Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Année 2023-24

Programmation avancée : Corrigé examen
Code UE : SBICU05

Corrigé examen : gestion d'élection

Citoyen : classe `Citizen`

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

```java
public class Citizen {
  private final static int VOTING_AGE = 18;
  private static int citizenCount = 0;

  private final String lastName;
  private final String firstName;
  private final int citizenId;
  private int age;
}
```

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.*

```java
public class Citizen {
  public Citizen(String firstName, String lastName, int age) {
    this.lastName = lastName;
    this.firstName = firstName;
    this.age = age;
    citizenId = citizenCount;
    citizenCount++;
  }
}
```

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

```java
public class Citizen {
  public void incrementAge(){
```

```
    age++;
  }

  public boolean canVote(){
    return age >= VOTING_AGE;
  }

  private String getCapitalizedFirstName(){
    return firstName.substring(0,1).toUpperCase()
      + firstName.substring(1).toLowerCase();
  }

  private String getUpperCaseLastName(){
    return lastName.toUpperCase();
  }

  public String getName() {
    return  getCapitalizedFirstName() + " "
      + getUpperCaseLastName();
  }

  public int getCitizenId() {
    return citizenId;
  }

  @Override
  public boolean equals(Object o) {
    if (!(o instanceof Citizen))
      return false;
    Citizen citizen = (Citizen) o;
    return citizenId == citizen.citizenId;
  }

  public int getAge() {
    return age;
  }

  public static void resetCitizenCount(){
    citizenCount = 0;
  }
}
```

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

```
public class CandidateResult{
  private final Citizen candidate;
  private int voteCount;

  public CandidateResult(Citizen candidate) {
    this.candidate = candidate;
    this.voteCount = 0;
  }

```

```java
  public int getVoteCount() {
    return voteCount;
  }


  public void addVote(){
    voteCount++;
  }

  public Citizen getCandidate() {
    return candidate;
  }
}
```

Question 5 (1,5 points) : *Créer une classe `CandidateResultTest` dans `src/test/java` testant les méthodes `getVoteCount`, `getCandidate` et `addVote` de la classe `CandidateResult`.*

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

public class CandidateResultTest {
  private Citizen candidate;

  @BeforeEach
  void initializeCandidate(){
    candidate = new Citizen("arnaud", "labourel", 41);
  }

  @Test
  void testGetVoteCount(){
    CandidateResult candidateResult = new CandidateResult(candidate);
    assertThat(candidateResult.getVoteCount()).isEqualTo(0);
  }

  @Test
  void testAddVote(){
    CandidateResult candidateResult = new CandidateResult(candidate);
    assertThat(candidateResult.getVoteCount()).isEqualTo(0);
    for(int index = 0; index < 10; index++){
      candidateResult.addVote();
      assertThat(candidateResult.getVoteCount()).isEqualTo(index + 1);
    }
  }

  @Test
  void testGetCandidate(){
    CandidateResult candidateResult = new CandidateResult(candidate);
    assertThat(candidateResult.getCandidate()).isSameAs(candidate);
  }

}
```

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

```
public class Percentages {
  public static String toString(double percentage){
    return Math.round(percentage * 100.) + "%";
  }
}
```

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

```
public class ElectionResult {

  public void addVote(String ballot){
    for(CandidateResult result : candidateResults)
      if(result.getCandidate().getName().equals(ballot)) {
        result.addVote();
        return;
      }
    nullVotes++;
  }

  public void print(){
    System.out.println("Expressed votes: " + getExpressedVotes());
    for(CandidateResult result : candidateResults)
      System.out.println(result.getCandidate().getName() + ": "
              + Percentages.toString(proportionOfExpressedVotes(result)));
    System.out.println("Null votes: " + nullVotes);
    System.out.println("Voter turnout: " + Percentages.toString(voterTurnout))
        ;
  }

  private double proportionOfExpressedVotes(CandidateResult result) {
    return result.getVoteCount() / (double) getExpressedVotes();
  }

  public int getExpressedVotes(){
    int sumOfVotes = 0;
    for(CandidateResult result : candidateResults){
      sumOfVotes += result.getVoteCount();
    }
    return sumOfVotes;
  }

  public int getNullVotes() {
    return nullVotes;
  }

  public double getVoterTurnout() {
    return voterTurnout;
```

```
39    }
40  }
```

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

```
1  public class PollingPlace {
2    private final List<Citizen> registeredVoters;
3    private final List<Citizen> participatingVoters;
4    private final List<String> ballots;
5  }
```

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.*

```
1    public PollingPlace(List<Citizen> possibleVoters) {
2      this.registeredVoters = new ArrayList<>();
3      for(Citizen citizen : possibleVoters)
4        if(citizen.canVote())
5          registeredVoters.add(citizen);
6      this.participatingVoters = new ArrayList<>();
7      this.ballots = new ArrayList<>();
8    }
```

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

```
1  public class PollingPlace {
2
3    public boolean castBallot(Citizen citizen, String ballot) {
4      if (!acceptVoteFrom(citizen))
5        return false;
6      participatingVoters.add(citizen);
7      ballots.add(ballot);
8      return true;
9    }
10
11   public boolean acceptVoteFrom(Citizen citizen) {
12     return registeredVoters.contains(citizen)
13             && !participatingVoters.contains(citizen);
14   }
15
16   public double voterTurnout(){
```

```
    return participatingVoters.size() / (double) registeredVoters.size();
  }

  public ElectionResult countTheVotes(List<Citizen> candidates){
    ElectionResult result = new ElectionResult(candidates, voterTurnout());
    for(String ballot : ballots){
      result.addVote(ballot);
    }
    return result;
  }
}
```

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

```
public class AbsoluteMajoritySelector implements CandidateSelector{
  @Override
  public boolean acceptCandidate(CandidateResult result, int expressedVotes) {
    return result.getVoteCount() > expressedVotes/2;
  }
}
```

```
public class ThresholdSelector implements CandidateSelector {
  private final double threshold;

  public ThresholdSelector(double threshold) {
    this.threshold = threshold;
  }

  public boolean acceptCandidate(CandidateResult result, int expressedVotes) {
    return result.getVoteCount() >= threshold * expressedVotes;
  }
}
```

Question 12 (0,5 point) : *Complétez le code de la méthode `List<Citizen> selectedCandidates(CandidateSelector selector)` dans la classe `ElectionResult` qui renvoie parmi les candidats dans le résultat de l'élection, ceux qui sont acceptés par le `selector` passé en argument.*

```
public class ElectionResult {
  List<Citizen> selectedCandidates(CandidateSelector selector){
    List<Citizen> selectedCandidates = new LinkedList<>();
    for (CandidateResult result : candidateResults){
      if(selector.acceptCandidate(result, getExpressedVotes()))
        selectedCandidates.add(result.getCandidate());
    }
    return selectedCandidates;
  }
```

```
10 }
```