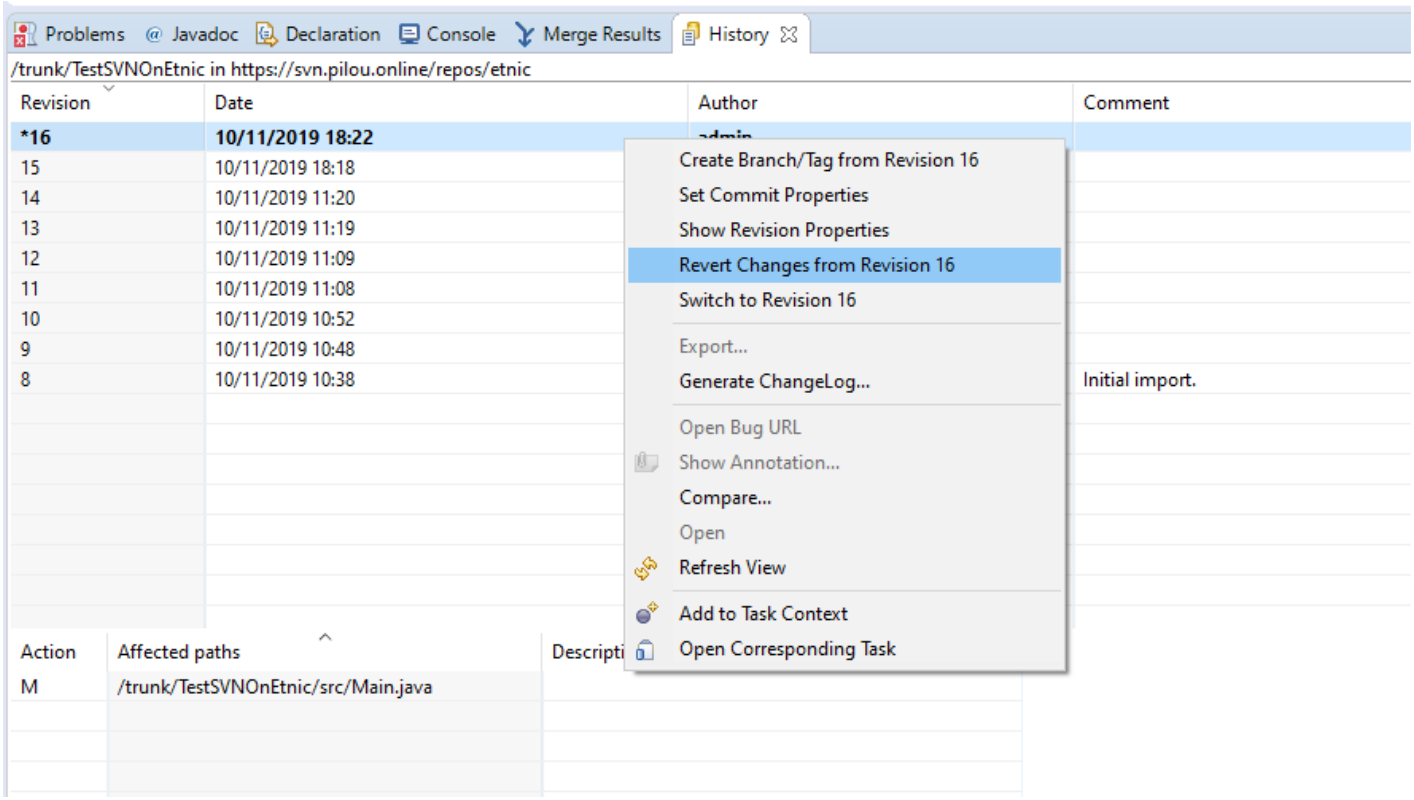
  Working copy should be a complete working copy (depth=infinity). If working copy is not a complete working copy, update the working copy before merging.

< Back | Next > | Finish | Cancel



Annuler un commit

Dans la page SVN History, il est possible d'annuler des changements


Problems
Javadoc
Declaration
Console
Merge Results
History
/trunk/TestSVNOnEtnic in https://svn.pilou.online/repos/etnic
Revision
Date
Author
Comment
*16
10/11/2019 18:22
15
10/11/2019 18:18
14
10/11/2019 11:20
13
10/11/2019 11:19
12
10/11/2019 11:09
11
10/11/2019 11:08
10
10/11/2019 10:52
9
10/11/2019 10:48
8
10/11/2019 10:38
Initial import.
Create Branch/Tag from Revision 16
Set Commit Properties
Show Revision Properties
Revert Changes from Revision 16
Switch to Revision 16
Export...
Generate ChangeLog...
Open Bug URL
Show Annotation...
Compare...
Open
Refresh View
Add to Task Context
Open Corresponding Task
Action
Affected paths
Descripti
M
/trunk/TestSVNOnEtnic/src/Main.java

Slide

Les slides sont ici https://www.pilou.online/files/LJO_ETNIC.pdf