

## Cinquième devoir (noté) : synthèse

J. Sam & J.-C. Chappelier

du 03 décembre au 21 décembre 2015

Ce devoir comprend trois exercices à rendre.

### 1 Exercice 1 — Jeu de la roulette

On s'intéresse ici à simuler des stratégies pour le jeu dit de la roulette.

La roulette est un jeu de hasard dans lequel chaque joueur mise des jetons sur une case. Les cases contiennent des numéros et sont de couleur rouge ou noire.

Quand les joueurs ont effectué leur mise, le croupier lance la bille et annonce « rien ne va plus ». À partir de ce moment, plus aucune mise n'est admise. Quand la bille s'arrête sur une case, les joueurs qui ont misé sur le numéro ou la couleur de la case reçoivent leur gain, calculé en fonction du type de mise ; sinon, ils perdent leur mise.

Télécharger le programme fourni sur le site du cours <sup>1</sup> et le compléter.

**ATTENTION :** vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernant spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :

---

1. <https://d396qusza40orc.cloudfront.net/intropoojava/assignments-data/Casino.java>

Window > Preferences > Java > Editor > Save Actions  
(et décocher l'option de reformatage si elle est cochée)

2. sauvegarder le fichier téléchargé sous le nom `Casino.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `[dossierDuProjetPourCetExercice]/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;

4. écrire le code à fournir entre ces deux commentaires :

```
/* *****  
 * Completez le programme a partir d'ici.  
 * ***** */  
  
/* *****  
 * Ne rien modifier apres cette ligne.  
 * ***** */
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Casino.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

## 1.1 Le code à produire

**Les mises** Il s'agit d'abord d'implémenter une classe `Mise` permettant de représenter les mises des joueurs.

Une *Mise* est simplement caractérisée par le *nombre de jetons misés* (un `int`).

La classe `Mise` comportera :

- un constructeur initialisant son attribut au moyen d'une valeur passée en paramètre ;
- un getter `getMise` retournant le nombre de jetons misés. Cette méthode ne devra pas être redéfinie dans d'éventuelles sous-classes de `Mise` ;
- une méthode `int gain(int n)` calculant et retournant le gain obtenu par la mise en fonction du numéro `n` tiré (celui sur lequel la bille s'est arrêtée lorsque la roulette a cessé de tourner). **Cette méthode ne peut être définie concrètement pour une mise quelconque.**

Pour simplifier, nous allons supposer que seuls les deux types de mises suivants sont possibles :

- les mises portant sur un numéro. Ces mises rapportent 35 fois le nombre de jetons misés ;
- les mises sur les cases rouges, lesquelles rapportent une seule fois la somme misee.

Il vous est demandé d'implémenter la hiérarchie de classes suggérée ci-dessus.

Vous créerez pour cela une classe `Pleine` pour représenter les mises sur un numéro (ce numéro sera passé au constructeur) et une classe `Rouges` pour représenter la mise sur les cases rouges.

Il s'agira bien sûr essentiellement d'implémenter les méthodes spécifiques de calcul de gain qui sont attachées à ces classes. Ces méthodes attribueront un gain si le numéro tiré correspond :

- au numéro de la mise pour les `Pleine` ;
- à la couleur de la mise pour les `Rouges`.

Le gain retourné est nul sinon.

Vous définirez en plus dans ces classes :

- des *constantes statiques* représentant :
  1. pour les `Pleine`, le facteur de multiplication de la mise (`FACTEUR_GAIN` valant 35) ;
  2. pour les `Rouges`, le tableau des numéros des cases rouges valant { 1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34 et 36 } (vous pourrez copier-coller ces valeurs depuis le commentaire du fichier fourni. Ce tableau s'appellera `ROUGES`).

**Les constantes seront publiquement accessibles.**

**Les joueurs** On représentera les joueurs d'une partie de roulette au moyen d'une classe `Joueur`. Chaque joueur possède un nom et une stratégie de mise (concrètement, un objet de type `Mise`).

Les méthodes associées à un joueur seront :

- un constructeur conforme à la méthode `main` fournie ;
- un getter `getNom` ;
- une méthode `void setStrategie(Mise mise)` permettant de modifier la stratégie du joueur (le genre de mise qu'il fait) ;
- une méthode `int getMise()` retournant le nombre de jetons utilisés par la stratégie courante du joueur. Si la stratégie courante vaut `null`, la méthode `getMise` retournera simplement zéro ;
- une méthode `int gain(int n)` retournant le gain permis par la stratégie courante du joueur si la case de numéro `n` est tirée. Si la stratégie courante vaut `null`, la méthode `gain` retournera simplement zéro.

**Jeu de Roulette** Pour représenter une partie de roulette, créez une classe `Roulette` comprenant (au moins) :

- un ensemble de joueurs (codé au moyen d'un `ArrayList`) ;
- le "gain" de la maison (du casino) : ce que la casino gagne ou perd au terme de la partie (un `int`) ;
- le numéro tiré.

La classe `Roulette` comportera au moins les méthodes suivantes :

- une méthode `participe` pour ajouter un joueur à l'ensemble des joueurs (voir des exemples d'utilisation dans la méthode `main` fournie). Le joueur sera toujours ajouté en fin de liste ;
- les getters `getTirage` retournant le numéro tiré, `getGainMaison` retournant le gain de la maison (initialisé à zéro à la construction) et `getParticipants` retournant le nombre de participants à la partie de roulette ;
- une méthode `rienNeVaPlus` qui donne une valeur au dernier numéro tiré ;  
Pour simplifier, nous tricherons ici et la méthode `rienNeVaPlus` prendra simplement en argument le numéro que l'on veut voir sortir. Ce numéro sera donc simplement affecté au numéro tiré (voir exemples dans la méthode `main` fournie) ;
- une méthode `int perteMise(int miseDuJoueur)` permettant de calculer combien un joueur perd de sa mise (cette perte revient au casino). On considérera qu'il n'est pas possible de faire ce calcul pour une partie

de roulette quelconque (deux types de jeu de roulette vont entrer en scène bientôt) ;

- une méthode `void calculerGainMaison()` mettant à jour le "gain" de la maison pour la partie en cours : pour une partie donnée, si un participant gagne (a un gain strictement positif), le casino perd le montant de son gain. S'il perd le casino gagne ce que le participant a perdu de sa mise (voir méthode précédente).
- une méthode `toString` qui génère la représentation d'une partie de roulette sous la forme d'une chaîne de caractères, selon le format suivant :

```
Croupier : le numéro du tirage est le <numéro tiré>
  Le joueur <nom joueur1> mise <montant1> et perd
  Le joueur <nom joueur2> mise <montant2> et gagne <gain2>
  ...
  Le joueur <nom joueurX> mise <montantX> et gagne <gainX>
Gain/perde du casino : <gain ou perte du casino>
```

Deux types de parties de roulette seront considérés :

- des parties de roulette française (classe `RouletteFrancaise`) pour lesquelles les joueurs perdent l'intégralité de leur mise ;
- des parties de roulette anglaise (classe `RouletteAnglaise`) pour lesquelles les joueurs ne perdent que la moitié de leur mise (nous simplifions bien sûr ici délibérément les règles usuelles de ces jeux). Par ailleurs les parties de roulette anglaise admettrons au plus 10 joueurs. Un joueur ne pourra être ajouté à la partie si le nombre maximal de joueurs est atteint. Une interface `ContrôleJoueurs` devra imposer à la classe `RouletteAnglaise` de définir une méthode `boolean check()` retournant `true` si un participant supplémentaire peut-être ajouté à la partie et `false` sinon.

## 1.2 Exemple de déroulement

```
Roulette anglaise :
Croupier : le numéro du tirage est le 12
  Le joueur Dupond mise 100 et perd
  Le joueur Dupont mise 30 et gagne 30
Gain/perde du casino : 20

Croupier : le numéro du tirage est le 1
  Le joueur Dupond mise 100 et gagne 3500
  Le joueur Dupont mise 30 et gagne 30
Gain/perde du casino : -3530

Croupier : le numéro du tirage est le 31
  Le joueur Dupond mise 100 et perd
  Le joueur Dupont mise 30 et perd
```

Gain/perte du casino : 65

Roulette française :

Croupier : le numéro du tirage est le 12

Le joueur Dupond mise 100 et perd

Le joueur Dupont mise 30 et gagne 30

Gain/perte du casino : 70

Croupier : le numéro du tirage est le 1

Le joueur Dupond mise 100 et gagne 3500

Le joueur Dupont mise 30 et gagne 30

Gain/perte du casino : -3530

Croupier : le numéro du tirage est le 31

Le joueur Dupond mise 100 et perd

Le joueur Dupont mise 30 et perd

Gain/perte du casino : 130

## 2 Exercice 2 — Gestion d'employés

On cherche ici à écrire un programme permettant de gérer les employés d'une entreprise d'informatique et leur salaire. Les employés (classe `Employe`) que l'on souhaite représenter sont caractérisés chacun par un nom (attribut `nom` qui ne changera pas une fois donné), un revenu mensuel et un taux d'occupation (pourcentage de temps travaillé par mois ; par exemple : emploi à 80%). Il existe trois types d'employés :

- les managers (classe `Manager`), caractérisés en plus par un nombre de jours voyagés et un nombre de nouveaux clients apportés ;
- les testeurs (classe `Testeur`), caractérisés par un nombre d'erreurs corrigées ;
- les programmeurs (classe `Programmeur`), caractérisés par un nombre de projets achevés.

Faites attention de bien veiller à une bonne encapsulation de toutes vos classes !

Télécharger le programme fourni sur le site du cours<sup>2</sup> et le compléter.

**ATTENTION :** vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :

---

2. <https://d396qusza40orc.cloudfront.net/intropoojava/assignments-data/Employes.java>

Window > Preferences > Java > Editor > Save Actions  
(et décocher l'option de reformatage si elle est cochée)

2. sauvegarder le fichier téléchargé sous le nom `Employes.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `[dossierDuProjetPourCetExercice]/src/`;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*
 * Completez le programme a partir d'ici.
 */

/*
 * Ne rien modifier apres cette ligne.
 */
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Employes.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

## 2.1 Le code à produire

**Employés et salaires** Commencez par doter chacune de vos classes d'un constructeur permettant d'initialiser *toutes* les valeurs des attributs concernés (dans un ordre compatible avec la méthode `main` fournie). Le taux d'occupation sera 100% par défaut.

De plus, si un constructeur reçoit un taux d'occupation inférieur à 10%, le taux effectivement retenu doit être de 10%. De même, si le taux d'occupation reçu est supérieur à 100%, il devra être limité à 100%.

A la construction d'un employé le message

Nous avons un nouvel employé : <...>

s'affichera (où <...> doit correspondre aux exemples de déroulement donnés plus bas).

Ajoutez ensuite une méthode double `revenuAnnuel()` qui calcule et retourne le salaire annuel comme suit :

- tout employé a un salaire de base qui vaut 12 fois son salaire mensuel multiplié par son taux d'occupation ;

- pour un manager, on ajoute un bonus de 500 francs pour chaque client apporté, et de 100 francs pour les dépenses de chaque jour voyagé ; des constantes publiques seront utilisées (`FACTEUR_GAIN_CLIENT` valant 500 et `FACTEUR_GAIN_VOYAGE` valant 100) ;
- pour un testeur, on ajoute un bonus de 10 francs pour chaque erreur corrigée ; une constante publique sera utilisée (`FACTEUR_GAIN_ERREURS` valant 10) ;
- et pour un programmeur, on ajoute un bonus de 200 francs pour chaque projet achevé ; une constante publique sera utilisée (`FACTEUR_GAIN_PROJETS` valant 200).

Ajoutez enfin le code nécessaire afin que l'exécution de votre code produise un affichage strictement analogue à celui de la méthode `main` fournie (entre `Test partie 1` : et `Test partie 2` : dans les traces d'exécution données plus bas).

Pour obtenir une représentation sous forme de chaîne de caractères avec deux chiffres après la virgule pour un `double` donné `d`, vous utiliserez la tournure suivante :

```
String.format("%.2f", d)
```

Le salaire annuel doit s'afficher avec 2 chiffres après la virgule.

**Demande de prime** Définissez ensuite dans la classe `Employe` un nouvel attribut de type `double` représentant un montant de prime obtenu par l'employé (toujours initialisé à zéro).

Modifiez le calcul du salaire annuel de base de sorte à ce que lui soit ajouté le montant de la prime.

Modifiez également la méthode permettant l'affichage d'un employé de sorte à ce que le montant de la prime s'affiche s'il n'est pas nul (voir les exemples de déroulement ci-dessous). La prime s'affichera avec 2 chiffres après la virgule.

Définissez ensuite dans la classe `Employe` une méthode `void demandePrime()` réalisant les traitements suivants :

- demander la saisie (au clavier) d'un montant de prime souhaitée par l'employé (un `double`) ;
- redemander ce montant tant que la donnée saisie est trop grande (l'employé ne peut demander plus de 2% de son salaire annuel) ou que la donnée saisie est non numérique (lancement d'une `InputMismatchException` par `nextDouble()`).

Les dialogues relatifs à l'interaction devront être strictement analogues à ceux donnés dans les exemples de déroulement fournis plus bas.

L'utilisateur n'aura droit qu'à 5 tentatives de saisie. Si après 5 tentatives le montant



de la prime n'a pu être saisi (parce qu'il a été à chaque fois ou trop élevé ou différent d'une valeur numérique), la prime de l'employé reste à zero. Sinon la prime de l'employé vaudra le montant saisi.

**Indication** : si une lecture échoue, pour faire en sorte que la prochaine lecture se passe bien il faudra «purger» le Scanner des données erronées en lui appliquant la méthode `getLine()`.

## 2.2 Exemple de déroulement

Exemple avec saisie de prime d'emblée correcte :

```
Test partie 1 :
Nous avons un nouvel employé : Serge Legrand, c'est un manager.
Nous avons un nouvel employé : Paul Lepetit, c'est un programmeur.
Nous avons un nouvel employé : Pierre Lelong, c'est un testeur.
Affichage des employés :
Serge Legrand :
    Taux d'occupation : 100%. Salaire annuel : 94472.00 francs.
    A voyagé 30 jours et apporté 4 nouveaux clients.

Paul Lepetit :
    Taux d'occupation : 75%. Salaire annuel : 58704.00 francs.
    A mené à bien 3 projets

Pierre Lelong :
    Taux d'occupation : 50%. Salaire annuel : 33976.00 francs.
    A corrigé 124 erreurs.

Test partie 2 :
Montant de la prime souhaitée par Serge Legrand ?
200
Affichage après demande de prime :
Serge Legrand :
    Taux d'occupation : 100%. Salaire annuel : 94672.00 francs, Prime : 200.00.
    A voyagé 30 jours et apporté 4 nouveaux clients.
```

Exemple avec saisie de prime correcte après quelques tentatives infructueuses :

```
Test partie 1 :
Nous avons un nouvel employé : Serge Legrand, c'est un manager.
Nous avons un nouvel employé : Paul Lepetit, c'est un programmeur.
Nous avons un nouvel employé : Pierre Lelong, c'est un testeur.
Affichage des employés :
Serge Legrand :
    Taux d'occupation : 100%. Salaire annuel : 94472.00 francs.
    A voyagé 30 jours et apporté 4 nouveaux clients.

Paul Lepetit :
    Taux d'occupation : 75%. Salaire annuel : 58704.00 francs.
    A mené à bien 3 projets
```

Pierre Lelong :  
 Taux d'occupation : 50%. Salaire annuel : 33976.00 francs.  
 A corrigé 124 erreurs.

Test partie 2 :  
 Montant de la prime souhaitée par Serge Legrand ?  
 jsdhf  
 Vous devez introduire un nombre!  
 Montant de la prime souhaitée par Serge Legrand ?  
 lsékd  
 Vous devez introduire un nombre!  
 Montant de la prime souhaitée par Serge Legrand ?  
 300000  
 Trop cher!  
 Montant de la prime souhaitée par Serge Legrand ?  
 40000  
 Trop cher!  
 Montant de la prime souhaitée par Serge Legrand ?  
 450  
 Affichage après demande de prime :  
 Serge Legrand :  
 Taux d'occupation : 100%. Salaire annuel : 94922.00 francs, Prime : 450.00.  
 A voyagé 30 jours et apporté 4 nouveaux clients.

### Exemple avec saisie de prime incorrecte pendant 5 tentatives :

Test partie 1 :  
 Nous avons un nouvel employé : Serge Legrand, c'est un manager.  
 Nous avons un nouvel employé : Paul Lepetit, c'est un programmeur.  
 Nous avons un nouvel employé : Pierre Lelong, c'est un testeur.  
 Affichage des employés :  
 Serge Legrand :  
 Taux d'occupation : 100%. Salaire annuel : 94472.00 francs.  
 A voyagé 30 jours et apporté 4 nouveaux clients.

Paul Lepetit :  
 Taux d'occupation : 75%. Salaire annuel : 58704.00 francs.  
 A mené à bien 3 projets

Pierre Lelong :  
 Taux d'occupation : 50%. Salaire annuel : 33976.00 francs.  
 A corrigé 124 erreurs.

Test partie 2 :  
 Montant de la prime souhaitée par Serge Legrand ?  
 10000000  
 Trop cher!  
 Montant de la prime souhaitée par Serge Legrand ?  
 2cent  
 Vous devez introduire un nombre!  
 Montant de la prime souhaitée par Serge Legrand ?  
 3569898  
 Trop cher!  
 Montant de la prime souhaitée par Serge Legrand ?  
 3cent  
 Vous devez introduire un nombre!  
 Montant de la prime souhaitée par Serge Legrand ?  
 40000000  
 Trop cher!  
 Affichage après demande de prime :

Serge Legrand :

Taux d'occupation : 100%. Salaire annuel : 94472.00 francs.

A voyagé 30 jours et apporté 4 nouveaux clients.

### 3 Exercice 3 — Elections

Le but de cet exercice est de simuler la désignation d'un chef par les membres de son parti.

Télécharger le programme fourni sur le site du cours<sup>3</sup> et le compléter.

**ATTENTION :** vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernant spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :

Window > Preferences > Java > Editor > Save Actions  
(et décocher l'option de reformatage si elle est cochée)

2. sauvegarder le fichier téléchargé sous le nom `Votation.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `[dossierDuProjetPourCetExercice]/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/* *****  
 * Completez le programme a partir d'ici.  
 * *****/  
  
/* *****  
 * Ne rien modifier apres cette ligne.  
 * *****/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Votation.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

---

3. <https://d396qusza40orc.cloudfront.net/intropoojava/assignments-data/Votation.java>

### 3.1 Le code à produire

Il vous est demandé de compléter le code selon la description qui suit.

**1) la classe `Postulant`** Il s'agit ici d'implémenter une classe `Postulant`, permettant de représenter les personnes se présentant au poste de chef. Un `Postulant` est caractérisé par :

- son *nom* ;
- son *nombre d'électeurs* (nombre de membres qui votent pour lui).

Il vous est demandé d'implémenter la classe `Postulant`, de sorte à ce que les contraintes suivantes soient respectées :

1. le constructeur de la classe devra initialiser le *nom* et le *nombre d'électeurs* au moyen de valeurs passées en paramètre (dans cet ordre). Le nombre d'électeurs du postulant sera à 0 par défaut. Le constructeur devra être compatible avec la méthode `main` fournie en exemple ;
2. la classe `Postulant` comportera :
  - un constructeur de copie ;
  - une méthode `elect`, compatible avec la méthode `main` fournie en exemple, permettant d'incrémenter le nombre d'électeurs du postulant ;
  - une méthode `init` permettant de remettre à zéro le nombre d'électeurs ;
  - un getter `getVotes` retournant le nombre d'électeurs ayant voté en faveur du postulant ;
  - un getter `getNom` retournant le nom du postulant ;
3. la classe doit être bien encapsulée.

**2) La classe `Scrutin`** La désignation d'un chef se fait par l'organisation d'un *scrutin*.

Un `Scrutin` est caractérisé par :

- l'ensemble des postulants se présentant au poste de chef ;
- le nombre de votants maximal (tous les membres du parti) ;
- la date (jour) du scrutin (un entier).

Il s'agit maintenant d'implémenter une classe `Scrutin` modélisant ce concept et respectant les contraintes suivantes :

1. la classe, et en particulier son constructeur, doit être compatible avec la méthode `main` fournie en exemple ; chaque postulant sera une copie de celui passé en paramètre. Le constructeur prendra un booléen comme dernier argument ; si ce booléen vaut `true`, chaque postulant verra son nombre de votants re-initialisé à zéro ; la valeur de cet argument sera `true` par défaut ;
2. la classe doit être proprement encapsulée ;
3. l'ensemble des postulants sera représenté par un `ArrayList` ;
4. la classe devra mettre à disposition les méthodes publiques suivantes :
  - une méthode `calculerVotants` retournant le nombre effectifs de votants (un entier). Il s'agit de la somme du nombre d'électeurs de tous les postulants ;
  - une méthode `gagnant` retournant le nom du postulant ayant le plus d'électeurs (un `String`). Si plusieurs postulants ont le même nombre d'électeurs, le dernier sera retenu ;
  - une méthode `resultats` affichant les résultats du scrutin selon le format suivant :
 

```
Taux de participation -> <taux de participation au scrutin> pour cent
Nombre effectif de votants -> <nombre effectif de votants>
Le chef choisi est -> <nom du chef choisi>
Répartition des electeurs
<nom du postulant 1> -> <pourcent 1> pour cent des électeurs
.....
<nom du postulant n> -> <pourcent n> pour cent des électeurs
```

(une ligne vide doit apparaître en fin d'affichage).

Le `<taux de participation au scrutin>` est le ratio, en pourcentage, entre le nombre effectif de votants et le nombre maximal de votants et vous utiliserez la méthode `gagnant` pour déterminer le `<nom du chef choisi>`.

`<pourcent i>` est le pourcentage d'électeurs en faveur du postulant `i` (ratio, en pourcentage, entre le nombre d'électeurs en sa faveur et le nombre effectif de votants).

Si le nombre effectif de votants est nul, la méthode `resultats` affichera simplement un message indiquant que le scrutin est annulé, à savoir :

```
"Scrutin annulé, pas de votants"
suivi d'un saut de ligne.
```

**Toutes les valeurs numériques seront affichées avec un seul chiffre après la virgule.**

Utilisez pour cela ce genre de tournures :

```
System.out.format("Le résultat est %.1f", valeur);  
où valeur est un double à afficher avec une décimale après la vir-  
gule.
```

Cette partie de votre programme peut être testée par la portion de la méthode `main` fournie comprise entre `// TEST1` et `// FIN TEST 1` (voir l'exemple de déroulement ci-dessous).

**3) Hiérarchie de Votes** Les membres du parti s'expriment par bulletins de vote (classe `Vote`). Un bulletin de vote est caractérisé par :

- le nom d'un *postulant* (celui qui est choisi par le bulletin en question, un `String`);
- la date effective où le bulletin a été déposé (pour simplifier, le jour uniquement, sous la forme d'un entier);
- la date limite au delà de laquelle ce bulletin ne peut être déposé (encore un jour).

La classe `Vote` doit fournir :

- une méthode de test d'invalidité : `boolean estInvalide()` qui ne peut être définie concrètement pour un bulletin de vote quelconque ;
- un constructeur initialisant le nom du postulant, la date effective du vote et la date limite au moyen de valeurs passées en paramètre (dans cet ordre) ;
- les getters `getDate` et `getDateLimite` ;
- la redéfinition de la méthode `toString` produisant une représentation du bulletin respectant strictement le format suivant :  
pour `<nom du postulant>` -> invalide si le bulletin est invalide et  
pour `<nom du postulant>` -> valide si le bulletin est valide.  
où `<nom du postulant>` est le nom du postulant en faveur de qui est le bulletin. **Notez le caractère espace qui débute la représentation du bulletin.**

Un `Vote` peut être fait par *bulletin papier* (classe `BulletinPapier`) ou *électroniquement* (classe `BulletinElectronique`). Un bulletin papier peut de plus être soumis par courrier (classe `BulletinCourrier`).

On considérera :

1. qu'un vote électronique est invalide si sa date est strictement supérieure à la date limite moins deux (ils doivent être faits avant les autres) ;
2. qu'un vote par bulletin papier est invalide si le bulletin n'est pas signé (un simple attribut booléen indiquera si c'est le cas ou pas). Cet attribut sera initialisé par un constructeur au moyen d'une valeur passée en dernier paramètre ;

3. et qu'un vote par courrier est invalide s'il n'est pas signé ou que sa date est strictement supérieure à la date limite.

Les classes dont la validité dépend d'une date sont tenues d'implémenter une interface `CheckBulletin` imposant une méthode boolean `checkDate()`. C'est par le biais de cette méthode que les tests de validité sur la base de la date seront effectués.

Les sous-classes de `Vote` offriront aussi les redéfinitions de la méthode `toString` produisant des représentations respectant les formats suivants :

- pour les bulletins papier :  
vote par bulletin papier pour <nom> -> invalide si le bulletin est invalide et  
vote par bulletin papier pour <nom> -> valide si le bulletin est valide ;
- pour les bulletins soumis par courrier :  
envoi par courrier d'un vote par bulletin papier pour <nom> -> invalide si le bulletin est invalide et  
envoi par courrier d'un vote par bulletin papier pour <nom> -> valide si le bulletin est valide ;
- et pour les bulletins électroniques :  
vote electronique pour <nom> -> invalide si le bulletin est invalide et  
vote electronique pour <nom> -> valide si le bulletin est valide.

où <nom> est le nom du postulant.

**4) Simulation d'un scrutin** Il s'agit maintenant d'enrichir la classe `Scrutin` de sorte à pouvoir tenir compte d'un ensemble de votes.

Ajoutez pour cela à votre classe `Scrutin` :

- un attribut `votes` représentant un ensemble de `Vote` (un `ArrayList`). Faites attention aux modifications éventuelles que cela peut impliquer sur le constructeur existant ;
- une méthode `compterVotes` mettant à jour le nombre d'électeurs de chaque postulant en fonction du contenu de `votes` : pour chaque vote valide de `votes`, en faveur du postulant `p`, incrémenter le nombre d'électeurs du postulant `p` ;
- une méthode `simuler` prenant en paramètre un taux de participation et un jour de vote. Cette méthode va simuler le déroulement d'un scrutin selon l'algorithme suivant :

1. calculer le nombre de votants comme étant le nombre maximal de votants multiplié par le taux de participation (et **ensuite**, convertissez la valeur obtenue en `int`);
2. pour chaque votant `i`, tirer au hasard un entier `candNum` entre 0 et le nombre de postulants (moins 1) au scrutin, au moyen de la méthode fournie `Utils.randomInt`;
3. si `i%3` retourne 0, ajouter un bulletin électronique à l'ensemble `votes` en faveur du candidat d'indice `candNum` dans le tableau de postulants du scrutin;
4. si `i%3` retourne 1, ajouter un bulletin papier à l'ensemble `votes` en faveur du candidat d'indice `candNum` dans le tableau de postulants du scrutin;
5. si `i%3` retourne 2, ajouter un bulletin courrier à l'ensemble `votes`, en faveur du candidat d'indice `candNum` dans le tableau de postulants du scrutin;
6. afficher le vote ainsi obtenu en utilisant la représentation adéquate décrite plus haut.

Pour simplifier, **tous les bulletins papiers des votants pairs seront non signés et les autres signés.**

Tous les bulletins seront datés au moyen du paramètre de la méthode `simuler`. Notez que la définition de la classe `Scrutin` a un attribut spécifiant la date du vote !

Cette partie de votre programme peut être testée par la portion de la méthode `main` fournie comprise entre `// TEST 2` et `// FIN TEST 2` (voir l'exemple de déroulement ci-dessous).

## 3.2 Exemple de déroulement

Test partie I:

-----

Taux de participation -> 36.7 pour cent

Nombre effectif de votants -> 11

Le chef choisi est -> Angel Anerckjel

Répartition des électeurs

Tarek Oxlama -> 18.2 pour cent des électeurs

Nicolai Tarcozi -> 27.3 pour cent des électeurs

Vlad Imirboutine -> 18.2 pour cent des électeurs

Angel Anerckjel -> 36.4 pour cent des électeurs



Test partie II:

-----

vote électronique pour Nicolai Tarcozi -> valide  
vote par bulletin papier pour Vlad Imirboutine -> valide  
envoi par courrier d'un vote par bulletin papier pour Vlad Imirboutine -> invalide  
vote électronique pour Tarek Oxlama -> valide  
vote par bulletin papier pour Vlad Imirboutine -> invalide  
envoi par courrier d'un vote par bulletin papier pour Nicolai Tarcozi -> valide  
vote électronique pour Angel Anerckjel -> valide  
vote par bulletin papier pour Angel Anerckjel -> valide  
envoi par courrier d'un vote par bulletin papier pour Nicolai Tarcozi -> invalide  
vote électronique pour Angel Anerckjel -> valide  
vote par bulletin papier pour Tarek Oxlama -> invalide  
envoi par courrier d'un vote par bulletin papier pour Tarek Oxlama -> valide  
vote électronique pour Tarek Oxlama -> valide  
vote par bulletin papier pour Angel Anerckjel -> valide  
envoi par courrier d'un vote par bulletin papier pour Vlad Imirboutine -> invalide  
Taux de participation -> 50.0 pour cent  
Nombre effectif de votants -> 10  
Le chef choisi est -> Angel Anerckjel

Répartition des électeurs

Tarek Oxlama -> 30.0 pour cent des électeurs  
Nicolai Tarcozi -> 20.0 pour cent des électeurs  
Vlad Imirboutine -> 10.0 pour cent des électeurs  
Angel Anerckjel -> 40.0 pour cent des électeurs

vote électronique pour Tarek Oxlama -> invalide  
vote par bulletin papier pour Vlad Imirboutine -> valide  
envoi par courrier d'un vote par bulletin papier pour Vlad Imirboutine -> invalide  
vote électronique pour Vlad Imirboutine -> invalide  
vote par bulletin papier pour Tarek Oxlama -> invalide  
envoi par courrier d'un vote par bulletin papier pour Tarek Oxlama -> valide  
vote électronique pour Tarek Oxlama -> invalide  
vote par bulletin papier pour Tarek Oxlama -> valide  
envoi par courrier d'un vote par bulletin papier pour Tarek Oxlama -> invalide  
vote électronique pour Nicolai Tarcozi -> invalide  
vote par bulletin papier pour Tarek Oxlama -> invalide  
envoi par courrier d'un vote par bulletin papier pour Vlad Imirboutine -> valide  
vote électronique pour Nicolai Tarcozi -> invalide  
vote par bulletin papier pour Vlad Imirboutine -> valide  
envoi par courrier d'un vote par bulletin papier pour Vlad Imirboutine -> invalide  
Taux de participation -> 25.0 pour cent  
Nombre effectif de votants -> 5  
Le chef choisi est -> Vlad Imirboutine

Répartition des électeurs

Tarek Oxlama -> 40.0 pour cent des électeurs

Nicolai Tarcozi -> 0.0 pour cent des électeurs  
Vlad Imirboutine -> 60.0 pour cent des électeurs  
Angel Anerckjel -> 0.0 pour cent des électeurs

vote électronique pour Nicolai Tarcozi -> invalide  
vote par bulletin papier pour Tarek Oxlama -> valide  
envoi par courrier d'un vote par bulletin papier pour Tarek Oxlama -> invalide  
vote électronique pour Tarek Oxlama -> invalide  
vote par bulletin papier pour Tarek Oxlama -> invalide  
envoi par courrier d'un vote par bulletin papier pour Angel Anerckjel -> valide  
vote électronique pour Tarek Oxlama -> invalide  
vote par bulletin papier pour Nicolai Tarcozi -> valide  
envoi par courrier d'un vote par bulletin papier pour Tarek Oxlama -> invalide  
vote électronique pour Nicolai Tarcozi -> invalide  
vote par bulletin papier pour Angel Anerckjel -> invalide  
envoi par courrier d'un vote par bulletin papier pour Nicolai Tarcozi -> valide  
vote électronique pour Angel Anerckjel -> invalide  
vote par bulletin papier pour Nicolai Tarcozi -> valide  
envoi par courrier d'un vote par bulletin papier pour Nicolai Tarcozi -> invalide  
Taux de participation -> 25.0 pour cent  
Nombre effectif de votants -> 5  
Le chef choisi est -> Nicolai Tarcozi

Répartition des électeurs  
Tarek Oxlama -> 20.0 pour cent des électeurs  
Nicolai Tarcozi -> 60.0 pour cent des électeurs  
Vlad Imirboutine -> 0.0 pour cent des électeurs  
Angel Anerckjel -> 20.0 pour cent des électeurs