Aix*Marseille université
Socialement engagée

Année 2023-24

Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Programmation avancée : TP 4 bonus
Code UE : SBICU05

Gestion d'élection

Introduction

Gestion d'élection

Dans ce sujet de TP, on va s'intéresser à un système permettant de simuler des votes. Il y aura donc des classes correspondant à des citoyens (*citizens*), des bureaux de vote (*polling places*) et des résultats d'élection (*election results*).

Consignes pour le TP

- Toutes les variables de type `List` seront à initialiser avec des objets de type `ArrayList`. Vous pouvez vous référer à l'annexe en fin du sujet pour avoir une documentation des méthodes de `List`.
- Suivez les commentaires `// TODO` pour effectuer les changements sur le code et la documentation. Une fois un changement effectué, vous devez enlever le commentaire correspondant.
- Votre code doit passer les tests déjà écrits (il ne faut pas les modifier). Pour les lancer, il faut passer par `Gradle -> verification -> test`

Comme pour le TP 2, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du TP 2. Le lien vers le projet à forker est le suivant : https://etulab.univ-amu.fr/alaboure/hotel-template.

Une fois le dépôt téléchargé, vous pouvez compiler et exécuter le programme en cliquant deux fois sur `hotel -> application -> run`.

Pour exécuter les tests, il faut passer par l'onglet *gradle* à droite et cliquer deux fois sur `hotel -> Tasks -> verification -> test`. Pour le moment, les tests ne passeront pas, car certaines classes sont incomplètes.

Citoyen : classe `Citizen`

On considère donc la classe `Citizen` permettant de modéliser des citoyens. Elle contient les attributs suivants :

- attributs d'instance :
  - `firstName` le prénom du citoyen,
  - `lastName` le nom de famille du citoyen,
  - `age` l'age du citoyen exprimer en nombre d'années entier,
  - `citizenId` le numéro d'électeur du citoyen ;
- attributs de classe :
  - `VOTING_AGE` correspondant à l'age de majorité électoral en France, c'est-à-dire 18,
  - `citizenCount` correspondant au nombre d'instances de `Citizen` qui ont été créées.

Tâche 1 : *Ajouter dans la classe `Citizen` les déclarations des attributs `VOTING_AGE`, `citizenCount`, `firstName`, `lastName`, `age` et `citizenId`.*

Tâche 2 : *Compléter le code du constructeur de la classe `Citizen` qui permet d'instancier un citoyen avec un prénom, un nom de famille et un âge. Un citoyen a pour numéro d'identifiant le nombre de citoyens qui ont été instanciés avant son instanciation. Le nombre de citoyens instanciés doit évidemment être mis à jour.*

La classe `Citizen` contient les méthodes suivantes :

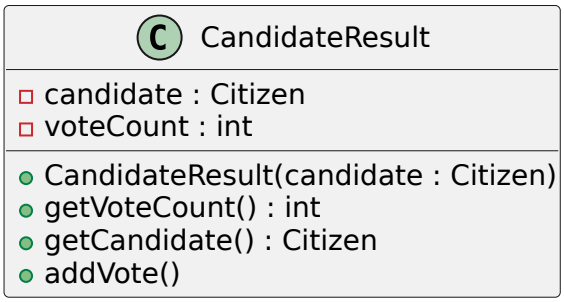
- méthodes d'instance :
  - `incrementAge` qui augmente d'un l'age d'un citoyen,
  - `canVote` de la classe `Citizen` qui renvoie `true` si le citoyen a l'age de voter et `false` sinon
  - `getUpperCaseLastName` qui renvoie le nom de famille du citoyen en majuscules (par exemple pour un citoyen ayant pour nom de famille `lAbourel`, la méthode devra renvoyer `LABOUREL`),
  - `getCapitalizedFirstName` de la classe `Citizen` qui renvoie le prénom du citoyen avec la première lettre en majuscule et toutes les autres lettres en minuscules (par exemple pour un citoyen ayant pour prénom `aRnauD`, la méthode devra renvoyer `Arnaud`),
  - `getName` qui renvoie le nom complet du citoyen, c'est-à-dire le prénom et le nom de famille avec un espace entre les deux. Le prénom et le nom devront être au format des deux méthodes précédentes (par exemple pour un citoyen ayant pour prénom `aRnauD` et pour nom `lAbourel`, la méthode devra renvoyer `Arnaud LABOUREL`),
  - `getAge` qui renvoie l'âge du citoyen,
  - `getCitizenID` qui renvoie le numéro d'électeur du citoyen,
  - `equals` de la classe `Citizen` qui renvoie `true` si l'objet passé en argument correspond à un citoyen ayant le même numéro d'électeur ;
- méthode de classe :
  - `resetCitizenCount` qui remet à zéro `citizenCount`.

Tâche 3 : *Changer le code des méthodes `incrementAge`, `canVote`, `getUpperCaseLastName`, `getCapitalizedFirstName`, `getName`, `getAge`, `getCitizenId`, `equals` et `resetCitizenCount`.*

Tâche 4 : *Activez les tests sur les méthodes de `Citizen` en supprimant les `@Disabled` devant chaque méthode de test de `CitizenTest` dans `src/test/java`. Vérifiez que tous les tests passent et corriger le code que vous avez écrit le cas échéant.*

Résultat d'un candidat : classe `CandidateResult`

On considère la classe `CandidateResult` qui permet de gérer les résultats d'élection d'un candidat.



Une nouvelle instance de `CandidateResult` contient initialement un nombre de votes (`voteCount`) de 0 et une instance de `Citizen`. La méthode `addVote` permet d'ajouter un vote au résultat du candidat.

Tâche 5 : *Compléter le code de la classe `CandidateResult`.*

Tâche 6 : *Créer une classe `CandidateResultTest` dans `src/test/java` testant les méthodes `getVoteCount`, `getCandidate` et `addVote` de la classe `CandidateResult`.*

Classe utilitaire `Percentages`

Pour afficher les résultats d'une élection, on aura besoin d'afficher des pourcentages. Pour cela, vous allez devoir compléter la méthode `toString` de la classe `Percentages`. Cette méthode devra à partir d'une proportion comprise entre 0 et 1, produire une chaîne de caractères représentant un pourcentage (un entier entre 0 et 100). On utilisera `Math.round` pour obtenir le pourcentage entier à afficher. Par exemple, `Percentages.toString(0.009)` renverra la chaîne de caractères `1%`.

Tâche 7 : *Compléter le code de la méthode `toString` de la classe `Percentages`.*

Tâche 8 : *Activez les tests sur la méthode `toString` de la classe `Percentages` en supprimant le `@Disabled` devant la méthode de test `testToString` de `PercentagesTest` dans `src/test/java`. Vérifiez que le test passe et corriger le code que vous avez écrit le cas échéant.*

Résultat d'élection : classe `ElectionResult`

La classe `ElectionResult` permet de représenter les résultats d'une élection. Elle contient les attributs d'instance suivants :

- `candidateResults` : la liste des résultats des candidats,
- `nullVotes` : le nombre de votes nuls et
- `voterTurnout` : le taux de participation pour l'élection (valeur comprise entre 0 et 1 représentant la proportion d'électeurs enregistrés qui ont voté).

La classe `ElectionResult` possède un constructeur qui prend en paramètre une liste de candidats et un taux de participation.

La classe `ElectionResult` contient les méthodes d'instance suivantes :

- `addVote` qui prend en argument un bulletin de vote représenté par une chaîne de caractères et qui ne renvoie rien, si le bulletin correspond (chaînes de caractères ayant les mêmes caractères dans le même ordre) au nom complet d'un des candidats, la méthode ajoute un vote au résultat du candidat, sinon la méthode ajoute un vote aux votes nuls ;
- `getExpressedVotes` qui n'a pas d'argument et qui renvoie le nombre de votes exprimés (votes correspondants à un candidat et donc non-nuls) ;
- `getNullVotes` qui n'a pas d'argument et qui renvoie le nombre de votes nuls ;
- `getVoterTurnout` qui n'a pas d'argument et qui renvoie le taux de participation pour l'élection (valeur comprise entre 0 et 1 représentant la proportion d'électeurs enregistrés qui ont voté) ;
- `print` qui n'a pas d'argument, ne renvoie rien et affiche les résultats des candidats (format à valider via les tests).

Tâche 9 : *Changer le code des méthodes `addVote`, `getExpressedVotes`, `getNullVotes`, `getVoterTurnout` et `print` de la classe `ElectionResult`.*

Tâche 10 : *Activez les tests sur les méthodes de `ElectionResult` en supprimant les `@Disabled` devant chaque méthode de test de `ElectionResultTest` dans `src/test/java`. Vérifiez que tous les tests passent et corriger le code que vous avez écrit le cas échéant.*

Bureau de vote : classe `PollingPlace`

La classe `PollingPlace` permet de modéliser des bureaux de votes. Elle contient les attributs d'instance suivants :

- `registeredVoters` : une liste de citoyens correspondant aux électeurs enregistrés du bureau de vote,
- `participatingVoters` : une liste de citoyens correspondant aux électeurs ayant voté du bureau de vote

et

— `ballots` : la liste des bulletins sachant que chaque bulletin est représenté par une chaîne de caractères.

Tâche 11 : *Ajouter dans la classe `PollingPlace` les attributs `registeredVoters`, `participatingVoters` et `ballots`.*

Tâche 12 : *Compléter le constructeur de la classe `PollingPlace` qui permet d'instancier un bureau de vote à partir d'une liste `possibleVoters` de citoyens donnée en argument. Le bureau de vote aura pour liste d'électeurs enregistrés les citoyens de `possibleVoters` qui ont l'age de voter, une liste vide d'électeur ayant voté et une liste vide de bulletins.*

La classe `PollingPlace` contient les méthodes suivantes :

— `acceptVoteFrom` qui renvoie **`true`** si le citoyen passé en argument a le droit de voter, c'est-à-dire qu'il est enregistré dans le bureau de vote et qu'il n'a pas déjà voté et **`false`** sinon ;

— `castBallot` qui prend un citoyen et bulletin en argument, si le citoyen a le droit de voter, elle stocke son bulletin dans la liste de bulletins, ajoute le citoyen aux électeurs ayant voté et renvoie **`true`**, dans le cas contraire elle ne fait rien à part renvoyer **`false`** ;

— `voterTurnout` qui renvoie le taux de participation, c'est-à-dire la proportion d'électeurs enregistrés qui ont voté (valeur entre 0 et 1) ;

— `countTheVotes` qui crée un résultat d'élection à partir des candidats passés en paramètre et ajoute tous les bulletins aux résultats de l'élection (c'est-à-dire comptabilise les votes exprimés par les bulletins stockés).

Tâche 13 : *Compléter la méthode `acceptVoteFrom`, `castBallot`, `voterTurnout` et `countTheVotes` de la classe `PollingPlace`.*

Tâche 14 : *Activez les tests sur les méthodes de `PollingPlace` en supprimant les `@Disabled` devant chaque méthode de test de `PollingPlaceTest` dans `src/test/java`. Vérifiez que tous les tests passent et corriger le code que vous avez écrit le cas échéant.*