Cours MOOC EPFL d'introduction à la programmation orientée objet, illustré en Java

Fifth assignment (graded) : synthesis

J. Sam & J.-C. Chappelier

This assignment consists of three exercises to submit.

1 Exercise 1 — Neurons

We are interested in this exercise to model in an (obviously) very basic way a brain consisting of a set of neurons:

- the neurons can be connected between them;
- one neuron is sensible to the reception of an external signal, which it transmits to all the connected neurons;
- some neurons can be specialized.

Download the source code available at the course webpage[1] and complete it.

WARNING: you should modify neither the beginning nor the end of the program, only add your own lines as indicated. It is therefore very important to respect the following procedure (the points 1 and 3 concern only Eclipse users):

1. remove automated formatting of the code in Eclipse:
   `Window > Preferences > Java > Editor > Save Actions` (and untick the formatting option if it's on);

[1] `https://d396qusza40orc.cloudfront.net/intropoojava/assignments-data/SimulateurNeurone.java`

2. save the downloaded file as `SimulateurNeurone.java` (respect the upper case). If you work with Eclipse, save it to `[projectFolderUsedForThatExercise]/src/`;

3. refresh the Eclipse project where the file is stored (right click on the project > "refresh") in order to take into account the new file;

4. write your code between these two provided comments:

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

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

5. save and test your program to be sure that it works properly, for example using the values given below;

6. upload the modified file (still named `SimulateurNeurone.java`) in "OUTPUT submission" (not in "Additional"!).

You are asked to complete the code according to the description in four parts that follows.

1.1 Code to produce

You are asked to complete the code according to the description that follows.

1) The `Position` class This utility class is used to model the position of a neuron (in two dimensions for simplicity).

It will be characterized by two `double` representing respectively the x and the y coordinates.

The public methods of the class `Position` will be:

- a constructor initializing the attributes using `double`s passed as arguments;
- a default constructor initializing the two coordinates with zero;
- the getters `getX()` and `getY()` ;

- a redefinition of the method `toString` generating a represenation respecting the following format:
  `(<x>, <y>)`
  where `<x>` is the x coordinate and `<y>` the y coordinate.

2)The `Neurone` class It consists here of implementing a class `Neurone`, allowing to represent a neuron of the brain.

A *neurone* is characterized by:

- a *position* (of type `Position`);
- an *internal signal* corresponding to the response to a stimulus received from the exterior (a `double`);
- an *attenuation factor* of the signal (a `double`);
- the set of neurons, `connexions` (connections), to which it is connected (an `ArrayList` of `Neurone`).

You are asked to implement the class `Neurone`, in a way that the following constraints are respected:

1. The constructor of the class should initialize the *position* and the *attenuation factor* using values passed as parameters, and the *internal signal* with 0. The constructor should be compatible with the `main` method provided as an example.
2. The `Neurone` class includes:
   - the getters `Position getPosition()`, `int getNbConnexions()`, `Neurone getConnexion( int index)`, `getAttenuation` and `getSignal()` returning respectively:
     - the position of the neuron ;
     - the number of neurons in the `connexions` ;

  - the index element `index` of the `connexions` ;
  - the attenuation factor ;
  - and the signal stored by the current instance.

- the method `void connexion(Neurone n)` that adds the neuron `n` to the sets of connections. The added neuron must be added at the end of the set ;
- a method `void recoitStimulus(double stimulus)` allowing to stimulate a neuron. This stimulation results in:
  - storing in the internal signal of the neuron the value of the stimulus multiplied by the attenuation factor;
  - propagating the internal signal to the connected neurons by invoking the method `recoitStimulus` of all the elements of `connexions`. The method `recoitStimulus` takes therefore as argument the *internal signal* of the considered neuron;
- a redefinition of the method `toString` producing a representation of the neuron respecting the following format:

```
Le neurone en position <position> avec attenuation <att> en connexion
- un neurone en position <position 0>
...
- un neurone en position <position n>
```

  where `<position>` is the representation in the form of a string of the position of the neuron, `<att>` is the attenuation factor and `<position i>` is the position of the ith element of the set `connexions` for the neurons (there are two spaces at the beginning of every line related to the set `connexions`). In the case where the neuron is not connected to any other, the produced representation will be:
  `" sans connexion\n"`

3. The class should be well encapsulated.

This part of the program can be tested using the part of the principal program given between `// TEST DE LA PARTIE 1` and `// FIN TEST DE LA PARTIE 1`.

3) The `NeuroneCumulatif` class Some neurons are specialized. They treat the received stimulus in a cumulative way:

```
internal signal  = internal signal + stimulus * attenuation
```

You now have to code a sub-class `NeuroneCumulatif` inheriting from the `Neurone` and redefining in an appropriate way the method `recoitStimulus`.

The constraints to respect are the following:

- the constructor of the sub-class will be compatible with the method `main` provided as example.
- the method `recoitStimulus` will not unnecessarily duplicate the portions of the code of the redefined `recoitStimulus` method.

This part of the program can be tested using the part of the principal program given between `// TEST DE LA PARTIE 2` and `// FIN TEST DE LA PARTIE 2`.

4) The `Cerveau` class A cerveau(brain) is a set of `Neurone` (`ArrayList`).
It will have the following public methods:

- the getters `int getNbNeurones()` and `Neurone getNeurone(int index)` returning respectively the number of neurons in the brain and the neuron in the position `index` in the table of the neurons of the brain;
- the method `void ajouterNeurone(Position pos, double attenuation)` creating a neuron using the provided parameters and adding it in the set of the neurons of the brain;
- the method `void ajouterNeuroneCumulatif(Position pos, double attenuation)` creating a cumulative neuron using the provided parameters of the method and adding it in the set of the neurons of the brain;

- the method `void stimuler(int index, double stimulus` stimulating the neuron with index `index` (méthode `recoitStimulus`);

- the method `sonder(int index)` returning the signal stored in the neuron with index `index`;

- the method void `creerConnexions()`, creating the connections between the neurons of the brain, according to the following ad hoc algorithm:

  1. the neuron with index zero is connected to the neuron with index 1 if it exists;
  2. the neuron with index zero is connected to the neuron with index 2 if it exists;
  3. for all odd indexes `i` inferior to the size of the set of neurons minus two: a connection is created between the neuron `i` and the neuron `i+1` as well as between the neuron `i+1` and the neuron `i+2`

- a redefinition of the method `toString` allowing the display of the brain according to the following format (here shown for a concrete example with 4 neurons):

```
*----------*

Le cerveau contient 4 neurone(s)
Le neurone en position (0.0, 0.0) avec attenuation 0.5 en connexion avec
- un neurone en position (0.0, 1.0)
- un neurone en position (1.0, 0.0)

Le neurone en position (0.0, 1.0) avec attenuation 0.2 en connexion avec
- un neurone en position (1.0, 0.0)

Le neurone en position (1.0, 0.0) avec attenuation 1.0 en connexion avec
- un neurone en position (1.0, 1.0)

Le neurone en position (1.0, 1.0) avec attenuation 0.8 sans connexion

*----------*
```

  There are two line breaks after every `"*----------*"`

This part of your program can be tested by the portion of the given `main` method after `// TEST DE LA PARTIE 3` (see the provided code here below).

1.2 Execution example

```
Test de la partie 1:
--------------------
Signaux :
5.0
5.0
10.0

Premiere connexion du neurone 1
Le neurone en position (1.0, 0.0) avec attenuation 1.0 en connexion avec
  - un neurone en position (1.0, 1.0)

Test de la partie 2:
--------------------
Signal du neurone cumulatif  -> 10.0

Test de la partie 3:
--------------------
Signal du 3eme neurone -> 4.8

*----------*

Le cerveau contient 4 neurone(s)
Le neurone en position (0.0, 0.0) avec attenuation 0.5 en connexion avec
  - un neurone en position (0.0, 1.0)
  - un neurone en position (1.0, 0.0)

Le neurone en position (0.0, 1.0) avec attenuation 0.2 en connexion avec
  - un neurone en position (1.0, 0.0)

Le neurone en position (1.0, 0.0) avec attenuation 1.0 en connexion avec
  - un neurone en position (1.0, 1.0)

Le neurone en position (1.0, 1.0) avec attenuation 0.8 sans connexion

*----------*
```

2 Exercise 2 — Staff management

The goal of this exercice is to manage the salaries of the employees of a software engineering compagny. An employee (class `Employe`) is characterized by his/her name (the field `nom` which must remain unchanged after its initialisation),

a monthly income and an occupation rate (the percentage of time worked monthly; for example 80%). There exist three types of employees:

- managers (class `Manager`), specifically characterized by the number of travelled days and by the number of new clients brought to the compagny;
- testers (class `Testeur`), characterized by the number of bugs they managed to solve while testing;
- programmers (class `Programmeur`), characterized by the number of projects they have completed.

Take care to make sure all your classes are well encapsulated!

Download the source code available at the course webpage[2] and complete it.

WARNING: you should modify neither the beginning nor the end of the program, only add your own lines as indicated. It is therefore very important to respect the following procedure (the points 1 and 3 concern only Eclipse users):

1. remove automated formatting of the code in Eclipse:
   `Window > Preferences > Java > Editor > Save Actions` (and untick the formatting option if it's on);
2. save the downloaded file as `Employes.java` (respect the upper case). If you work with Eclipse, save it to `[projectFolderUsedForThatExercise]/src/`;
3. refresh the Eclipse project where the file is stored (right click on the project > "refresh") in order to take into account the new file;
4. write your code between these two provided comments:

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

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

[2] `https://d396qusza40orc.cloudfront.net/intropoojava/assignments-data/Employes.java`

5. save and test your program to be sure that it works properly, for example using the values given below;

6. upload the modified file (still named `Employes.java`) in "OUTPUT submission" (not in "Additional"!).

2.1 The code to be produced

Employees and salaries Your task is to complete the provided code in accordance with the following description.

Start by equipping your classes with constructors enabling the initialisation of all their attributes. The constructors must be compatible with the provided `main` method; the occupation rate will be 100% by default. Moreover, if a constructor receives as parameter an occupation rate lower than 10%, the effective occupation rate of the employee must be set to 10%. Similarly if the occupation rate received by the constructor is greater than 100%, the effective occupation rate will be set to 100%. The constructor of an `Employe` will display the message `Nous avons un nouvel employé : <...>` (where `<...>` must correspond to what is displayed in the execution examples given below).

Add a method `double revenuAnnuel()` (yearly income) which computes and returns the yearly income of an employee as follows:

- each employee has a base yearly income computed as 12 times the monthly income multiplied by the occupation rate;
- for a manager, a bonus of 500 francs per client brought to the company is added as well as 100 francs per day for the expenditure of the travelled days; public constants will be used (`FACTEUR_GAIN_CLIENT` equal to 500 and `FACTEUR_GAIN_VOYAGE` equal to 100);
- for a tester, a bonus of 10 francs per corrected bug is added; a public constant will be used (`FACTEUR_GAIN_ERREURS` equal to 10);
- finally, for a programmer, a bonus of 200 francs per completed project will be added; a public constant will also be defined (`FACTEUR_GAIN_PROJETS` equal to 200).

Add all the necessary code so that the execution of the provided `main` produces the output given in the execution examples below (between `Test partie 1 :` and `Test partie 2 :`).

In order to produce a string reprentation of a given `double d` with 2 digits after the decimal point, you will use the following instruction :

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

The yearly income must be displayed using this format. The method `toString` of the class `Employe` must be coded in such a way that no code duplication occur in the `toString` methods of the sub-classes.

Exceptional bonuses In the class `Employe`, define a new attribute of type `double` representing the amount of exceptional bonus ("prime" in French) granted to the employee. This bonus will always be initialized to zero.

Then modify the computation of the base yearly income so that it incorporates the exceptional bonus (the bonus is simply added to the yearly income).

Modify the method enabling the display of the employee so that the bonus is displayed if non zero (see the execution examples below). The bonus must be displayed with 2 digits after the decimal point.

Finally, define a new method `void demandePrime()` in the class `Employe`. This method will:

- prompt the user for the amount of the exceptional bonus requested by the employee (a `double`);
- ask again to enter this amount as long as it is too large (the employee cannot ask for more than 2% of her/his yearly income) or it is a non numerical value (`InputMismatchException` thrown by `nextDouble()`).

The dialogs related to the interaction with the user will be strictly conform to the execution examples given below.

The user will have at most 5 opportunities to enter the amount. If after 5 attemps the bonus could not be properly entered (because each time it is either too large or non numerical), the exceptional bonus will remain zero. Otherwise, the value entered by the user is assigned to the bonus.

Indication : if reading an input fails, the `Scanner` must be cleaned from erroneous data using the method `nextLine()`. The `Scanner` must be declared as a local variable of the method `void demandePrime()`.

2.2 Execution example

Example where a correct bonus is entered straightaway:

```
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.
```

Example where a correct bonus is entered after a number of fruitless attempts:

```
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.
```

Example where the maximal number of 5 attempts is exceeded:

```
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 Exercise 3 — Elections

The goal of this exercise is to simulate the election of a leader by the members of his (or her) party.

Download the source code available at the course webpage[3] and complete it.

[3] `https://d396qusza40orc.cloudfront.net/intropoojava/assignments-data/Votation.java`

WARNING: you should modify neither the beginning nor the end of the program, only add your own lines as indicated. It is therefore very important to respect the following procedure (the points 1 and 3 concern only Eclipse users):

1. remove automated formatting of the code in Eclipse:
   `Window > Preferences > Java > Editor > Save Actions` (and untick the formatting option if it's on);
2. save the downloaded file as `Votation.java` (respect the upper case). If you work with Eclipse, save it to `[projectFolderUsedForThatExercise]/src/`;
3. refresh the Eclipse project where the file is stored (right click on the project > "refresh") in order to take into account the new file;
4. write your code between these two provided comments:

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

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

5. save and test your program to be sure that it works properly, for example using the values given below;
6. upload the modified file (still named `Votation.java`) in "OUTPUT submission" (not in "Additional"!).

3.1 The code to be produced

Your task is to complete the provided code in accordance with the following description.

1. the `Postulant` class We start by implementing a class `Postulant` (means `Applicant`) that will be used to represent each person applying for the position of leader. A `Postulant` is characterized by:

- a *name*;

- the *number of voters* (number of people that will vote for him/her).

You are required to satisfy the following constraints when implementing the `Postulant` class:

1. the constructor will initialize the applicant's *name* and the `number of voters` given values taken as parameters (in that order). The number of voters will be 0 by default. The constructor must be compatible with the provided `main` method;
2. the `Postulant` class will contain:
   - a copy constructor;
   - a method `elect`, compatible with the provided `main` method, that increments the number of voters associated to the applicant;
   - a method `init` that resets the number of voters to zero;
   - a getter `getVotes` that will return the number of voters that are backing the applicant;
   - a getter `getNom` (means `getName`) that returns the applicant's name;
3. the class must be well encapsulated.

2. The `Scrutin` class The election of a leader takes place through the organisation of a *vote* (*scrutin* in French).

A `Scrutin` is characterized by:

- the set of applicants for the position of leader;
- the total number of available voters (all members in the party);
- the date (day) of the vote (an integer).

Your task is to implement the `Scrutin` class in order to model this concept while respecting the following constraints:

1. the class, in particular its constructor, must be compatible with the provided `main` method; each applicant shall be a copy of the one taken as argument. The constructor will also take a boolean as final argument; if this boolean is `true`, each applicant shall have its associated number of voters reset to zero; the value of this argument must be `true` by default;

2. the class must be well encapsulated;

3. the set of applicants must be represented using an `ArrayList`;

4. the class must implement the following public interface:

   - a method `calculerVotants` (means `computeVoters`) that returns the actual number of voters (an integer). This value is computed as the sum of the voters associated to each applicant;

   - a method `gagnant` (means `winner`) that returns the name (a `String`) of the applicant that received the highest number of votes. In the case of a tie, the last such applicant will be returned;

   - a method `resultats` (means `results`) that will display the results of the vote in the following format (must be strictly respected):

     ```
     Taux de participation -> <participation rate of the vote> pour cent
     Nombre effectif de votants -> <actual number of voters>
     Le chef choisi est -> <name of the elected leader>
     Répartition des electeurs
     <name of applicant 1> -> <percent 1> pour cent des électeurs
     .....
     <name of applicant n> -> <percent n> pour cent des électeurs
     ```

     (an empty line must appear at the end of the display).

     The `<participation rate of the vote>` is the ratio, in percents, between the actual number of voters (as computed by the previous method) and the total number of available voters. You shall then use the method `gagnant` described above to determine the `<name of the elected leader>`.

     `<percent i>` is the percentage of voters that backed applicant number `i` (ratio, in percents, of the number of voters backing the applicant and the actual number of voters).

     If the actual number of voters is zero, the method `resultats` will simply display a message indicating that the vote is canceled, namely:
     `"Scrutin annulé, pas de votants"`
     followed by an end-of-line character.

All numerical values must be displayed with a single decimal digit. This can be achieved by using the following instructions:
`System.out.format("The result is %.1f", value);`
where `value` is a `double` that should be displayed with a single decimal digit.

This part of your program can be tested with the provided `main` method using the portion of provided code situated between `// TEST 1` and `// FIN TEST 1` (see the execution example below).

3. `Vote` hierarchy The members of a party express their choice through a ballot (the class `Vote`). A `Vote` is characterized by:

- the name of the *applicant* (the person for whom the vote is cast, a `String`);
- the actual date when the vote was cast (for simplicity, only the day, an integer);
- the limit date after which the vote can no longer be cast (again, the day).

The `Vote` class must provide:

- an invalidity testing method: `boolean estInvalide()` (means `isInvalid`) that cannot be defined concretely for an arbitrary ballot;
- a constructor that initializes the applicant's name, the actual cast date and the limit date given values taken as parameters (in that order);
- the getters `getDate` and `getDateLimite` (means `getLimitDate`);
- the redefinition of the `toString` method that produces a `String` representation of the ballot strictly respecting the following format:
  `pour <name of the applicant> -> invalide` if the ballot is invalid and
  `pour <name of the applicant> -> valide` if it is valid
  where `<name of the applicant>` is the name of the applicant for which this vote has been cast. **Please note the leading space character in the above representations**.

A ballot can consist in a *paper ballot* (the `BulletinPapier` class, where *bulletin* means *ballot*) or an `electronic vote` (the `BulletinElectronique`

class). A paper ballot can furthermore be submitted *by mail* (the `BulletinCourrier` class).

Invalidity of a ballot is determined based on its type:

1. an electronic vote is invalid if its cast date is strictly superior to the limit date minus two (electronic votes must be sent in earlier than the others);
2. a paper ballot is invalid if the ballot hasn't been signed: a boolean attribute will indicate this being or not the case. This attribute shall be initialized by a constructor taking a boolean value as the last argument (`true` means that it is signed);
3. a vote sent in by mail is invalid if it hasn't been signed or if its cast date is strictly superior to the limit date.

The classes that require a checking dates to determine validity shall implement an interface `CheckBulletin` that requires the implementation of a method `boolean checkDate()`. This method should be used whenever it is necessary to check dates to determine validity of a ballot. This method will return `true` when the date is valid.

The subclasses of `Vote` must also redefine the `toString` method in order to produce `String` representations strictly respecting the following formats:

- for the paper ballots:
  `vote par bulletin papier pour <name> -> invalide` if the ballot is invalid and
  `vote par bulletin papier pour <name> -> valide` if it is valid;
- for the votes sent in by mail:
  `envoi par courrier d'un vote par bulletin papier pour <name> -> invalide` if the ballot is invalid and
  `envoi par courrier d'un vote par bulletin papier pour <name> -> valide` if it is valid;
- and for electronic votes:
  `vote electronique pour <name> -> invalide` if the ballot is invalid and
  `vote electronique pour <name> -> valide` if it is valid.

where `<name>` is the name of the applicant for which the vote has been cast.

4. Vote simulation Your final task is to enrich the `Scrutin` class in order to handle a set of ballots.

In order to implement this feature, add to your `Scrutin` class:

- a `votes` attribute that represents a set of `Vote` instances (use an `ArrayList`). Be wary of the x potential changes this may imply for the existing constructor;

- a method `compterVotes` (means `countVotes`) that updates the number of voters for each applicant given the contents of `votes`: for each valid ballot in `votes`, increment the number of voters of the applicant for which the vote was cast.

- a method `simuler` (means `simulate`) that takes in parameter a participation rate and a vote date. This method will simulate an election based on the following algorithm:

  1. compute the actual number of voters as the total number of voters (members of the party) multiplied by the participation rate (and **then**, convert the result to an `int`);
  2. for each voter `i`, draw an integer at random `candNum` between 0 and the number of applicants (minus 1) using the provided method `Utils.randomInt`;
  3. if `i%3` returns 0, add an electronic vote to `votes` in favor of the candidate with index `candNum` in the list of applicants;
  4. if `i%3` returns 1, add a paper ballot to `votes` in favor of the candidate with index `candNum` in the list of applicants;
  5. if `i%3` returns 2, add a vote sent in by mail `votes` in favor of the candidate with index `candNum` in the list of applicants;
  6. display the vote you obtained using the correct representation as described previously.

To simplify, **all paper ballots from odd voters will be signed, and all others will not**.

All ballots will be dated using the parameter from the `simuler` method. Remember, the definition of the `Scrutin` class has an attribute for the date of the vote!

This part of your program can be tested with the provided `main` method using the portion of provided code situated between `// TEST 2` and `// FIN TEST 2` (see the execution example below).

3.2 Execution example

```
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
```