Exercices de Programmation Java (draft)

Laboratoires et TP du cours INF1256

Johnny Tsheke, Ing. Jr.

Université du Québec à Montréal
– UQAM –

2 décembre 2017

Table des matières

Listings

Chapitre 1

Introduction

Dans ce manuel, nous présentons les exercices de Java pour le cours INF1256 – Informatique pour les sciences de la gestions. Par défaut, le masculin est utilisé pour alléger le texte sans aucune considération particulière au genre de la personne qui le lit. Les présentations du cours, les solutions des exercices et d'autres informations sont disponible sur le site du cours : http://inf1256.gitlab.io.

Ce manuel est mis à jour fréquemment. Consulter le site du cours pour avoir la dernière version.

Première partie

Exercices et laboratoires

Chapitre 2

Environnement de développement

Ce chapitre présente brièvement les environnements de développement utilisés dans les séances des laboratoires. Comme ces logiciels sont gratuits, nous expliquerons aussi comment les installer chez soi. Par la suite, nous aborderons l'utilisation d'un gestionnaire de code source pour faciliter les sauvegarde ainsi que la collaboration lors des travaux de groupe.

2.1 Installation

Dans cette section, nous présentons brièvement la procédure d'installation des différents logiciels. Nous y indiquons aussi les sites pour télécharger ces logiciels (gratuits). Il n'est pas nécessaire d'installer tous les logiciels présentés ici. Certains ne sont repris ici qu'à titre d'information. Au laboratoire, nous utilisons `eclipse`.

2.1.1 Installation de Java

L'environnement de développement intégré (IDE) utilisé dans ce cours a besoin de Java pour bien fonctionner. On a donc besoin d'installer Java avant même l'IDE. Il existe plusieurs fournisseurs de Java mais sous suggérons fortement le Java de *Sun* (devenu *Oracle*). On peut téléchager Java gratuitement sur le site suivant :
`http://java.sun.com`
Le mieux serait de prendre un package `JDK`. Sinon, prendre au moins un `JRE` correspondant à votre système d'exploitation

2.1.2 Installation de eclipse

2.1.2.1 Prendre le bon Eclipse

http://www.eclipse.org/ Au laboratoire, nous utilisons eclipse

2.1.2.2 Configuration de la machine virtuelle Java

Cette section se réfère aux informations disponible sur le site http ://-wiki.eclipse.org/Eclipse.ini. Malgré l'installation préalable de la bonne version Java, il arrive parfois qu'Eclipse refuse de démarrer. Parfois le problème vient du fait qu'il ne trouve pas la machine virtuelle Java. Pour cela il faut ajouter dans la configuration d'éclipse un paramètre lui permettant de trouver la machine virtuelle Java. Pour se faire, on peut procéder de la manière suivante :

— Vérifier que la bonne version Java est installé. Pour cela, il suffit d'ouvrir un terminal (Sous Windows l'inviter de commande) et taper la commande suivante :
`java -version`

— Si la bonne version Java est installée, il faut localiser le dossier d'installation de Java puis déterminer le chemin vers les fichiers suivants selon votre systèmes d'exploitation.

Windows : `bin\javaw.exe`

Linux : `bin/java`

Mac

2.1.3 Installation de netbeans

https://netbeans.org/

2.1.4 Installation de Bluej

Pour les débutants
http://www.bluej.org/

2.1.5 Installation de git

Pour la gestion de code source
https://git-scm.com/

2.2 Initiation à git

2.3 Initiation à l'utilisation d'un IDE

Dans cette section nous allons présenter brièvement l'utilisation de quelques IDE.

2.3.1 Initiation à Eclipse

2.3.1.1 Création d'un nouveau projet Java

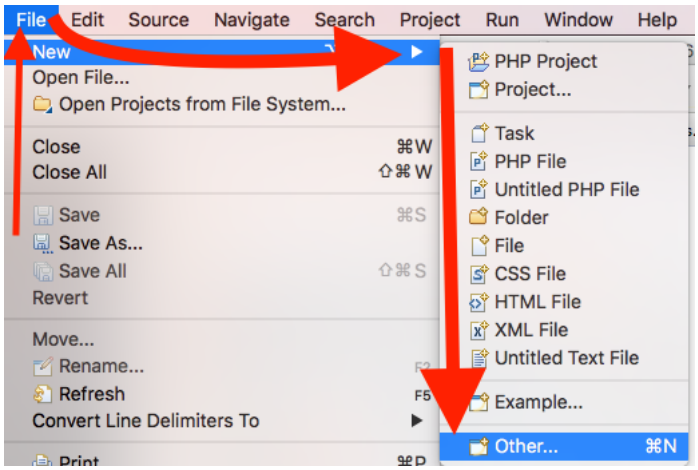

FIGURE 2.1 – Création d'un nouveau Autre ...

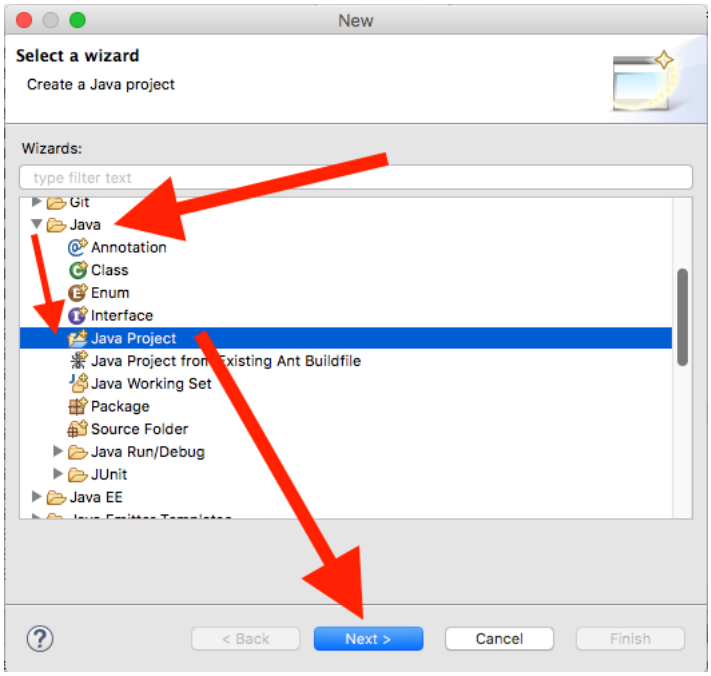

FIGURE 2.2 – Sélection d'un nouveau projet Java

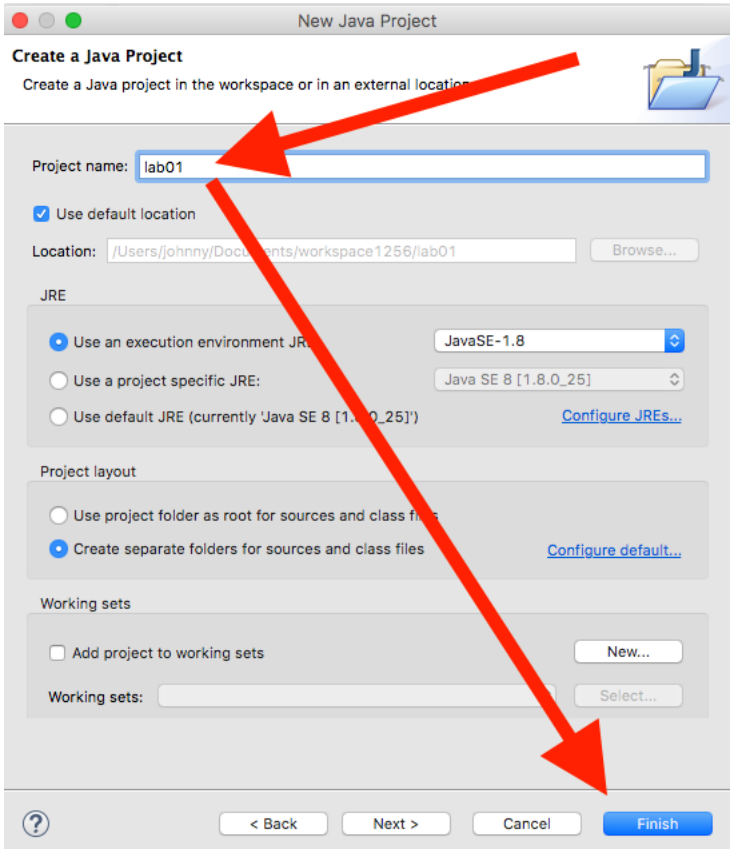

FIGURE 2.3 – Attribution de nom à un nouveau projet Java

Pour créer un nouveau projet Java, on peut procéder la de la manière suivante.

1. Aller sur le menu `Fichier/nouveau` (`File/New`) puis choisir `Autre` (`Other ...`) comme illustré à la figure 2.3

2. Par la suite, sélectionner `Java/Projet Java` (`Java/Java Project`) comme le montre la figure 2.2 et cliquer sur `Suivant` (`Next`)

3. À la fenêtre suivante illustré àa la figure 2.3, donner un nom au projet Java (`Project name`) puis cliquer sur `Terminer` (`Finish`)

2.3.1.2 Ajout d'une classe Java

2.3.1.3 Exécution d'un programme Java

2.4 IDE et gestionnaire de code

2.5 Outils gratuit en ligne

2.5.1 Dessiner des graphes en ligne

https://www.draw.io/

2.5.2 Programmer en ligne

https://repl.it/
https://www.tutorialspoint.com/codingground.htm

Chapitre 3

Programmation générale

3.1 Rappels

Dans cette section, nous donnons quelques petit rappels pour commencer la programmation Java.

- En Java, on met un point-virgule (`;`) après chaque instruction
- Après `//` le reste de la ligne est en commentaire
- Le texte entre `/*` et `*/` est en commentaire
- Une classe définit un type de données et son nom doit commencer par une lettre majuscule
- Habituellement, en Java un programme doit-être dans une classe. Cette classe doit être enregistrée dans un fichier portant le même nom. ex :

```
public class HelloWorld{
 //mettre le corps de la classe HelloWorld ici
 }
```

- Il est recommandé de regrouper les classes dans des espaces de noms appelés `package`
- Le nom d'un`package` doit-être entièrement en minuscule
- Pour ajouter une classe dans un `package`, on ajoute la déclaration du package au tout début du code avant même la déclaration de la classe (habituellement en première ligne)

```
package nompackage;
public class HelloWorld{
 //mettre le programme ici
 }
```

- Pour utiliser les classes d'un autre `package`, il faut les importer avant

la déclaration de la classe mais après la déclaration du `package` de la présente classe

```
package nompackage;
import java.util.*; //classes d'une librairie Java
public class HelloWorld{
 //mettre le programme ici
 }
```

- En programmation Java, une `méthode` est une fonction définie dans une classe
- On écrit les algorithme à l'intérieur des `méthodes`
- Pour exécuter une classe toute seule, il faut y définir une méthode appelée `main` et celle-ci doit-être accessible de partout

```
package nompackage;
import java.util.*; //classes d'une librairie Java
public class HelloWorld{
   public static void main(String[] args) {
       //méthode qui s'exécute par défaut au démarrage
     }
 }
```

- Pour un petit programme, on mettra tout le code dans la méthode `main`
- En Java, pour afficher le message `Bonjour`, on écrit :
  `System.out.println("Bonjour");`
- Pour déclarer une `variable` nommée `nomVariable` qui va contenir un nombre réel, on écrit :
  `Double nomVariable;`
  ou encore
  `double nomVariable;`

- Pour déclarer une variable nommée `nomAutreVariable` qui va contenir une chaine des caractères, on écrit :
  `String nomAutreVariable;`
- Pour déclarer une `variable` nommée `nombre` qui va contenir un nombre entier, on écrit :
  `int nombre;`
- pour affecter une valeur (12.7) à une variable (`nomVariable`) déjà déclarée, on écrit :
  `nomVariable = 12.7 ;`
- Pour entrer des données au clavier, on peut :
  - importer `java.util.*`
    `import java.util.*;`

  - déclarer une variable (`scanne`) de type `Scanner` dans la méthode :
    `Scanner scanne = new Scanner(System.in);`
  - par la suite, pour lire un nombre réel au clavier et l'enregistrer dans une variable `nomVariable` déjà déclarée :
    `nomVariable = scanne.nextDouble();`
  - Pour lire un nombre entier au clavier :
    `nombre = scanne.nextInt();`
  - Pour lire une chaine de caractères :
    `nomAutreVariable = scanne.next();`
  - À la fin des lectures au clavier, fermer le scanner :
    `scanne.close();`
- Pour afficher la valeur d'une variable (`nomVariable`) :
  `System.out.println(nomVariable);`
- Pour afficher un message concaténé avec la valeur d'une variable (`nomVariable`) :
  `System.out.println(" Le nombre est : "+nomVariable);`
- Pour convertir un nombre entier en nombre réel, il suffit de l'affecter à une variable déclarée comme un `double`. ex :

```
int nombreEntier = 1; // vaut 1
// autres instructions éventuelles
double nombreReel = nombreEntier; // Conversion implicite à 1.0
```

- Pour convertir une chaine à une valeur de type énuméré(`enum`), on peut utiliser la méthode `valueOf`. Voici un exemple :

```
public class HelloWorld{
     public enum Salutation{BONJOUR, BONSOIR};
     public static void main(String []args){
         String matin =  "BONJOUR"; //chaine
         Salutation sal = Salutation.valueOf(matin); //conv.
        System.out.println("Hello World");
       System.out.println(sal);
    }
}
```

3.2 Exercices de base

Dans cette section le lecteur trouveras quelques exercices de base

- On demande d'écrire un programme Java (classe `HelloWorld`) qui affiche le message suivant :
  `Hello World!`

Chapitre 4

Laboratoires

Ce chapitre contient les exercices des laboratoires.

4.1 Laboratoire 1

L'objectif de ce laboratoire est de :
— résoudre un problème en écrivant un algorithme en pseudo-code
— se familiariser avec l'environnement de développement (eclipse, Java)
— exécuter et tester un programme écrit en Java

Pour ce laboratoire, on créera sur eclipse, un projet Java nommé `inf1256lab01`.

4.1.1 Conversion de température de °F → °C

On demande de :
— créer une classe Java nommé `ConversionFC` (enregistrer dans le fichier ConversionFC.java) avec la méthode `main`
— écrire sur papier un algorithme en pseudo-code qui converti une température exprimée en degrés Fahrenheit en degrés Celsius. La température à convertir sera demandée à l'utilisateur et le résultat des calculs sera affiché à l'écran
— écrire un programme Java de votre algorithme dans la méthode `main`
— exécuter et tester le programme Java

Pour rappel :

Fusion de l'eau 0 °C = 32 °F

Ébullition de l'eau 100 °C = 212 °F

Échelles : °C → 100, °F → 180

Convertir : x °C en y °F ou vice versa

°C → °F : y °F =((x * (9/5))+32) °F

°F → °C : x °C =((y - 32)* (5/9)) °C

4.1.2 Calcul des taxes TPS et TVQ

On demande de :

— ajouter une classe Java nommée `Taxes` (fichier Taxes.java)

— concevoir un programme qui demande à l'utilisateur le montant total avant taxes. Le programme effectue les calculs et affiche les informations suivantes :

  ◇ le montant avant taxes,

  ◇ Le montant de la TPS (Taxe sur les Produits et Services, -canada-)

  ◇ Le montant de la TVQ (Taxe de Vente du Québec)

  ◇ Le montant total avec taxes

— écrire le programme en Java (dans la méthode `main`)

— exécuter et tester le programme Java

Pour rappel :

- le montant de la TPS égal à 5% du montant avant taxes
- le montant de la TVQ égal à 9.975% du montant avant taxes

Pour calculer 5% d'un montant x, il suffit de multiplier x par 0.05

4.2 Laboratoire 2

L'objectif de ce laboratoire est de :
— se familiariser avec les types de données et les opérateurs
— se familiariser avec l'environnement de développement (eclipse, Java)
— exécuter et tester le programme Java
Pour ce laboratoire, on créera sur eclipse, un projet Java nommé `inf1256lab02`.

4.2.1 Calcul de la valeur future

On demande de :
— ajouter une classe nommée `ValeurFuture` (valeurFuture.java)
— écrire un programme qui fait les opérations suivantes
  ◇ on demande d'entrer le capital (P)
  ◇ on demande le pourcentage du taux d'intérêt (i)
  ◇ par la suite le programme calcule l'intérêt (I) sur une seule période et la valeur future (F) de l'investissement sur cette période. Pour rappel :
    • I=P*i
    • F=P+I ou encore F=P*(1+i)
  ◇ le programme affiche alors les valeurs de P, i, I et F une à une par ligne
  ◇ comme `i` est entré en pourcentage, la valeur à utiliser réellement dans les formules sera i/100.0
  ◇ pour le moment, on ne fait pas encore de validation. On suppose que toutes les données sont bien encodées par l'utilisateur

Suggestions :
□ déterminer le type de données des variables selon le contexte de l'application
□ On peut utiliser une constante pour garder la valeur `100.0`
□ Pour respecter la convention des noms des variables Java, on peut utiliser les noms suivants
  ⋆ `capital` pour `P`
  ⋆ `taux` pour `i`
  ⋆ `interet` pour `I`
  ⋆ `valeurFuture` pour `F`
  ⋆ `CENT` pour la constante `100.0`

4.3 Laboratoire 3

L'objectif de ce laboratoire est de :
- se familiariser avec les types de données et les opérateurs
- se familiariser davantage avec l'environnement de développement (eclipse ou netbeans ou autre)
- améliorer sa programmation Java
- exécuter et tester son programme
- créer un package (ou paquetage) et faire des importations
- afficher les données avec un formatage

Pour ce laboratoire, on créera un projet Java nommé `inf1256lab03` et tous les fichiers développés seront placé dans le package `lab3`.

4.3.1 Conversion d'échelles de distance

On demande d'écrire un programme qui converti en km une distance exprimée en `miles` et vice-versa. Pour rappel :

```
1 mile = 1.609344 Km
```

Cette valeur sera gardée dans une constante (`MILE_TO_KM`).

Pour commencer, votre programme affichera un petit message de bienvenu et une brève description de ce que ça fait. Par la suite, on demandera à l'utilisateur de faire un choix parmi :

1. Conversion de km vers miles
2. Conversion de miles vers km

Selon le choix, on demandera à l'utilisateur d'entrer le nombre de km (choix 1) ou miles (choix 2) à convertir. Le programme effectue les calculs puis affiche le résultat avec 3 chiffres après la virgule. Le nombre saisie par l'utilisateur sera affiché comme tel.

Votre programme devra veiller aux erreur de saisie des données. Ainsi, si on demande un nombre et l'utilisateur entre autre chose, le programme devra afficher un message d'erreur approprié avant de se terminer proprement.

Suggestions :
- importer le package `java.util.*`
- utiliser la classe Scanner pour lire les données au clavier
- utiliser `System.out.format('format',arguments)` pour formater l'affichage

4.4 Laboratoire 4

L'objectif de ce laboratoire est de :
— écrire un programme Java sans définir d'autre méthodes que main
— se familiariser avec les structures répétitives (`for`, `while`)
— s'initier à l'utilisation des instructions de branchement (`break`, ...)
Pour ce laboratoire, on créera un projet Java nommé `inf1256lab4` et on mettra toutes les classes dans un package nommé `lab04`

4.4.1 Somme de n premiers nombres entiers avec `for`

On demande de créer une classe Java `CalculSomme` et d'écrire un programme qui a le comportement suivant.

Étpe 0 Au démarrage le programme initialise les variables et les constantes. On déclarera notamment la constante suivante qui fixe le nombre entier maximum pour lequel on peut calculer la somme des n premiers nombres entiers.

```
ENTIER_MAX = 1000;
```

Étape 1 Le programme affiche un message de bienvenu et explique que ça calcule la somme des n premiers entiers puis passe à l'étape suivante

Étape 2 À cette étape, le programme demande à l'utilisateur d'entrer un nombre entier inférieur ou égal à `ENTIER_MAX` pour calculer la somme. Si l'utilisateur saisit :
— un nombre entier inférieur ou égal à `ENTIER_MAX`, alors on l'enregistre dans la variable `nombre` et on passe à l'étape 3(étape de calcul)
— autre chose alors on affiche un message d'erreur et on passe à l'étape 5 (Fin du programme)

Étape 3 On déclare une variable `somme` de type entier qu'on initialise à 0. Par la suite, en utisant la boucle `for`, on calcule la somme (`somme`) des nombres allant de 0 à la valeur de la variable `nombre` saisit à l'étape précédente.

Étape 4 À cette étape, on affiche le résultat des calculs sous la forme suivante :
La somme de ***`nombre`*** *premiers nombres entiers est égale à* ***`somme`***
EX : si `nombre = 4` alors la somme `somme = 10` ($0 + 1 + 2 + 3 + 4$)

Étape 5 A cet étape on affiche simplement : `Fin du programme`

4.4.2 Caisse enregistreuse simple avec `while`

Dans cet exercice, on veut simuler une caisse enregistreuse simple utilisée dans les épiceries. On demande de :
— Ajouter une classe Java `CaisseSimple`
— Déclarer les variables et les constantes nécessaires et programmer la classe selon la description suivante :

- ◇ Afficher un message de bienvenu et expliquer à l'utilisateur que le programme s'arrête normalement s'il tape le mot `FIN`.
- ◇ Entrer dans la boucle :
  - □ demander de saisir le prix du prochain article ou `FIN` pour terminer. On entre un produit à la fois. Si le client achète plusieurs instances d'un même article (ex : 3 Pains), on entre autant de fois le prix (Ex : on entre 3 fois 2.56 pour 3 pains)
  - □ Si l'utilisateur entre le mot `FIN` alors on met à jour les variables nécessaires pour arrêter la boucle (Pour que la condition devienne fausse – `false`)
  - □ Sinon si l'utilisateur entre nombre réel (y compris les nombres entiers), alors on additionne ce montant à la somme précédente et on continue la boucle
  - □ Sinon si l'utilisateur entre tout autre chose, alors on affiche un message d'erreur et on sort immédiatement de la boucle (instruction `break`)

  Suggestion : utiliser des expressions régulières et les méthodes `hasNext(pattern)` `et hasNextDouble` de la classe `java.util.Scanner` pour déterminer ce qui est saisit au clavier
- ◇ Afficher le total (toutes les taxes sont incluses dans les prix de vente) à payer et le message de fin du programme

4.5 Laboratoire 5

L'objectif de ce laboratoire est de :
— se familiariser avec les structures répétitives et sélectives
— Se familiariser avec les fonctions de la classe `Math`

Pour ce laboratoire, on créera un projet Java nommé `inf1256lab5` et on mettra toutes les classes dans un package nommé `lab05`

4.5.1 Tirage au sort

La directrice de votre service aimerait faire organiser un tirage au sort pour offrir un cadeau à un des employés qui participeront au diner annuel. Pour ce faire, chaque employé recevra un jeton portant un numéro compris entre 500 et 535. Compte tenu des réservations, on sait que tout les jetons seront distribués.

On demande de :
— ajouter une classe nommée `TirageAuSort`
— utiliser une ou plusieurs méthodes de la classe `Math` (`java.lang.Math`) pour effectuer ce tirage ce au sort
— afficher le numéro tiré

4.5.2 Validation des codes postaux

On demande de :
— ajouter une classe nommé `CodesPostaux`
— déclarer 5 constantes locales avec les valeurs des codes postaux canadiens suivants :
  ◇ `"H3A 3Z1"`,
  ◇ `"H2A 1L1"`,
  ◇ `"H8T 3N1"`,
  ◇ `"H2A 5T1"`,
  ◇ `"H1B 8Q7"`
— afficher les codes postaux
— demander à l'utilisateur d'entrer un code postal et le valider pour accepter seulement un des codes énumérés ci-dessus.
— afficher le code postal accepté

Suggestion :
— écrire un pattern pour ces codes postaux. Comme on a un nombre réduit, le pattern sera de la forme suivante
  `cp1|cp2|cp3|cp4|cp5`
— Comme par défaut la lecture d'une entrée se fait jusqu'à l'espace blanc et que le code postal doit contenir un espace blanc, alors il faut changer le délimiteur pour le caractère de fin de ligne de la manière suivante avant de commencer la lecture des codes postaux.

```
Scanner clavier = new Scanner(System.in);
clavier.useDelimiter("\n");
```

4.6 Laboratoire 6

L'objectif de ce laboratoire est de :
— écrire un programme Java à partir d'un pseudo code
— se familiariser avec les structures répétitives et sélectives
— se familiariser avec les méthodes prédéfinies de la classe Math

Pour ce laboratoire, on créera un projet Java nommé `inf1256lab6` et on mettra toutes les classes dans un package nommé `lab06`

4.6.1 Calcul de l'impôt fédéral du Canada pour 2015

On demande d'ajouter une classe Java et de programmer l'algorithme [1] 1 dans la méthode `main`

Algorithme 1 : Calcul de l'impôt fédéral pour 2015

début
  $impFed \leftarrow 0$ `/* Initialisation impôt fédéral */`;
  $trancheSup \leftarrow 0$;
  $tranche1 \leftarrow 0$;
  $tranche2 \leftarrow 6705.00$;
  $tranche3 \leftarrow 16539.00$;
  $tranche4 \leftarrow 29327.00$;
  $revenu \leftarrow$ *demander et entrer le revenu*`/* On entre un nombre */`;
  si $revenu < 0$ **alors**
    Afficher un message d'erreur
  sinon si $0 <= revenu < 44701$ **alors**
    $impFed \leftarrow tranche1 + (15\% \quad de \quad revenu)$;
  sinon si $44701 <= revenu < 89401$ **alors**
    $trancheSup \leftarrow revenu - 44701.00$;
    $impFed \leftarrow tranche2 + (22\%\ de\ trancheSup)$;
  sinon si $89401 <= revenu < 138586$ **alors**
    $trancheSup \leftarrow revenu - 89401.00$;
    $impFed \leftarrow tranche3 + (26\% \quad de \quad trancheSup)$;
  sinon
    $trancheSup \leftarrow revenu - 138586.00$;
    $impFed \leftarrow tranche4 + (29\% \quad de \quad trancheSup)$;
  fin
  si $revenu >= 0$ **alors**
    Afficher que le brut est egal a $revenu$;
    Afficher que le l'impot federal est egal a $impFed$;
  fin
fin

1. inspiré de http://www.cra-arc.gc.ca/tx/ndvdls/fq/txrts-fra.html

4.6.2 Valeurs futures pour plusieurs périodes

On demande d'ajouter une autre classe Java et de programmer l'algorithme 2 dans la méthode `main`

Algorithme 2 : Calcul des valeurs futures sur plusieurs périodes

début
  $PMAX \leftarrow 5$`/* nombre des périodes */`;
  $periode \leftarrow 0$`/* période Initiale  */`;
  $valeurInitiale \leftarrow$ *demander et entrer le montant*`/* nombre réel */`;

  $taux \leftarrow$ *demander et entrer le tauxd'intrt*`/* nombre réel */`;
  Afficher $valeurInitiale$, $taux$, $PMAX$;
  tant que $periode <= PMAX$ **faire**
    $valeurFuture \leftarrow valeurInitiale \times (1 + taux)^{periode}$;
    Afficher $periode$, $valeurFuture$;
    $periode \leftarrow periode + 1$;
  fin
fin

Suggestion :
— importer `java.lang.Math`
— utiliser $Math.pow(a, b)$ pour calculer a^b (Cette méthode retourne un double)

4.6.3 Diagrammes des algorithmes

Ceci est un exercice supplémentaire et facultatif à faire chez soi :
— Écrire un organigramme de l'algorithme 1 de la page 30
— Écrire un organigramme de l'algorithme 2 de la page 31

4.7 Laboratoire 7

L'objectif de ce laboratoire est de :
— écrire un programme Java avec des méthodes
— créer un jar exécutable (`Runnable Jar File`)
— déployer la jar exécutable et l'exécuter en commande en ligne
Pour ce laboratoire, on créera un projet Java nommé `inf1256lab07`.

4.7.1 Programme et méthodes

On demande de faire la programmation suivante.
— créer une classe nommée Java `CaisseEpicier`
— ajouter les méthodes suivantes :
  ◇ `double scannePrix()` ne reçoit rien en argument. Elle demande à l'utilisateur de saisir le prix du prochain article. Si l'utilisateur entre autre chose qu'un nombre (réel ou entier), on lui repose à nouveau la question jusqu'à ce qu'il entre une réponse valide. Cette méthode retourne le nombre valide saisi.
  ◇ `void afficheFacture(double tauxTps, double tauxTvq, double prix)` ne retourne rien. calcule et affiche les informations suivantes
    • le montant avant taxes,
    • le montant de la TPS,
    • le montant de la TVQ,
    • le montant total avec taxes,
— programmer la méthode `main` avec les fonctionnalités suivantes
  ◇ déclarer la variable suivante qui **permettra d'appeler les méthodes non statique à partir de la méthode statique** `main`
  ```
  CaisseEpicier ce = new CaisseEpicier();
  ```
  ◇ déclarer et initialiser les variables suivantes
  ```
  double txTPS = 0.05; //taux de la taxe TPS
  double txTVQ = 0.0975; //taux de la taxe TVQ
  double prixArticle = 0.0; //prix de l'article
  ```
  ◇ demander à l'utilisateur de saisir le prix de l'article et affecté le prix saisi à la variable `prixArticle`. Pour cela, il faut appeler la méthode `scannePrix` de la manière suivante.
  ```
  prixArticle = ce.scannePrix();
  ```
  ◇ afficher la facture en appelant la méthode `afficheFacture` avec les variables ci-haut
  ```
  ce.afficheFacture( txTPS, txTVQ, prixArticle );
  ```
  ◇ exécuter et tester le programme

4.7.2 Création de Jar exécutable

On demande de :
— créer un jar exécutable `caisse.jar`. Pour cela, vous pouvez suivre la mache suivante sur eclipse :

- ◇ Cliquer sur le projet, puis `Fichier/exporter`
- ◇ `Java/Runnable Jar file`
- ◇ Sur `Launch Configuration`, sélectionner la classe et le projet
- ◇ Sur `Export destination`, choisir l'emplacement pour le fichier et lui donner le nom suivant
  `caisee.jar`
- ◇ cliquer sur `Finish`

— copier le fichier `caisse.jar` dans un autre dossier ou autre machine (avec Java installé) puis l'exécuter à partir un terminal avec la commande suivante :
`java -jar caisse.jar`

4.8 Laboratoire 8

L'objectif de ce laboratoire est de :
— se familiariser à la programmation orientée objet (OO)
— écrire un programme Java avec plusieurs classes
— mieux comprendre les variables de classe et d'instance
— créer un fichier Jar exécutable de l'application
— exécuter et distribuer l'exécutable de l'application développée
Pour ce laboratoire, on créera un projet Java nommé `inf1256lab08` et on mettra toutes les classes dans le package `lab08`

4.8.1 Classe Employe

On demande de définir une classe Java nommée `Employe` avec les spécifications suivantes
— accessibilité de la classe : `package-protected` (Accessible seulement au niveau du package). C'est à dire sans `modifier` (ex : `public`) explicite au niveau de la céclaration de la classe
— une variable de classe(avec `static`) nommée `indexAutomatique` de type entier avec une accessibilité `private` et initialisée à 1
— une variable d'instance (sans `static`) nommée `matricule` de type entier avec une accessibilité `private` et initialisée à 0. Cette variable garde le numéro matricule de l'employé
— une variable d'instance (Sans `static`) nommée `anciennete` de type entier avec une accessibilité `private` et initialisée à 0. Cette variable garde l'ancieneté de l'employé.
— deux autres variables d'instance d'accessibilité `private` pour garder respectivement le nom et le prénom de l'employé
— un constructeur `public Employe(String fn, String gn)` qui reçoit en argument le prénom et le nom puis les affecte aux variables d'instances correspondantes. Ce constructeur affecte aussi à la variable d'instance `matricule`, la valeur de `indexAutomatique` . Par la suite, on incrémente `indexAutomatique` de un. La variable ancienneté est aussi incrémentée de 1.
— une méthode `public static int getIndex()` qui retourne la valeur de `indexAutomatique`
— une méthode `public int getMatricule()` qui retourne la valeur de `matricule`
— une méthode `protected void displayInfo()` qui affiche le nom, prénom, le numéro matricule et l'ancienneté de l'employé

4.8.2 Classe Compagnie et le programme principal

On demande de créer une classe `Compagnie` avec la méthode `main` selon les spécifications suivantes
— accessibilité de la classe : `public`

— importer la classe `Employe` de la section 4.8.1
— afficher la valeur de `indexAutomatique` de la classe `Employe` obtenue en appelant le méthode de classe `getIndex()` avec le nom de la classe
— créer deux instances de la classe `Employe` en utilisant respectivement les variables `employe1` et `employe2`
— afficher les informations de `employe1` en appelant la méthode `displayInfo()`
— afficher les informations de `employe2` en appelant la méthode `displayInfo()`
— afficher à nouveau la valeur de `indexAutomatique` de la classe `Employe` obtenue en appelant le méthode de classe `getIndex()` avec le nom de la classe

4.8.3 Création d'un fichier Jar exécutable

Après avoir programmer toutes les classes :
— essayer d'abord d'exécuter l'application à partir de L'IDE
— observer les valeurs de l'index automatique, matricules et anciennetés
— Si tout fonctionne bien, alors, créer un jar exécutable puis essayer de l'exécuter en ligne de commande
— dans la mesure du possible, copier le fichier jar exécutable sur une autre machine où Java est installé puis essayer d'exécuter le programme en ligne de commande.

4.9 Laboratoire 10

L'objectif de ce laboratoire est de :
- se familiariser avec la programmation orientée objet (OO)
- se familiariser avec la lecture des fichiers textes et le traitement des chaines (`String`)

Pour ce laboratoire, on créera un projet Java nommé `inf1256lab10` et on mettra toute les classes dans le package `lab10`.

4.9.1 Données dans un fichier texte

On demande de :
- supposer que toutes les données manipulées dans ce laboratoire sont valides
- ajouter un sous dossier nommé `donnees` à la racine du projet Java
- ajouter dans le sous dossier `donnees`, le fichier `info.txt` fourni dans le listing 4.1

Listing 4.1 – Fichier texte du laboratoire de la section 4.9 : `info.txt`

```
Solange:35:17
Martin:35:12
Vincent:35
Karl:30:8
David:35:20
Harel:25:4
Monique:28
Geroges:38
Thomas:35:2
```

4.9.2 Programmation d'une classe

On demande d'ajouter une classe `Employes` dans le package `lab10` et de la programmer selon les spécifications suivantes :
- ajouter la méthode d'instance `void lireDonnees()` qui va lire et afficher les données du fichier `donnees/info.txt` dont les lignes ont le formatage fourni dans le listing 4.1 :
  - chaque ligne de ce fichier texte est composé des éléments séparés par deux points (`:`). Ces éléments sont placés dans l'ordre suivant :
    - le premier élément est le nom de l'employé
    - le 2e élément est le nombre d'heures normales de travail
    - le 3e élément est parfois absent. Quand il est présent, il représente le nombre d'heures supplémentaires de travail
  - pour chaque ligne on affichera les informations sous la forme :

    Nom : (nom de l'employé)

    HNormales : (nombre heures normals de travail)

    HSupp : pour les heures supplémentaires s'il y en a, sinon ne pas afficher ce titre

  Suggestion :
  - utiliser une boucle `for` pour lire le fichier texte

  - ◇ pour chaque ligne, décomposer les éléments en fonction du caractère `:`. Exemple :
    `elem=ligne.split(":")`
    `elem` devient ainsi un tableau. Pour savoir s'il y a des heures supplémentaires, il suffit de tester s'il y a plus de 2 éléments.
- ajouter la méthode `public static void main(String[] args)` à programmer de la manière suivante :
  - ◇ créer une instance de la classe `Employes`
  - ◇ appeler la méthode `lireDonnees()` de l'instance créée

4.10 Laboratoire 12

L'objectif de ce laboratoire est de :
- se familiariser davantage avec la programmation orientée objet (OO)
- savoir déterminer soi-même le type de données (méthode, variables, ...) approprié
- manipuler les collections : tableaux, ensembles (**set**), dictionnaire (`map`)

Pour ce laboratoire, on créera un projet Java nommé `inf1256lab12` et on mettra toutes les classes dans le package `lab12`

4.10.1 Manipulation Tableau

On demande d'ajouter une classe `Tableau` et la programmer selon les instructions suivantes

- ajouter dans la méthode `main`, un tableau `tabSimple` contenant les données suivantes :

12	15.98	12.9	45	7.9	-3.8

- ajouter dans la méthode `main`, un deuxième tableau `matriceCarre` contenant les données suivantes :

12.4	15.98	12.9	45	7.9	-7.68
12	1.98	12.9	5	7.9	3.1
0.6	15.98	1.47	54	7.9	-3.8
12	100	12.0	40	7.9	-3.8
2	15.98	12.9	4	7.9	-3.8
1.8	15.98	12.9	45	7.9	43.8

- déclarer une méthode `moyenneTableauSimple` qui reçoit en argument un tableau simple de nombre réels et retourne la moyenne des valeurs des éléments du tableau.
- déclarer une méthode `sommeDiagonaleMontante` qui reçoit en argument un tableau (de nombres réels) carré à deux dimensions et retourne la somme des éléments de la diagonale montante. Pour rappel, la diagonale montante est constituée des éléments du tableau allant du coin inférieur gauche au coin supérieur droit
- dans la méthode `main`, afficher la moyenne de `tabSimple` obtenue en appelant la méthode `moyenneTableauSimple`
- dans la méthode `main`, afficher la somme de la diagonale montante de `matriceCarre` obtenue en appelant la méthode `sommeDiagonaleMontante`

4.10.2 Collection : set et map

On demande d'ajouter une classe `Cours` et la programmer selon les instructions suivantes

— ajouter une variable de classe nommée `repertoireCours` de type dictionnaire (`Map`) ayant pour clé les sigles de cours (une chaine de caractères) et pour valeur les titres des cours (une chaine de caractères)
— dans la méthode `main`, ajouter les cours suivants dans `repertoireCours`

Sigle cours	**Titre cours**
INF1256	Informatique pour les sciences de la gestion
INF1120	Programmation 1
INF2005	Programmation Web
INF3005	Programmation Web Avancée

— dans la méthode `main`, définit un ensemble `sectionActuelle` de type ensemble (`Set`) des chaines de caractères
— toujours dans la méthode main, ajouter les chaines `INF1120` et **INF1256** dans `sectionActuelle`
— définir une méthode `afficheCoursSession` qui reçoit en argument un ensemble (`Set`) des chaines des caractères contenant les sigles des cours d'une session. La méthode affiche ces sigles et les titre des cours correspondant dans la variable `repertoireCours`. Cette méthode ne retourne rien et n'est pas `static`.
— à partir de la méthode `main`, appeler la méthode `afficheCoursSession` en passant en argument l'ensemble `sectionActuelle` pour afficher les cours de la session actuelle.

4.11 Laboratoire 13

L'objectif de ce laboratoire est de :
— se familiariser avec les classes abstraites
— se familiariser avec les sous classes
— se familiariser avec l'encapsulation de données
— manipuler des objets dans une collection (`map`)
Pour ce laboratoire, on créera un projet Java nommé `inf1256lab13` et on mettra toutes les classes dans le package `lab13`

4.11.1 Définir une classe abstraite

On demande de définir une classe abstraite publique nommée `FairePart` selon les spécifications suivantes.
— une variable d'instamce `nom` de type `String` et d'accessibilité `protected`
— une variable d'instance `numero` de type `int` et d'accessibilité `protected`
— les signatures des setters et getters abstraites des variables ci-haut et d'accessibilité `public`

4.11.2 Définir sous classe

On demande de définir une sous classe de la classe `FairePart` et nommée `Invitation`. Votre implémentation devra respecter les conditions suivantes :
— implémenter les méthodes déclarées dans la super classe
— une variable de classe (`static`) nommée `compteur`, de type `int` et d'accessibilité `private` et initialisée à 1
— un constructeur `Invitation(String nomInvite)` d'accessibilité `public` qui appelle le setter `setNom(nomInvite);`. on appelle aussi le setter `setNumero(this.compteur)` puis on incrémente la variable `compteur` d'une unité.

4.11.3 Programme principal

On demande de définir une classe `Evenement` qui a les fonctionnalité suivante
— une variable d'instance `invites` de type `Map` dont les clés sont de type `Integer` et les valeurs de type `Invitation`
— `void ajouteInvitation()` demande le nom de l'invité puis crée un objet de type `Invitation`. Par la suite on ajoute cet objet dans la variable `invites` avec la clé correspondant au numéro de l'invitation
— `void afficheInvites()` parcourt la variables `invites` et affiche les noms de tous les invités
— la méthode `main` contient le programme principal et affiche un menu :
  1 Ajouter une invitation
  2 Afficher a liste des invités
  3 Quitter

Ce menu s'affiche en boucle jusqu'à ce que l'utilisateur choisisse de quitter. À chaque choix, on appelle la méthode correspondante et on revient dans la boucle.

4.12 Laboratoire 14

L'objectif de ce laboratoire est de :
- se familiariser avec les classes abstraites
- se familiariser avec les sous classes
- se familiariser avec l'encapsulation de données

Pour ce laboratoire, on créera un projet Java nommé `inf1256lab14` et on mettra toutes les classes dans le package `lab14`

4.12.1 Définir une classe abstraite

On demande de définir une classe abstraite publique nommée `FairePart` selon les spécifications suivantes.
- une variable d'instamce `nom` de type `String` et d'accessibilité `protected`
- une variable d'instance `numero` de type `int` et d'accessibilité `protected`
- les signatures des setters et getters abstraites des variables ci-haut et d'accessibilité `public`

4.12.2 Définir une sous-classe

On demande de définir une *sous-classe* de la classe `FairePart` et nommée `Invitation`. L'implémentation devra respecter les conditions suivantes :
- contenir le code des méthodes abstraites déclarées dans la *super-classe*
- déclarer une variable de classe (`static`) nommée `compteur`, de type `int` et d'accessibilité `private` et initialisée à 1 (à la déclaration)
- un constructeur `Invitation(String nomInvite)` d'accessibilité `public` qui appelle le setter `setNom(nomInvite)`. On appelle aussi le setter `setNumero(this.compteur)` puis on incrémente la variable `compteur` d'une unité (Toujours dans le constructeur)

4.12.3 Programme principal

On demande de définir une classe `Rencontre` avec une méthode *main* à programmer de la manière suivante :
- créer 3 instances de `Invitation` pour Marie, Paul et Jean respectivement.
- en appelant les *getters* appropriés, afficher le nom de chacun des ces trois invités avec son numéro d'invitation
- afficher un message annonçant le changement des numéros d'invitations
- en appelant le *setter* adéquat, modifier les numéros d'invitations : 3 pour Marie, 1 pour Jean et 15 pour Paul.
- en appelant les *getters* appropriés, afficher le nom de chacun des ces trois invités avec son numéro d'invitation
- ajouter d'autre manipulation à volonté

Deuxième partie

Travaux pratiques

Chapitre 5

TPs de la session d'hiver 2017

Ce chapitre contient les énoncés des TP de la session d'hiver 2017.

5.1 TP 1 Hiver 2017 : Gestion des Feuilles de Temps – Prototype

QC RH Inc est une société dont les employés travaillent sur plusieurs projets. La compagnie aimerait faire développer une application permettant d'établir plus rapidement les feuilles de temps des employés en comptabilisant le temps passé sur chaque projet. Cette application doit permettre de saisir les informations des employés et de calculer le nombre d'heures total pour la paie de la période correspondante. Les heures supplémentaires éventuelles seront gardées en banque. L'application devrait afficher pour chaque employé, les informations saisies et les résultats des calculs. Pour cette première étape, on ne traitera que les informations d'un seul employé à la fois (lors d'une exécution). On demande de concevoir et de développer un prototype de cette application en vous basant sur la description donnée ci-après. On notera que dans ce secteur d'activités, la durée légale du temps de travail (`TEMPS_TRAVAIL`) est de 40 heures par semaine. La déclaration de feuille de temps se fait à chaque semaine parce que la compagnie détermine les heures à payer sur une hase hebdomadaire. L'employé qui a travaillé moins que la durée légale du temps de travail et qui a des heures en banque peut les déclarer pour se faire payer.

5.1.1 Informations sur les employés

Pour le développement de ce prototype, l'application demandera les informations suivantes pour chaque employé.

- Le numéro matricule (`matricule`) de l'employé. Ce numéro est un nombre entier
- Le nom (`nom`)
- Le prénom (`prenom`)

— Le nombre d'heures en banque (`nhBanque`) prestées précédemment mais qui n'ont pas encore été payées
— Le nombre d'heures passées à travailler sur le projet 1 (`nhProjet1`)
— Le nombre d'heures passées à travailler sur le projet 2 (`nhProjet2`)
— Le nombre d'heures passées à travailler sur le projet 3 (`nhProjet3`)
— Le nombre d'heures passées à travailler sur le projet 4 (`nhProjet4`)

Toutes les variables devront être initialisée à la déclaration. Quant aux projets (1, 2, 3, 4), on devra d'abord demander à l'utilisateur si l'employé a travaillé sur le projet avant de demandé le nombre d'heures. On présume qu'un employé a presté 0 H sur un projet s'il n'a pas travaillé sur ce projet. Le nombre d'heures sera entré sous forme de nombre réel. Ex :

$$1H30 \;=\; 1 + (\frac{30}{60}) = 1.5H$$

On saisira donc au clavier 1.5 pour $1H30$ mais on affichera $01:30$ pour permettre à l'utilisateur de comprendre facilement.

5.1.2 Calcul des heures à payer et à mettre en banque

Les calculs ne se feront qu'après avoir saisie toutes les informations nécessaires de la feuille de temps. Pour chaque feuille de temps, l'application calcule les informations suivantes :

`nhTotal` : la somme des heures passées dans différents projets et les heures en banques

`nhAPayer` : les heures à payer sont déterminées au moyen de la formule suivante.

$$nhAPayer \;=\; \min\,(\; nhTotal \;,\; TEMPS\_TRAVAIL \;)$$

Cette formule se justifie par le fait qu'on paye toutes les heures si ça ne dépasse pas le durée légale du temps de travail. Sinon, on paye la durée légale de temps de travail et les heures supplémentaires iront en banque. Dans ce prototype, on n'aborde pas encore la reprise du temps.

`nhNouvelleBanque` : Les heures à reporter en banque sont celles qui n'auront pas été payées. On les détermines à l'aide de la formule suivante :

$$nhNouvelleBanque \;=\; nhTotal \;-\; nhAPayer$$

Pour effectuer ces calculs, on gardera les nombres d'heures en chiffres décimaux, mais l'affichage se fera selon les instructions de la section suivante.

5.1.3 Affichage de la feuille de temps

L'affichage de la feuille de temps n'interviendra qu'après tous les calculs. Pour chaque feuille de temps, on affichera les informations recueillies selon la section 5.1.1 et celles calculées conformément à la section 5.1.2. On affichera une information par ligne et tous les affichages seront alignés à gauche. Quand aux hures, elle seront affichées au format suivant

`hh : mm`

Dans cette écriture, les heures (hh) seront représentées avec au moins 2 chiffres et les minutes (mm) en 2 chiffres aussi. Les chiffres manquants éventuels seront remplacés par des 0.

5.1.4 Principales étapes du programme

Étape 0 : Au début, votre programme devra initialiser les variables et les constantes nécessaires avant d'afficher un message de bienvenue et une brève description de ce qu'il fait. Par la suite, il passe à l'étape suivante.

Étape 1 : Demander à l'utilisateur s'il veut saisir une feuille de temps. Si la réponse est oui (`'O'` ou `'o'`) alors on demande les informations détaillées à la section 5.1.1, on les enregistre dans les variable et on passe à l'**Étape 2**. Si la réponse est non (`'N'` ou `'n'`), on passe à l'**Étape 4**. **Notez que toutes les entrées doivent être validées**. Cela veut dire qu'on doit s'assurer que l'utilisateur entre une donnée valide avant de lui demander une autre information. S'il entre une donnée invalide (Ex : chaine au lieu de nombre), on doit afficher un message d'erreur et demander à nouveau la même information.

Étape 2 : Calculer les informations de la section 5.1.2 pour la feuille de temps. On concerve aussi les informations obtenues à l'**Étape 1**.

Étape 3 : Afficher les informations selon les directives de la section 5.1.3

Étape 4 : Afficher un message de fin du programme.

5.1.5 Consignes et informations pratiques

Cette section contient les consignes sur l'organisation et la remise du TP.

Groupe : Travail à faire par groupe de deux personnes au maximum avec respect strict de l'intégrité académique. À part les librairies autorisées éventuellement, on ne peut utiliser que le code de du groupe TP auquel on appartient. Il est interdit de copier du code sur Internet.

Découpage : Ce travail est découpé et pondéré de la manière suivante

- Partie 1 (60%) : Un **document pdf** sur les aspects de conception

  Organigrammes : Donnez les organigrammes suivants.

  O1 : la séquence des principales étapes reprises à la section 5.1.4

  O2 : la séquence des opérations de calcul de la section 5.1.2 en partant des initialisations suivantes

```
nhBanque ← "Nombre heures en banque"
nhProjet1 ← "nombre heures projet1"
nhProjet2 ← "nombre heures projet2"
nhProjet3 ← "nombre heures projet3"
nhProjet4 ← "nombre heures projet4"
```

O3 : les opérations pour entrer et valider le `nom`

O4 : les opérations pour entrer et valider le `matricule`

O5 : les opérations pour entrer et valider les heures prestées dans un projet (pour un seul projet)

N'entrez pas dans les détails des petites opérations. Pour information, le site web suivant offre gratuitement des outils nécessaires pour dessiner des organigrammes.
https://www.draw.io/

Algorithmes : Donnez, en pseudo-code, un algorithme correspondant à l'organigramme **O2**

◇ Partie 2 (40%) : Un projet Java portant le nom `inf1256TP1`. Pour ce TP 1, à part les constantes et les variables globales, on mettra toute la programmation dans la méthode `main`. La **note de 0 sera attribuée automatiquement à la partie 2 pour tout programme qui ne compile pas ou ne s'exécute pas**

Date de remise : **Au plus tard le 2017-02-24 à 23H55**

Format de remise : Déposer **un fichier zip** contenant tous vos fichiers et dossiers sur http://www.moodle.uqam.ca/. La remise par un membre du groupe est suffisant pour tout le groupe. Il faut se mettre d'accord sur la personne qui va faire la remise.

Gestion de code source : Pour mieux collaborer en équipe et minimiser le risque de perte de code en cas de problème, il vous est encouragé vivement d'utiliser un gestionnaire de code source comme `git`. L'accès à votre projet git doit demeurer privé jusqu'à la fin de la session. Les fournisseurs suivants offrent la possibilité d'héberger un projet privé gratuitement.
— https://gitlab.com
— https://bitbucket.org
L'utilisation d'un gestionnaire de code source (`git`) ne fait pas partie de l'évaluation. C'est juste pour vous faciliter le travail.

Aide : Au besoin, n'hésitez pas à contacter l'enseignant

5.2 TP 2 Hiver 2017 : Gestion des Feuilles de Temps – Méthodes et exécutable

QC RH Inc est une société dont les employés travaillent sur plusieurs projets. La compagnie aimerait faire développer une application permettant d'établir plus rapidement les feuilles de temps des employés en comptabilisant le temps passé sur chaque projet. Cette application doit permettre de saisir les informations des employés et de calculer le nombre d'heures total pour la paie de la période correspondante. Les heures supplémentaires éventuelles seront gardées en banque. L'application devrait afficher pour chaque employé, les informations saisies et les résultats des calculs. Pour cette version, on devra pouvoir traiter les informations de plusieurs employés lors d'une exécution. La société est globalement satisfaite du prototype développé au TP1 (voir 5.1), mais elle aimerait qu'on apporte notamment les améliorations suivantes :

- restructurer le code en méthodes pour éviter des redondances et faciliter l'évolution et la maintenance de l'application
- fournir un fichier jar exécutable pour permettre le déploiement (installation) éventuel de l'application
- ajouter quelques fonctionnalités et valider toutes les réponses ou saisies de l'utilisateur.

Dans l'énoncé du présent Travail, l'expression `saisir et valider` signifie que si l'utilisateur ne saisit pas une valeur valide, alors on affiche un message d'erreur puis on lui demande à nouveau de saisir la bonne valeur. On boucle ainsi jusque quand il aura saisi la bonne valeur.

Pour faciliter la tâche, on donne les descriptions suivantes pour les méthodes à implémenter. On se souviendra que dans ce secteur d'activités, la durée légale du temps de travail (`TEMPS_TRAVAIL`, une constante globale) est de 40 heures par semaine. La déclaration de feuille de temps se fait à chaque semaine parce que la compagnie détermine les heures à payer sur une hase hebdomadaire. L'employé doit déclarer les heures éventuelles en banque et le temps repris éventuellement pour permettre de déterminer les heures à payer et celles à mettre dans la nouvelle banque.

5.2.1 Méthodes pour la saisie des informations

Pour cette nouvelle version, l'application demandera les informations suivantes pour chaque employé.

- Le numéro matricule (`matricule`) de l'employé. Ce numéro est un nombre entier
- Le nom (`nom`)
- Le prénom (`prenom`)
- Le nombre d'heures en banque (`nhBanque`) prestées précédemment mais qui n'ont pas encore été payées
- Le temps repris (`nhReprises`). C'est le nombre d'heures à décompter (soustraire) des heures en banque. Il faut donc respecter la condition

suivante au moment de saisir le temps repris :
0 <= $nhReprises$ <= $nhBanque$
Par défaut, le temps repris est égal à 0. Il faut demander à l'utilisateur s'il a repris du temps avant de saisir et valider ce temps.

— Le nombre d'heures passées à travailler sur le projet 1 (`nhProjet1`)
— Le nombre d'heures passées à travailler sur le projet 2 (`nhProjet2`)
— Le nombre d'heures passées à travailler sur le projet 3 (`nhProjet3`)
— Le nombre d'heures passées à travailler sur le projet 4 (`nhProjet4`)

Toutes les variables devront être initialisées à la déclaration. Quant aux projets (1, 2, 3, 4), on devra d'abord demander à l'utilisateur si l'employé a travaillé sur le projet avant de demander le nombre d'heures. On présume qu'un employé a presté 0 H sur un projet s'il n'a pas travaillé sur ce projet. Le nombre d'heures sera entré sous forme de nombre réel. Ex :

$$1H30 \quad = \quad 1 + (\frac{30}{60}) = 1.5H$$

On saisira donc au clavier 1.5 pour $1H30$ mais on affichera 01 : 30 pour permettre à l'utilisateur de comprendre facilement.

Pour faciliter la saisie de ces informations, on demande de programmer les méthodes suivantes.

— `boolean ouiOuNon(String message)` reçoit en argument une chaine de caractères contenant le message à afficher à l'utilisateur pour saisir et valider une réponse de type `oui` ou `non`. Si la réponse correspond à oui, la méthode retourne `true`. Si la réponse correspond à non, on retourne `false`. Dans tout autre cas, on continue de poser la question jusqu'à ce qu'une réponse valide soit saisie
— `int saisirEtValiderNombreEntier(String message)` reçoit en argument une chaine contenant le message à afficher à l'utilisateur pour le nombre entier à saisir et valider. Elle retourne le nombre entier valide saisi
— `double saisirEtValiderNombreReel(String message)` reçoit en argument une chaine contenant le message à afficher à l'utilisateur pour le nombre réel à saisir et valider. Elle retourne le nombre réel valide saisi. Les nombres entiers sont aussi acceptés comme nombres réels(ex : `20` et `20.0` sont acceptés comme nombre réels) .
— `double saisirEtValiderNombreReel(String message, double mini, double maxi)` reçoit un message, un nombre minimum et un nombre maximum. Cette méthode sert à saisir et valider un nombre réel compris entre `mini` et `maxi`. Pour se faire, elle appelle la méthode :
`saisirEtValiderNombreReel(message)`
tant que le nombre saisi n'est pas dans l'intervalle voulu. Elle retourne le nombre réel validé.

Note : Cette méthode porte le même nom que la méthode précédente mais le nombre et les types d'arguments sont différents. Ce qui distingue leurs signatures (Java `overloading`)

— `String saisirEtValiderChaineCaracteres(String message, String pattern)` reçoit en argument une chaine (`message`) contenant le message à afficher à l'utilisateur pour la chaine de caractères à saisir et valider conformément au `pattern` passé en deuxième argument. Elle retourne la chaine de caractères valide saisi

Pour chacune des informations de l'employé, il faut appeler la méthode appropriée pour la saisir et la valider. Ex :

```
message = "Entrez le numéro matricule -- nombre entier svp";
matricule = saisirEtValiderNombreEntier(message);
```

5.2.2 Calcul des heures à payer et à mettre en banque

Pour effectuer les calculs, on se sert des méthodes suivantes :

— `double calculTotalHeures(double hbanque, double hreprises, double... hprojets)` calcule et retourne le nombre d'heures total. Elle reçoit en argument le nombre d'heures en banque, le nombre d'heures reprises et un nombre variable d'arguments supplémentaires représentant chacun le nombre d'heures travaillées dans un projet. La méthode retourne le résultat de la somme des heures passées dans différents projets et les heures en banques moins les heures reprises.

— `double calculHeuresAPayer(double nhTotal)` calcule et retourne le nombre d'heures à payer. Le calcul de la valeur à retourner se fait selon la formule suivante.

$$hAPayer \;=\; \min\;(hTotal\;,\;TEMPS\_TRAVAIL\;)$$

— `double calculNouvelleBanque(double hTotal, double hAPayer)` calcule et retourne le nombre d'heures à mettre dans la nouvelle banque. La valeur à retourner se calcule avec la formule suivante :

$$hNouvelleBanque \;=\; nhTotal \;-\; nhAPayer$$

Les calculs ne se feront qu'après avoir saisi toutes les informations nécessaires de la feuille de temps. Pour chaque feuille de temps, l'application calcule les informations suivantes :

`nhTotal` : Le nombre d'heure total

`nhAPayer` : les heures à payer

`nhNouvelleBanque` : Les heures qui n'auront pas été payées et qu'il faut rapporté en banque.

Pour effectuer ces calculs, on gardera les nombres d'heures en chiffres décimaux, mais l'affichage se fera selon les instructions de la section suivante.

5.2.3 Affichage de la feuille de temps

L'affichage de la feuille de temps n'interviendra qu'après tous les calculs (de la feuille de temps). Pour chaque feuille de temps, on affichera les informations recueillies selon la section 5.1.1 et celles calculées conformément à la section 5.1.2. On affichera une information par ligne et tous les affichages seront alignés à gauche. Quant aux heures, elles seront affichées au format suivant
`hh : mm`
Dans cette écriture, les heures (hh) seront représentées avec au moins 2 chiffres et les minutes (mm) en 2 chiffres aussi. Les chiffres manquants éventuels seront remplacés par des 0. Les heures seront affichées en appelant la méthode suivantes :

— `void afficheHeures(String message, double nbheures)` reçoit en argument un message et le nombre d'heures qu'il faut afficher dans le format indiqué ci-haut. Cette méthode ne retourne rien. Ex :

```
//appel
message = "Nombre heures total ";
nbheures = 1.5;
afficheHeures( message, nbheures );

//affichage
Nombre heures total   01 : 30
```

5.2.4 Principales étapes du programme

Étape 0 : Au début, votre programme devra initialiser les variables et/ou les constantes nécessaires avant d'appeler la méthode `bienvenue()` dont la description est donnée ci-après . Par la suite, il passe à l'étape suivante (`Étape 1`).

— `void bienvenue()` ne reçoit aucun argument et ne retourne rien. cette méthode affiche un message de bienvenue et une brève description de ce fait le programme.

Étape 1 : Demander à l'utilisateur s'il veut saisir une feuille de temps. Si la réponse est oui (`'O'` ou `'o'`) alors on demande les informations détaillées à la section 5.1.1, on les enregistre dans les variables et on passe à l'**Étape 2**. Si la réponse est non (`'N'` ou `'n'`), on passe à l'**Étape 4**. **Ne pas oublier que toutes les entrées doivent être validées**. Cela veut dire qu'on doit appeler les méthodes appropriées pour saisir et valider une donnée valide avant de demander une autre information.

Étape 2 : Calculer les informations de la section 5.2.2 pour la feuille de temps. On conserve aussi les informations obtenues à l'**Étape 1**.

Étape 3 : Afficher les informations selon les directives de la section 5.2.3 puis retourner à l'**Étape 1** (boucle).

Étape 4 : Afficher un message de fin du programme.

5.2.5 Consignes et informations pratiques

Cette section contient les consignes sur l'organisation et la remise du TP.

Groupe : Travail à faire dans le même groupe qu'au TP1 avec respect strict de l'intégrité académique. À part les librairies autorisées éventuellement, on ne peut utiliser que le code du groupe TP auquel on appartient. Il est autorisé de s'inspirer et d'utiliser certains extraits de l'exemple de solution du TP1 fourni par l'enseignant. Il est interdit de copier du code sur Internet.

Découpage : Ce travail est découpé et pondéré de la manière suivante.

- Partie 1 (95%) : Un projet Java portant le nom `inf1256TP`. Pour ce TP 2, le programme principal sera dans la méthode `main` et on appellera les méthodes expliquées ci-avant. Il est permis d'ajouter des méthodes supplémentaires si vous en avez besoin pour améliorer votre programme. Tout doit se faire dans une seule classe. La **note de 0 sera attribuée automatiquement à tout le TP 2 pour tout programme qui ne compile pas ou ne s'exécute pas**
- Partie 2 (5%) : Un jar exécutable (`Runnable JAR file`) et un petit fichier `README.text` expliquant en quelques lignes comment déployer (installer) et exécuter le programme.

Date de remise : **Au plus tard le 2017-03-24 à 23H55**

Format de remise : Déposer **un fichier zip** contenant tous vos fichiers et dossiers sur http://www.moodle.uqam.ca/. La remise par un membre du groupe est suffisant pour tout le groupe. Il faut se mettre d'accord sur la personne qui va faire la remise.

Gestion de code source : Pour mieux collaborer en équipe et minimiser le risque de perte de code en cas de problème, il vous est encouragé vivement d'utiliser un gestionnaire de code source comme `git`. L'accès à votre projet git doit demeurer privé jusqu'à la fin de la session. Les fournisseurs suivants offrent la possibilité d'héberger un projet privé gratuitement.

— https://gitlab.com
— https://bitbucket.org

L'utilisation d'un gestionnaire de code source (`git`) ne fait pas partie de l'évaluation. C'est juste pour vous faciliter le travail.

Suggestion : Bien comprendre l'énoncé et réutiliser au mieux possible le code source du TP 1

Aide : Au besoin, n'hésitez pas à contacter l'enseignant

5.3 TP 3 Hiver 2017 : Gestion des Feuilles de Temps – Héritage, Collections et Fichiers textes

QC RH Inc est une société dont les employés travaillent sur plusieurs projets. La compagnie aimerait faire développer une application permettant d'établir plus rapidement les feuilles de temps des employés en comptabilisant le temps passé sur chaque projet. Cette application doit permettre de saisir les informations des employés et de calculer le nombre d'heures total pour la paie de la période correspondante. Les heures supplémentaires éventuelles seront gardées en banque. A la demande et pour chaque employé spécifié, l'application devrait afficher, les informations saisies et les résultats des calculs. Pour cette version, on devra pouvoir traiter les informations de plusieurs employés lors d'une exécution. La société est globalement satisfaite de la dernière version de l'application développé au TP2 (voir 5.2). Cette fois-ci, elle aimerait obtenir un module qui va permettre aux gestionnaires de valider et/ou rejeter les feuilles de temps déclarées par les employés. Pour cela, toutes les classes du présent travail seront placées dans le package `h2017`. Le nouveau programme devra notamment offrir les fonctionnalités suivantes :

- lire les informations des feuilles de temps à partir d'un fichier texte
- afficher les feuilles de temps
- accepter ou rejeter avec un commentaire une feuille de temps
- enregistrer, dans un fichier texte, les feuilles de temps avec les statuts d'acceptation et/ou de rejet éventuel

En plus du code source du programme, un fichier jar exécutable devra être fourni pour permettre le déploiement (installation) éventuel de l'application

Pour simplifier le travail, on supposera que toutes informations contenues dans le fichier des feuilles de temps sont valides.

5.3.1 Format du fichier des feuilles de temps

Un fichier des feuilles de temps est un fichier texte contenant les informations d'une feuille de temps par ligne. Sur une ligne, les information sont disposées de la manière suivante :

- Si le gestionnaire n'a ni accepté ni rejeté la feuille de temps :
  `matricule|nom|prenom|nhBanque|nhTrav|nhRep`
- Si le gestionnaire a accepté la feuille de temps
  `matricule|nom|prenom|nhBanque|nhTrav|nhRep|ACC`
- Si le gestionnaire a rejeté la feuille de temps
  `matricule|nom|prenom|nhBanque|nhTrav|nhRep|REJ|Commentaire`

Dans ce formalisme, les éléments sont interprétés comme ceci :

| Séparateur des champs

matricule matricule de l'employé, un nombre entier

nom nom de famille de l'employé

prenom prénom de l'employé

nhBanque le nombre d'heures présentement en banque de l'employé

nhTrav Le total de nombre d'heures réellement travaillées par l'employé au cours de la période

nhRep Le nombre d'heures que l'employé a repris de sa banque (voir `nhBanque`)

ACC Statut indiquant que la feuille de temps a été acceptée

REJ Statut indiquant que la feuille de temps a été rejetée

Commentaire Commentaire du gestionnaire donnant la raison pour laquelle la feuille de temps a été rejetée

Le listing 5.1, illustre un exemple d'un fichier texte contenant une feuille de temps par ligne.

Listing 5.1 – Exemple de fichier de feuilles de temps : `feuillesTemps.txt`

```
15|Gagnon|Charles|0.0|38.5|0.0
8|Garvais|Carl|10.0|25.5|3.0|ACC
35|Kadima|Charles|2.0|24.5|5.0|REJ|Erreur temps repris
12|Tremblay|Marcel|5.0|38.5|2.0|ACC
2|Veerbos|Marc|0.0|27.2|0.0
```

5.3.2 Classe Employe

Listing 5.2 – Classe abstraite : `Travailleur.java`

```
/**
 * @author Johnny Tsheke @ UQAM --INF1256
 * 2017-04-08
 */
package h2017;
import h2017.*;

public abstract class Travailleur {
            protected final StatutFeuille STATUT_INCONNU = StatutFeuille.INC;
            protected final StatutFeuille STATUT_REJETE = StatutFeuille.REJ;
            protected final StatutFeuille STATUT_ACCEPTE = StatutFeuille.ACC;
            private int matricule = 0;
        private String nom = "";
        private String prenom = "";
        private double nhBanque = 0; //heures en banque
        private double nhTravaillees = 0; //heures travaillées
        private double nhReprises = 0; //heures reprises
        private StatutFeuille statut = this.STATUT_INCONNU;
        private String commentaire = "";

        //setters et getters --implementer ces methodes dans la sous classe
        //matricule
          void setMatricule(int ma){
                  this.matricule = ma;
          }
         public int getMatricule(){
                  return(this.matricule);
         }
         //nom
         void setNom(String nomTrav){
                  this.nom = nomTrav;
         }
         public  String getNom(){
                  return (this.nom);
         }
       //prenom
          void setPrenom(String prenomTrav){
                  this.prenom = prenomTrav;
          }

         public String getPrenom(){
                  return (this.prenom);
         }
       //nhBanque
          void setNhBanque(double nhBank){
                  this.nhBanque = nhBank;
          }
         public double getNhBanque(){
```

```
                return (this.nhBanque);
            }
            //nhTravaillees
            void setNhTravaillees(double nhTrav){
                this.nhTravaillees = nhTrav;
            }

            public double getNhTravaillees(){
                return (this.nhTravaillees);
            }
        //nhReprises
            void setNhReprises(double nhRep){
                this.nhReprises = nhRep;
            }
            public  double getNhReprises(){
                return (this.nhReprises);
            }
        //nhReprises
            void setStatut(StatutFeuille stat){
                this.statut = stat;
            }
            public StatutFeuille getStatut(){
                return (this.statut);
            }
        //commentaire
            void setCommentaire(String comment){
                this.commentaire = comment;
            }
            public  String getCommentaire(){
                return(this.commentaire);
            }

        //décomposition des élements a partir d'une ligne de texte
            //voir ennonce TP3 hiver 2017 pour details
            abstract void decomposerFeuilleTemps(String ligneFeuilleTemps);
            //rassembler elements pour former une ligne de texte.
            //voir ennonce TP3 H2017
            public abstract String assemblerFeuilleTemps();
            //affichage d'une donnée de heures
            abstract void afficheHeures(String message, double nbheures);
            //la methode suivante affiche les information de la feuille de temps
            //une information par ligne
            public abstract void afficheFeuilleTemps();

}
```

Listing 5.3 – Type énuméré : `StatutFeuille.java`

```
/**
 * @author Johnny Tsheke @ UQAM
 * 2017-04-08
 */
package h2017;

/**
 * @author johnny
 *
 */
public enum StatutFeuille {
        ACC,REJ,INC
}
//ACC acepté
//REJ rejeté
//INC inconnu, cad non encore traité, pas enregitré dans feuille temps
```

Vous pouvez télécharger le projet Java nécessaire pour ce travail à partir du lien https://gitlab.com/inf1256/inf1256tp/repository/archive.zip?ref=TP3H2017. Vous pouvez aussi obtenir ce code plus facile au moyen de la commande git suivante :

```
git clone -b TP3H2017 https://gitlab.com/inf1256/inf1256tp.git
```

Si vous créez votre propre projet Java, vous pouvez mettre les fichiers textes des feuilles de temps dans le sous dossier `donnees` et les classes ainsi que les types de données `enum` dans le package `h2017`.
On demande de créer une classe `Employe` dans le package `h2017` selon les spécifications suivantes :

- la classe `Employe` est une sous classe de la classe abstraite **Travailleur**
- implémeter toutes les méthodes déclarées dans la classe `Travailleur` (Ne pas reprendre le mot `abstract` dans la déclaration des méthodes de la classe `Employe`)
- les méthodes – setters – dont les noms commencent par `set` affecte à la variable d'instance correspondante la valeur de la variable passée en argument.
- les méthodes – getters – dont les noms commencent par `get` retournent les valeurs des variables d'instances correspondantes.
- `void decomposerFeuilleTemps(String ligneFeuilleTemps)` reçoit en argument une chaine correpondant à une ligne du fichier de feuille de temps tel qu'expliqué à la section 5.3.1. Elle décompose cette ligne et pour chaque champ, appelle le `setter` correspondant pour faire l'affectation de la variable d'instance concernée.
- `public String assemblerFeuilleTemps()` prend les valeurs des différent champs pour former une chaine correspondant à une ligne du fichier de feuilles de temps (voir 5.3.1). Le statut et éventuellement le commentaire ne feront partie de la chaine que si la feuille de temps a été acceptée et/ou rejetée
- `void afficheHeures(String message, double nbheures)` a la même description qu'au TP 2 (Voir 5.2.3)
- `public void afficheFeuilleTemps()` affiche toutes les composantes de la feuille de temps. On affiche une composante par ligne. On appelle les `getters` pour obtenir les valeurs des variables d'instances. Les heures seront affichées en appelant la méthode `afficheHeures`.
- `public Employe(String ligne)` est un constructeur qui reçoit en argument une ligne contenant les informations d'une feuile de temps puis appelle la méthode `decomposerFeuilleTemps(ligne)` pour faire les affectations nécessaires.

5.3.3 Classe TP3 et le programme principal

Cette classe `TP3` contiendra le programme principal et sera dans le package `h2017`. On demande de l'implémenter selon les spécifications suivantes :

- importer les classes et type de données nécessaires (y compris ceux du package `h2017`)
- une variable d'instance `clavier` de type `Scanner` pour permettre la saisie de données au clavier. Cette variable sera initialiser au début de la méthode `main` et fermée à la fin de la méthode `main`.
- Une variable d'instance `bd` de type `Map` ayant pour type de clés `Integer` et pour type de valeur `Employe`. Cette variable contiendra les objets de la classe `Employe` et les matricules des employés seront utilisées comme clés.
- `void lireFeuillesTemps(String nomFichier)` est une méthode qui reçoit en argument le nom du fichier contenant les feuilles de temps. Elle parcours le fichier ligne par ligne et, pour chaque ligne, elle crée un objet

de type `Employe`. par la suite, on appelle la méthode `getMAtricule()` de l'objet créé pour obtenir le numéro matricule qui servira de clé pour insérer l'objet dans la variable `bd` expliquée ci-haut.

- `void ecrireFeuillesTemps(String nomFichier)` est une méthode qui reçoit en argument un nom de fichier pour y écrire les feuilles de temps (éventuellement validées et/ou rejetées). Pour cela, on parcourt les objets de la variable `bd` ci-haut et pour chaque objet, on appelle la méthode `assemblerFeuilleTemps()` pour obtenir la ligne de texte à écrire dans le fichier.
- `int saisirEtValiderNombreEntier(String message)` reçoit en argument une chaine contenant le message à afficher à l'utilisateur pour le nombre entier à saisir et valider. Elle retourne le nombre entier valide saisi (même méthode qu'au TP2 , voir 5.2.1)
- `void afficherUneFeuille()`, demande le numéro matricule (appeler `saisirEtValiderNombreEntier`) et cherche l'objet dans la variable `bd` pour afficher la feuille de temps. Pour se faire, on appelle la méthode `afficheFeuilleTemps()` de l'objet. Si le nombre saisi ne correspond à aucun objet dans `bd`, on affiche un message pour le signifier à l'utilisateur.
- `void afficherToutesLesFeuilles()` affiche toutes es feuilles de temps. Pour cala, il faut parcourir tous les objets `Employe` de la variable `bd` et pour chacun, appeler la méthode `afficheFeuilleTemps()` pour afficher la feuille de temps correspondante.
- `boolean ouiOuNon(String message)` reçoit en argument une chaine de caractères contenant le message à afficher à l'utilisateur pour saisir et valider une réponse de type `oui` ou `non`. Si la réponse correspond à oui, la méthode retourne `true`. Si la réponse correspond à non, on retourne `false`. Dans tout autre cas, on continue de poser la question jusqu'à ce qu'une réponse valide soit saisie (même méthode qu'au TP2 , voir 5.2.1)
- `String saisirEtValiderChaineCaracteres(String message, String pattern)` reçoit en argument une chaine (`message`) contenant le message à afficher à l'utilisateur pour la chaine de caractères à saisir et valider conformément au `pattern` passé en deuxième argument. Elle retourne la chaine de caractères valide saisi (même méthode qu'au TP2, 5.2.1)
- `void bienvenue()` ne reçoit aucun argument et ne retourne rien. cette méthode affiche un message de bienvenue et une brève description de ce fait le programme (même méthode qu'au TP2 mais il faut adapter le message, 5.2.1).
- `void validerFeuilleTemps()` cette methode demande le numéro matricule de l'employé puis cherche l'objet correspondant dans la variable `bd`. Si l'objet existe, on affiche la feuille de temps correspondante en appelant la méthode `afficheFeuilleTemps()` de l'objet. Par la suite, on appelle la méthode `ouiOuNon` pour demander si l'utilisateur valide la feuille de temps.
  - Si la réponse est oui, on appelle la méthode `setStatut` de l'objet en passant `StatutFeuille.ACC` en paramètre pour indiquer que la

feuille de temps a été acceptée
  - ◇ Si la réponse est non, on appelle `setStatut` de l'objet en passant `StatutFeuille.REJ` en paramètre pour indiquer que la feuille de temps a été rejetée. La raison du rejet de la feuille de temps sera obtenue en appelant la méthode `saisirEtValiderChaineCaracteres`. Avec le commentaire saisi, on appellera la méthode `setCommentaire` de l'objet pour garder la raison du rejet de la feuille de temps

— la méthode `main` contiendra le programme principal. On initialise toutes les variables nécessaire y compris les variables d'instances expliquées ci-haut. Pour faire simple, on prendra `donnees/feuillesTemps.txt` comme fichier d'entrée contenant les feuilles de temps et `donnees/feuillesValidees.txt` comme fichier de sortie (après validation éventuelle). On charge les feuilles de temps en appelant la méthode `lireFeuillesTemps`. Par la suite, on appelle la méthode `bienvenue()` avant d'afficher le menu suivant en boucle :

1 : Afficher toutes les feuilles de temps
2 : Afficher Une feuille de temps
3 : Valider une feuille de temps
4 : Sauvegarder les feuille de temps et quitter

Si l'utilisateur choisi une option, on appelle la méthode correspondante puis on revient dans la boucle. Sauf s'il choisi de sauvegarder la feuille de temps auquel cas, on appelle la méthode `ecrireFeuillesTemps` et après, on quitte la boucle pour terminer le programme. C'est la seule façon de terminer le programme normalement.

5.3.4 Consignes et informations pratiques

Cette section contient les consignes sur l'organisation et la remise du TP.

Groupe : Travail à faire dans le même groupe qu'au TP2 avec respect strict de l'intégrité académique. À part les librairies autorisées éventuellement, on ne peut utiliser que le code du groupe TP auquel on appartient. Il est autorisé de s'inspirer et d'utiliser certains extraits des exemples fournis par l'enseignant. Il est interdit de copier du code sur Internet.

Découpage : Ce travail est découpé et pondéré de la manière suivante.
  - ◇ Partie 1 (95%) : Un projet Java portant le nom `inf1256TP`. Il est permis d'ajouter des méthodes supplémentaires si vous en avez besoin pour améliorer votre programme. **Aucune méthode ne doit-être implémentée dans une autre**. La **note de 0 sera attribuée automatiquement à tout le TP 3 pour tout programme qui ne compile pas ou ne s'exécute pas**
  - ◇ Partie 2 (5%) : Une bonne structure de projet avec les différentes classes, un jar exécutable (`Runnable JAR file`) et un petit fichier `README.text` expliquant en quelques lignes comment déployer (installer) et exécuter le programme.

Date de remise : **Au plus tard le 2017-04-21 à 23H55**

Format de remise : Déposer **un fichier zip** contenant tous vos fichiers et dossiers sur http://www.moodle.uqam.ca/. La remise par un membre du groupe est suffisant pour tout le groupe. Il faut se mettre d'accord sur la personne qui va faire la remise.

Gestion de code source : Pour mieux collaborer en équipe et minimiser le risque de perte de code en cas de problème, il vous est encouragé vivement d'utiliser un gestionnaire de code source comme `git`. L'accès à votre projet git doit demeurer privé jusqu'à la fin de la session. Les fournisseurs suivants offrent la possibilité d'héberger un projet privé gratuitement.

— https://gitlab.com

— https://bitbucket.org

L'utilisation d'un gestionnaire de code source (`git`) ne fait pas partie de l'évaluation. C'est juste pour vous faciliter le travail.

Suggestion : Bien comprendre l'énoncé et réutiliser au mieux possible le code source des TP 1 et 2

Aide : Au besoin, n'hésitez pas à contacter l'enseignant