TP9

Exercice – vote électronique

Nous souhaitons réaliser une application de vote électronique avec différents modes de scrutin. Un scrutin consiste pour des électeurs à choisir un vote parmi un ensemble de votes possibles. Un vote est défini par le texte (ou nom) représentant ce vote, évidemment non modifiable. Il est possible de voter seulement si le scrutin n'est pas clos. Une fois clos, l'application détermine le résultat du scrutin. Le calcul du résultat (*i.e.*, la désignation du vainqueur) dépend du mode de scrutin. Nous considérons ici deux modes de scrutin :

— Un scrutin à la majorité absolue : est déclaré vainqueur le vote (autre que les votes blancs et nuls) qui a obtenu plus de 50% des suffrages exprimés (votes blancs et nuls non compris).

— Un scrutin à la majorité relative : est déclaré vainqueur le vote (autre que les votes blancs et nuls) qui a reçu le plus de voix parmi les suffrages exprimés, pour peu qu'au moins 15% des inscrits au scrutin aient exprimé ce vote.

Il est possible, dans les deux cas, qu'il n'y ait aucun vainqueur.

Toutes les classes de cet exercice devront appartenir au *package* `scrutin`.

1. Créez une classe `Vote` et écrivez les méthodes qui définissent la forme canonique de la classe (*i.e.*, constructeurs, accesseurs, `toString`, `equals` [1]).
2. Ajoutez 4 constantes correspondant à des votes particuliers : les votes `OUI`, `NON`, `BLANC` et `NUL`.
3. Créez une classe `MajoriteAbsolue` qui modélise le mode de scrutin à la majorité absolue. Cette classe doit notamment stocker les résultats sous la forme d'une table d'association entre un objet `Vote` et le nombre des suffrages qu'il a reçus. Toutefois, elle ne s'occupe pas des électeurs, ni du fait qu'ils ne peuvent voter qu'une seule fois.
   (a) Écrivez un constructeur qui prend en paramètre le nombre d'inscrits (électeurs), ainsi que l'ensemble des votes soumis au scrutin. Les votes blancs et nuls doivent être systématiquement ajoutés à ces votes.
   (b) Écrivez une méthode `Set<Vote> getVotesPossibles()` qui renvoie l'ensemble des votes possibles pour ce scrutin.
   (c) Écrivez une méthode `void ajouterVote(Vote v)` qui ajoute le vote `v` aux votes déjà effectués pour le scrutin. Tous les votes sont acceptés tant que le scrutin n'est pas clos, que le vote fasse partie ou non des votes possibles. Si le scrutin est clos, la méthode lève une exception non vérifiée de type `ScrutinClosException`. Si le vote `v` ne fait pas partie des votes autorisés par le scrutin alors il est considéré comme un vote nul.
   (d) Écrivez une méthode `void afficherResultats()` qui affiche pour chaque vote possible (y compris les votes blancs et nuls), le nombre de suffrages reçus une fois le scrutin clos. Si le scrutin n'est pas clos, la méthode lève une exception non vérifiée de type `ScrutinNonClosException`.
   (e) Écrivez une méthode `void cloturer()` qui clôt le scrutin et affiche les résultats.

1. Deux votes portant le même texte sont égaux (insensible à la casse).

(f) Écrivez une méthode `Vote getVainqueur()` qui renvoie le vote vainqueur ou `null` s'il n'y a pas de vainqueur. Si le scrutin n'est pas clos, la méthode lève une exception non vérifiée de type `ScrutinNonClosException`.

4. Créez une classe `MajoriteRelative` dont le comportement est similaire à celui de la classe `MajoriteAbsolue`, excepté pour la désignation du vainqueur (*i.e.*, la méthode `getVainqueur`). La gestion des égalités n'est pas demandée.
5. Créez une classe `TestScrutin` contenant une méthode `main` qui :
   (a) Crée un référendum, c'est-à-dire un scrutin à la majorité absolue dont les votes possibles sont `OUI` et `NON`, pour 200 votants,
   (b) Ajoute 175 votes choisis aléatoirement entre `OUI` et `NON`,
   (c) Clôt le scrutin, affiche les résultats et annonce le vainqueur.

Exercice – gestion d'un répertoire

Nous souhaitons réaliser une application de gestion d'un répertoire téléphonique permettant d'associer à un contact zéro ou plusieurs numéros de téléphone (portable, domicile, travail, fax, *etc.*). Elle doit notamment permettre :

— D'ajouter/supprimer un contact du répertoire
— D'ajouter/modifier/supprimer un numéro de téléphone d'un contact
— De rechercher un contact par son nom/prénom
— De rechercher un contact par son numéro de téléphone
— D'afficher la liste des contacts
— De sauvegarder/charger un répertoire

L'application est composée (au minimum) des trois classes suivantes : `NumeroTelephone`, `Contact` et `Repertoire`. Proposez deux implémentations où :

1. La classe `Repertoire` gère une liste de `Contact` qui elle-même gère une liste de `NumeroTelephone`.
2. La classe `Repertoire` gère une table d'association entre un `Contact` et une liste de `NumeroTelephone`.

L'application doit fonctionner indifféremment avec l'une ou l'autre implémentation. Comparez les deux implémentations.