Aix*Marseille université
Socialement engagée

Année 2023-24

Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Programmation avancée : Examen
Code UE : SBICU05

Examen : gestion d'élection

Consignes

- Pour les chaînes de caractères et les listes, vous avez une annexe donnant les méthodes utiles sur `String` et `List`.
- Pour les tests, vous avez une annexe donnant les assertions de base.
- vous avez accès à https://ametice.univ-amu.fr/.
- Vous avez 2 heures pour composer.
- 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`
- Lorsque vous avez fini de composer, assurez-vous d'avoir bien mis votre projet dans le répertoire *exam* puis lancez par double-clic le programme *CLIQUE MOI FORT EN FIN D'EXAM* (situé dans leur espace temporaire), puis déconnectez-vous au plus vite.

Gestion d'élection

Dans ce sujet d'examen, 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*).

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.

Question 1 (1,5 points) : *Ajouter dans la classe `Citizen` les déclarations des attributs `VOTING_AGE`, `citizenCount`, `firstName`, `lastName`, `age` et `citizenId`.*

Question 2 (1 point) : *Compléter le code du constructeur de la classe `Citizen` qui permet d'instancier un citoyen avec un prénom, un nom de famille et un age. 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.*

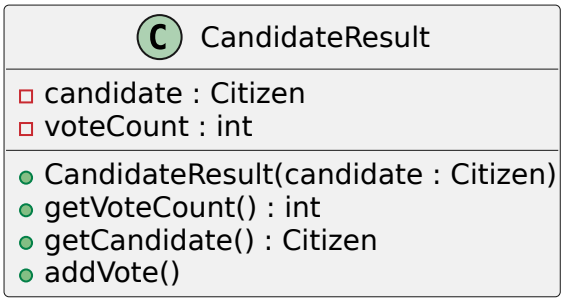
La classe `Citizen` contient les méthodes suivantes :

- 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`.

Question 3 (4 points) : *Changer le code des méthodes `incrementAge`, `canVote` , `getUpperCaseLastName`, `getCapitalizedFirstName`, `getName`, `getAge`, `getCitizenId`, `equals` et `resetCitizenCount`.*

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.

Question 4 (1,5 points) : *Compléter le code de la classe `CandidateResult`.*

Question 5 (1,5 points) : *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%`.

Question 6 (0,5 points) : *Compléter le code de la méthode `toString` de la classe `Percentages`.*

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 vote 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 vote 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).

Question 7 (2,5 points): *Changer le code des méthodes `addVote`, `getExpressedVotes`, `getNullVotes`, `getVoterTurnout` et `print` de la classe `ElectionResult`.*

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.

Question 8 (1 point) : *Ajouter dans la classe `PollingPlace` les attributs `registeredVoters`, `participatingVoters` et `ballots`.*

Question 9 (1 point) : *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).

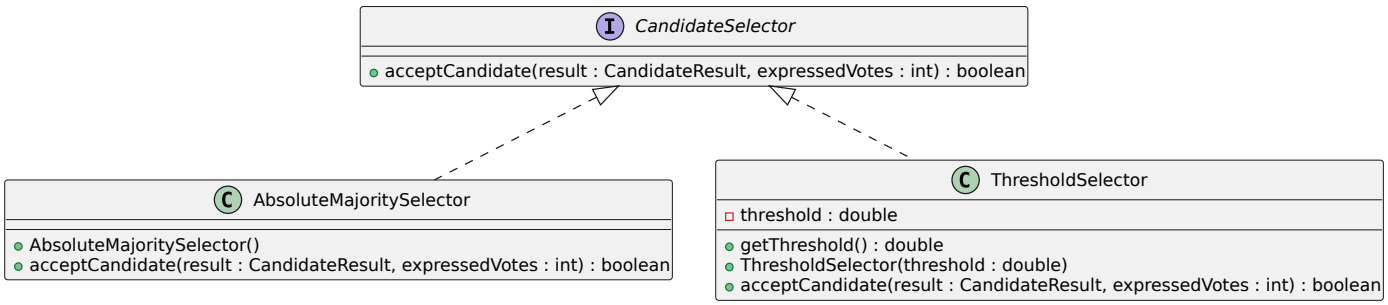
Question 10 (3 points) : *Compléter la méthode `acceptVoteFrom`, `castBallot`, `voterTurnout` et `countTheVotes` de la classe `PollingPlace`.*

Interface `CandidateSelector`

On souhaite définir deux classes ayant une méthode `acceptCandidate` qui accepte ou non un candidat à partir du résultat du candidat et du nombre de vote exprimé :

— `AbsoluteMajoritySelector` qui accepte les candidats ayant strictement plus que la moitié des votes exprimés.
— `ThresholdSelector` qui accepte les candidats dont le nombre de votes correspond à une proportion supérieure ou égale à un seuil (*threshold*). Par exemple, pour un seuil `0.15`, les candidats acceptés seront ceux qui ont au moins 15% des votes.

Ces deux classes ont les diagrammes suivants :



Question 11 (2 points) : *Créez les classes `AbsoluteMajoritySelector` et `ThresholdSelector` dans le dossier `src/main/java`.*

Question 12 (0,5 point) : *Complétez le code de la méthode `List<Citizen>`*

selectedCandidates(CandidateSelector selector) *ElectionResult*

selector

Annexe

Documentation de la classe `String`

Voici un extrait de la documentation de la classe `String` :

```java
/**
 * Returns a string that is a substring of this string. The
 * substring begins with the character at the specified index and
 * extends to the end of this string.
 * Examples:
 * "unhappy".substring(2) returns "happy"
 * "Harbison".substring(3) returns "bison"
 * "emptiness".substring(9) returns "" (an empty string)
 *
 * @param      beginIndex   the beginning index, inclusive.
 * @return     the specified substring.
 */
public String substring(int index){/* code */}
/**
 * Returns a string that is a substring of this string. The
 * substring begins at the specified {@code beginIndex} and
 * extends to the character at index {@code endIndex - 1}.
 * Thus the length of the substring is {@code endIndex-beginIndex}.
 * Examples:
 * "hamburger".substring(4, 8) returns "urge"
 * "smiles".substring(1, 5) returns "mile"
 *
 * @param      beginIndex   the beginning index, inclusive.
 * @param      endIndex     the ending index, exclusive.
 * @return     the specified substring.
 */
public String substring(int beginIndex, int endIndex){/* code */}
/**
 * Converts all of the characters in this {@code String} to lower
 * case using the rules of the default locale.
 * @return  the {@code String}, converted to lowercase.
 */
public String toLowerCase(){/* code */}
/**
 * Converts all of the characters in this {@code String} to upper
 * case using the rules of the default locale.
 * @return  the {@code String}, converted to uppercase.
 */
public String toUpperCase(){/* code */}
```

Documentation de `List`

Pour cet examen, toutes les variables de type `List` seront à initialiser avec des objets de type `ArrayList`.

Vous pouvez utilisez les méthodes suivantes de `List<E>` :

- `boolean add(E e)` : Appends the specified element to the end of this list.
- `int size()` : Returns the number of elements in this list.
- `E get(int index)` : Returns the element at the specified position in this list.
- `boolean remove(Object o)` : Removes the first occurrence of the specified element from this list, if it is present.
- `boolean isEmpty()` : Returns `true` if this list contains no elements.

Rappels sur les tests

Une classe de test par classe à tester. Une méthode de test par méthode ou cas à tester.

Le code d'une classe de test testant une classe `NameTestedClass` a le format suivant :

```
1 import org.junit.jupiter.api.Test;
2 import static org.assertj.core.api.Assertions.*;
3
4 public class NameTestedClassTest {
5   @Test
6   void testTestMethod(){
7     // code containing assertions
8   }
9 }
```

Pour tester, on utilise des assertions qui doivent être vraies. Vous trouverez ci-dessous une listes des assertions les plus utiles.

- `assertThat(object).isNotNull()` : vérifie que la référence **n'**est **pas** `null`
- `assertThat(actual).isSameAs(expected)` : vérifie que les deux objets sont les mêmes (même référence : utilisation de `==`).
- `assertThat(condition).isTrue()` : vérifie que `condition` est vraie.
- `assertThat(condition).isFalse()` : vérifie que `condition` est faux.
- `assertThat(actual).isEqualTo(expected)` : vérifie que `expected` est égal à `actual` (en appelant `equals` sur `actual`).
- `assertThat(actual).isNotEqualTo(expected)` : vérifie que `expected` **n'**est égal **pas** à `actual` (en appelant `equals` sur `actual`).
- `assertThat(iterable).contains(e1, e2, ..., ek)` : vérifie que `iterable` (qui peut être une `List`, un tableau, ...) contient `e1, e2, ..., ek`.
- `assertThat(iterable).containsOnly(e1, e2, ..., ek)` : vérifie que `iterable` (qui peut être une `List`, un tableau, ...) contient exactement `e1, e2, ..., ek` dans cet ordre.
- `assertThat(actual).isCloseTo(expected, within(delta))` : vérifie que $|expected - actual| \leq delta$ (comparaison de double).