Lycée des Arts et Métiers

2CI – Sciences de la programmation
Exercice de révision

Ouvrez le projet NetBeans « VotingSystem » qui se trouve dans votre dossier afin de le compléter comme indiqué ci-dessous.

Question 1

Il s'agit de réaliser un programme permettant de simuler un vote en ligne.

Le diagramme UML se trouve tout à la fin de l'énoncé !

La classe «Candidate»

1. Créez la classe **`Candidate`**, qui représente un candidat d'une liste d'élection, à l'aide des automatismes de NetBeans tout en respectant les contraintes suivantes :
   - L'attribut **`votes`** est initialisé directement à zéro.
   - Les deux autres attributs sont initialisés par le constructeur.
   - Tous les attributs possèdent des manipulateurs et des accesseurs.
2. La méthode **`voteFor`**() incrémente le nombre de votes du candidat d'une unité.

3. Ajoutez la méthode **`toString()`** qui retourne une représentation textuelle d'un candidat sous la forme

   `<votes> : <name> <firstname>`

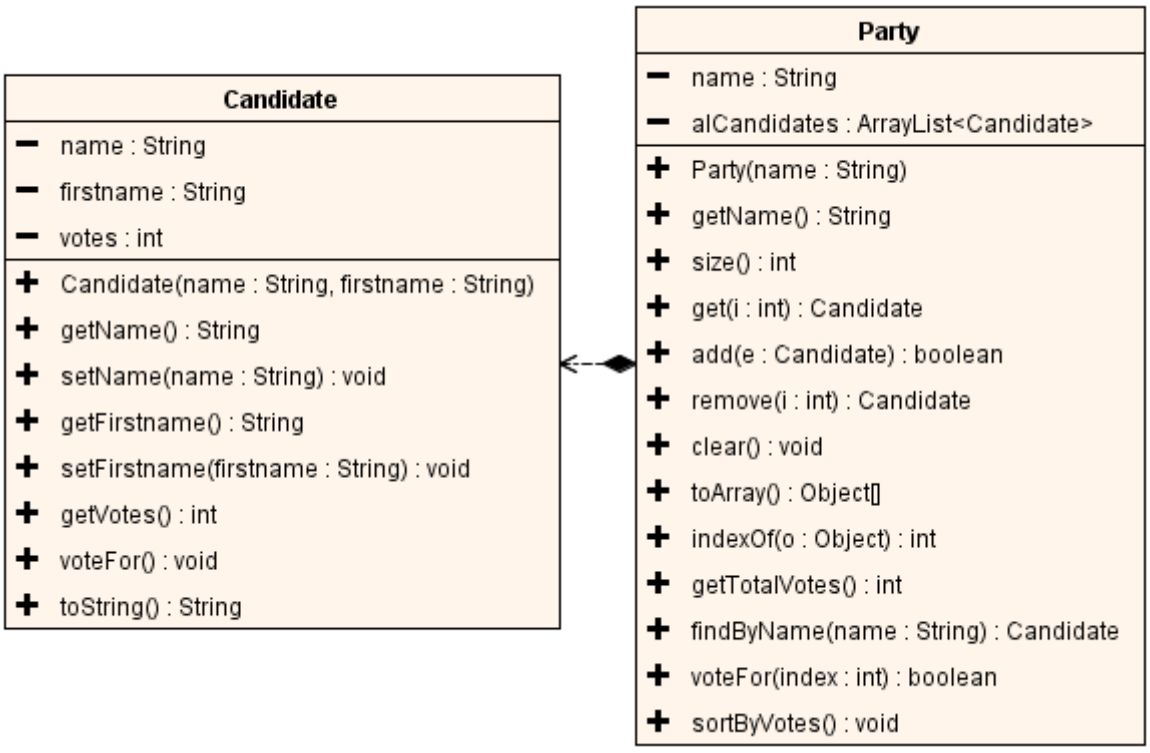
   où **`<name>`** est le nom du candidat, **`<firstname>`** est son prénom et **`<votes>`** la représentation textuelle du nombre de votes.

4. Faire en sorte que la partie **`<votes>`** de la méthode **`toString()`** de la classe **`Candididate`** soit composée d'exactement 5 symboles. S'il y en a moins, il faut rajouter le nombre d'espaces nécessaires au début.

(Voir programme de démonstration !)

La classe « Party »

5. Ajoutez maintenant la classe **`Party`** qui représente une liste d'élection. Cette classe possède donc une liste de candidats.
6. Générez le code pour les méthodes **`size()`**, **`get(...)`**, **`add(...)`**, **`remove(...)`**, **`clear()`**, **`toArray()`** et **`indexOf(...)`**. **(→ voir UML)**
7. Si la position indiquée par le paramètre est une position valide d'un candidat, la méthode **`voteFor(int)`** rajoute un vote au candidat concerné et retourne la valeur **`true`**, sinon elle retourne la valeur **`false`**.
8. La méthode **`getTotalVotes()`** calcule et retourne le nombre total de votes obtenus par chaque candidat.
9. La méthode **`findByName(String)`** recherche dans la liste le candidat dont le nom est identique à celui passé en tant que paramètre. La méthode retourne le candidat éventuellement trouvé. Si la liste ne contient pas un tel candidat, la valeur **`null`** est à retourner.
10. La méthode **`sortByVotes()`** trie la liste des candidats suivant l'ordre ***décroissant*** de leur nombre de votes (donc celui avec le plus haut score se trouve tout en haut de la liste) à l'aide de l'algorithme du tri par sélection directe du maximum.



La classe « MainFrame »

11. Décommentez la méthode **`initParty(String)`** qui se trouve déjà dans le code source. Vous allez l'utiliser afin de disposer dès le démarrage du programme de quelques candidats à élire, ce qui vous aide à tester votre programme.
12. Ajoutez un attribut du type **`Party`** et initialisez-le à l'aide de la méthode **`initParty(String)`**. Indiquez comme paramètre le nom de l'une des listes électorales prédéfinies (à vous d'en choisir une), c.-à-d. : passez à la méthode l'un des noms suivants : adr, csv, dp, greng, kpl, lenk, lsap, piraten
13. Ajoutez la méthode **`updateView()`** qui met à jour la liste visuelle : (→ démo !)
    - afficher le nom de la liste d'élection dans le libellé y prévu,
    - afficher le nombre de candidats,
    - afficher le nombre total de votes de tous les candidats,
    - trier la liste suivant le nombre de votes,
    - afficher la liste.
14. En cliquant sur l'un des boutons « Vote for selected candidate », le candidat sélectionné reçoit un vote, puis la vue est mise à jour. Si aucun candidat n'a été sélectionné, rien ne se passe.
15. En cliquant sur le bouton « Search », le candidat dont le nom correspond au nom entré dans le champ d'édition doit être sélectionné (s'il existe).