##

## INF-111 Travail pratique #2

**Partie 1**

Pour qui voter ?

**Remise : 6 novembre** **2015**

Travail en équipe de deux maximum

# Projet

Réaliser et implémenter un index permettant de retracer les députés d’une élection fédérale canadienne selon la circonscription pour laquelle ils sont inscrits. Les données proviennent d’un fichier texte dont le format est connu.

1. **Objectifs du travail**

Ce travail a pour but de valider votre compréhension des concepts suivants dans un environnement orienté-objet :

* Création de classe représentant un objet simple (bean).
* Création d’une classe comprenant des collections d’objets.
* Utilisation de tableau 2D.
* Utilisation d’objets de classes existantes en Java et de classes personnelles.

**Tous les attributs de ces classes doivent être privés**

1. **Description générale**

Le programme à écrire consiste à obtenir d’un fichier texte qui contient des données pour une élection fédérale au Canada. Cela dans le but d’obtenir facilement diverses informations sur les députés, les partis et les circonscriptions.

Le fichier dont nous parlons nous fournit les données sur les circonscriptions les partis et les députés. L’ordre des données sur une même ligne est :

* Le numéro de circonscription
* Le nom de circonscription
* Le nom du parti
* nom du député

Chaque information est séparée par le caractère tabulateur (‘\t’) et chaque ligne se termine par un symbole de fin de ligne (voir fichier texte fourni avec l’énoncé).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24001 | ‘\t’ | Abitibi--Baie-James--Nunavik--Eeyou | ‘\t’ | Le Parti Vert du Canada | ‘\t’ | Benoit | ‘\t’ | Patrick | ‘\n’ |  |

1. **Problème et stratégie**

Il y a principalement deux difficultés dans ce projet. Le premier est de récupérer les données en provenance d’un fichier texte et le second est de permettre une recherche rapide des députés par circonscription. **Pour la suite de la lecture, nous présumons que vous avez consulté et compris le contenu du fichier texte fourni.**

Le problème que nous voulons résoudre provient que nous avons un fichier texte et qu’il y a beaucoup de répétition. Il n’est pas aisé d’obtenir des informations sans trier les données avant d’effectuer la recherche. Par exemple, si on veut tous les députés du parti « bloc québécois », il faut trier les données sur la colonne qui contient le nom du parti avant d’obtenir le résultat. Une façon de contourner le problème est d’utiliser Excel et les options de tri. Une autre est d’écrire une application qui lit les données du fichier et crée les classes nécessaires pour répondre facilement aux questions de l’utilisateur. C’est cette dernière option que nous retenons.

Dans une classe **Election**, nous avons décidé que de créer trois collections sans doublon avec respectivement les noms des circonscriptions, les noms de partiset les noms de député. Ces collections de chaînes de caractères (String) serviront éventuellement à obtenir les choix de l’utilisateur dans une boîte de dialogue (**JOptionPane.showInputDialog**). Nous allons retenir aussi pour chaque député, à quelle circonscription il est inscrit et de quel parti il est membre (classe **Depute**)[[1]](#footnote-1). Finalement, nous allons utiliser une collection d’objets pour retenir le numéro d’nue circonscription, son nom et le député élu (classe **Circonscription**)[[2]](#footnote-2).

Enfin, pour répondre rapidement à différentes requêtes, nous allons créer un index dont l’utilisation vous sera décrite dans une deuxième partie. L’index est un tableau à deux dimensions d’entiers qui permettra d’optimiser le résultat des requêtes de l’utilisateur.

1. **Description sommaire des classes**

**Constantes :** Cette classe vous est fournie avec des constantes statiques utiles au projet.

**Depute :** Cette classe à écrire n’a que deux attributs qui correspondent à leur position dans les deux collections qui contiennent leur nom dans la classe Election.

* le numéro du parti
* le numéro de circonscription

**Circonscription :** Cette classe à écrire contient le numéro et le nom de la circonscription et le numéro référant au député élu s’il y a lieu. Lorsqu’il n’y a pas d’élu, la valeur est Constantes.VIDE.

**Election :** Voici les attributs de cette classe à écrire. Toutes les collections sont sans doublon.

* L’année de l’élection
* Collection des noms de circonscription
* Collection des noms de parti
* Collection des noms de député
* Collection d’objets de la classe **Depute** précédente
* Collection d’objets de la classe **Circonscription** précédente
* Tableau 2D d’entiers pour l’index (int [][])

**UtilitaireFichier :** Module utilitaire à écrire qui lit un fichier texte et initialise une instance de la classe **Election**. Vous devez écrire le code pour lire le fichier et remplir les collections de l’objet reçu.

1. **Description détaillée des classes**

**Depute** : Écrivez seulement deux attributs privés de type entier et les méthodes nécessaires au projet (constructeur, getter, setter, …). Les attributs sont les numéros de case où se trouvent la circonscription et le numéro du nom de parti dans les collections associées de la classe **Election**.

**Circonscription :** Écrivez les trois attributs privés et les méthodes nécessaires. Un numéro, un nom et un numéro de case du député élu correspondant à sa position dans la collection des députés. S’il n’y en a pas encore, le numéro de l’élu est Constantes.VIDE.

**Election :** Cette classe représente un objet unique mais contient plusieurs collections. Écrivez vos attributs pour vos collections. La définition du tableau 2d pour l’index peut attendre. Vous devez écrire les méthodes qui ajoutent les noms aux collections. Ces méthodes ajoutent le nom à la bonne collection et retourne la position où le nom a été ajouté.

Écrivez seulement le constructeur qui reçoit une année en paramètres et qui crée les instances des collections vides.

Pour vos collections, vous pouvez utiliser une de celles de java.util : Vector, ArrayList, LinkedList, … Si vous voulez les utiliser, vous devez spécifier le type de la collection entre <> (Exemple : Vector<String> tabNomsParti = new Vector<String> ();) .

Consultez les API pour en comprendre l’utilisation.

Voici les en-têtes formels des méthodes qui doivent MINIMALEMENT[[3]](#footnote-3) être implémentées.

 **public** **int** ajouterCirconscription(String nom, **int** numero)

Si le nom n’existe pas déjà dans la collection associée, vous l’ajoutez et vous créez un objet de la classe Circonscription pour retenir le nom, le numéro et l’élu (mis à VIDE pour l’instant). Vous l’ajoutez à la collection de Circonscription. Vous retenez la position où il a été ajouté.

Sinon, vous retenez la position du nom trouvé pour la retourner

Vous retournez la position.

**public** **int** ajouterNomParti(String nomParti)

Si le nom n’existe pas déjà dans la collection associée, vous l’ajoutez et vous retournez la position où il a été ajouté ou celle où il a été trouvé dans la collection.

**public** **void** ajouterDepute(**int** circonscription, String nomDepute, **int** noParti)

Si le nom n’existe pas, vous l’ajoutez dans la collection associée et vous créez un objet de la classe **Depute** pour retenir le numéro de circonscription et le numéro du parti. Que vous ajoutez à la collectionde **Depute**. Pas besoin de retourner de position. Le nom reçu est la concaténation du nom et du prénom.

Pour faciliter les dialogues avec l’utilisateur plus tard, nous avons besoin de tableau statique de String pour utiliser **JOptionPane.showInputDialog**. Écrivez les méthodes qui obtiennent et retournent les tableaux statiques équivalents à la collection associée.

 **private** String[] obtenirNomsCirconscription(){

 }

 **private** String[] obtenirNomsParti(){

 }

 **private** String[] obtenirNomsDepute(){

 }

**Indices :**

* La plupart des collections de Java ont une méthode pour vérifier si un objet existe déjà (ex : contains).
* Utilisez toArray(T[]) qui retourne un tableau statique des données contenu dans la collection pour obtenir les tableaux nécessaires au dialogue.

**UtilitaireFichier :** Une seule méthode static publique qui ouvre un fichier sélectionné à l’aide de **javax.swing.JFileChooser**. Elle initialise l’objet de la classe Election. Toutes les collections sont remplies à partir du fichier sélectionné. On présume que le fichier texte respecte le format du fichier décrit précédemment incluant les tabulateurs.

public static void getCirconscriptions(Election election);

Vous devez donc ouvrir le ficher et lire son contenu **ligne par ligne** pour créer vos collectionsde l’objet de la classe **Election**. Voici quelques indices qui vous aideront à réaliser cette méthode.

* Pour sélectionner un fichier voir :
	+ javax.swing.JFileChooser
	+ File
	+ showOpenDialog
	+ JFileChooser.***APPROVE\_OPTION***
* Pour lire un fichier texte voir :
	+ Scanner(File)
	+ useDelimiter("\t") //Pour passer les tabulateurs
	+ nextInt()
	+ next()
	+ nextLine()
	+ hasNext() //False si c’est la fin de fichier

Il faut lire dans l’ordre :

* Le numéro de circonscription (nextInt)
* Le nom de circonscription (next)
* Le nom de parti (next)
* Le nom du député (next)
* Le prénom du député (nextLine)
* Algorithme général:

Vous utilisez les classes pour obtenir le fichier texte qui contient les circonscriptions (JFileChooser et Scanner). Tant que toutes les lignes n’ont pas été lue (hasNext() == true), obtenir le numéro et le nom de circonscription et utiliser la méthode prévue de la classe **Election**. Même chose pour le nom de parti et pour le nom de député.

Exemple : String nomParti = ficTexte.next();

 int noParti = ajouterNomParti(nomParti);

1. **Programme principal**

Dans la méthode main(), vous créez un objet de la classe **Election**. Ensuite, vous utilisez JOptionPane pour établir le dialogue avec l’utilisateur. Le programme demande si l’utilisateur veut :

* + 1. Le parti et la circonscription d’un députe
		2. Tous les députés d’un parti
		3. Tous les députés d’une circonscription.
		4. Quitter

Selon l’option choisie, pour l’instant, ne faites qu’afficher la liste des noms associée à l’option du menu.

Pour l’option 1, vous affichez le nom des députés dans une boîte de dialogue et vous affichez la valeur sélectionnée si l’utilisateur n’a pas annulé. Vous effectuez la même chose pour les deux autres options soit affichez les noms de parti et les noms de circonscription.

\*\*\*Les méthodes de la classe **Election** pour obtenir les tableaux de String devraient exister.

1. **Partie 2**

Vous obtiendrez l’énoncé de la partie 2 dès que vous envoyez la partie 1 fonctionnelle. Il ne sert à rien de commencer la partie 2 si la partie 1 ne fonctionne pas correctement. Plus vite vous envoyez la partie 1, plus vite vous commencez la partie 2.

Le programme complet doit être remis le 16 novembre.

1. **Barème de correction**

**Exécution** 50%

* Doit respecter le comportement décrit dans l’énoncé.
1. Classe **Depute** non implémentée
2. Classe **Circonscription** non implémentée
3. Classe **Election**
4. Collections de Java non utilisées.
5. Méthodes minimales non implémentées (page 4)
6. Programme principal non implémenté

**Qualité de programmation** 50%

* Les erreurs pénalisantes concernant le respect des normes sont le mêmes que pour le tp1. **Vous devez en plus générer les commentaires Javadoc.**
1. Identificateurs ne respectent pas la norme.
2. Aération et/ou indentation et/ou impression laissent à désirer (**80 colonnes max**).
3. Découpage en sous programmes insuffisants.
4. Répétition inutile de code dû au manque de sous programmes.
5. Non utilisation d’un sous-programme lorsque c’est possible ou exigé.
6. Répétition inutile de code dû à l’incompréhension de l’utilité du paramétrage.
7. Constantes non définies.
8. Constantes non utilisées lorsque possible (même dans les commentaires).
9. Constantes en minuscules.
10. Commentaire d’en-tête de programme manquant (explication, auteurs et version).
11. Commentaires des constantes manquants.
12. Commentaires des variables manquants.
13. **Commentaire Javadoc manquant (**spécification)
14. Commentaires non judicieux ou inutile.
15. **Commentaires manquants sur la stratégie employée dans chaque sous-programme dont l’algorithme n’est pas évident et nécessite réflexion.**
16. Commentaires mal disposés.
17. Code inutile.
18. Sous-programme qui fait plus d’une tâche.
19. Affichage dans une fonction de calcul.
20. Qualité du français dans les commentaires.
21. Non-respect des droits d’auteurs.
22. Non utilisation de boucle lorsque possible.
23. Non utilisation de switch-case lorsque possible.
24. Non utilisation de l’opérateur ternaire lorsque possible.
25. Autres (s’il y a une pratique non énumérée qui ne fait pas de sens).

Bon travail!

1. Cela sera utile pour la partie 2 au moment de générer l’index. [↑](#footnote-ref-1)
2. Utile au tp3. [↑](#footnote-ref-2)
3. Si vous avez le prétexte, n’hésitez pas à créer des sous-programmes locaux privés. [↑](#footnote-ref-3)