

# Programmation Orientée Objet Langage Java

## Sujets TP

### Consignes

#### Méthode de travail

- Un seul sujet sert de fil conducteur à l'ensemble des séances.
- Chaque séance de TP est précédé du ou des cours nécessaires que vous aurez **étudiés avant**.
- Pour réaliser le TP :
  - Lisez le sujet en entier avant de commencer,
  - Revoyez les points de cours correspondants au fur et à mesure,
  - Consultez la documentation en ligne du langage.
- Pour capitaliser vos connaissances, notez bien les réponses aux questions posées dans chaque sujet.
- Les TP non achevés pendant la séance doivent être retravaillés, ils serviront de base pour la séance suivante : le minimum est d'étudier la correction proposée sur le e-campus.

#### Réalisation des TP

- Les TP s'effectuent **en binôme** sous Linux avec Eclipse ou NetBeans.
- Pour réduire les temps de réponse, définissez le workspace d'Eclipse en local /var/tmp. Pensez à sauvegarder régulièrement les fichiers sur votre compte.

#### Suivi pendant TP

- Les profs s'occupent des dépannages ponctuels et vérifient l'avancement du TP.
- Ils vous aident à résoudre les problèmes rencontrés en vous renvoyant si nécessaire sur le cours étudié et la documentation en ligne.

#### Evaluation

- Les TP ne sont pas notés puisqu'ils doivent vous permettre d'acquérir des compétences.
- Cependant, une auto-évaluation de votre niveau vous sera demandée en fin de module et sera confirmée, ou pas, par les enseignants.

**Documentation à consulter :** <http://docs.oracle.com/javase/7/docs/api/>

Classe concernée : 3IRC

Date d'émission : février 2017

Émetteur : F. PERRIN, M. BREDA

Contact : [fp@cpe.fr](mailto:fp@cpe.fr)

Service : Info/Télécom

Disponible sur le e-campus



# Préambule

## Enjeux

Être capable de concevoir et développer en Java des programmes souples, extensibles et faciles à maintenir. Cela suppose de respecter les principes suivants afin de garantir une forte cohésion et un faible couplage :

- Responsabilité unique : une classe ne doit avoir qu'une seule raison de changer.
- Ouverture fermeture : une classe doit être ouverte aux extensions mais fermée aux modifications.
- Substitution de LISKOV : une méthode utilisant une référence vers une classe de base doit pouvoir référencer des objets de ses classes dérivées sans les connaître (polymorphisme).

## Moyens

L'objectif pédagogique des TP est de mettre en œuvre, dans un programme construit au fur et à mesure des séances, les différents concepts de la Programmation Orientée Objet en les illustrant en Java :

- Abstraction et encapsulation : classe, objet, constructeur, accesseur, etc.
- Héritage et polymorphisme : interfaces, classes abstraites, classes dérivées.
- Utilisation des collections en Java.
- Programmation événementielle et graphique.

Les algorithmes sous-jacents sont volontairement fort simples de manière à se concentrer uniquement sur la programmation Objet.

## Points techniques abordés

- Classe Object et Interfaces Comparable, Cloneable : package java.lang ; interface Comparator : package java.util
- Framework de collections : interfaces Collection, List, Set, Map, classes d'implémentation de ces interfaces, classe Collections : package java.util
- Programmation graphique et événementielle : packages javax.swing et java.awt

## Recommandations

- Faites générer par votre IDE (Eclipse ou NetBeans) dans chaque classe autant de méthodes que possible, y compris la fonction *main()* pour les tests unitaires (kit de survie en annexe en fin de document).
- Les tests d'intégration seront faits par la suite dans une classe spécifique.
- Consultez les documentations recommandées et les trucs et astuces en fin de chaque sujet de TP.

# Sujet

Le programme permet de simuler des élections en vue de désigner un chef parmi un certain nombre de d'hommes politiques. L'exécution de la simulation se fait d'abord avec une IHM très sommaire en mode console puis en mode graphique.

Un scrutin a lieu un jour donné. Il est caractérisé par :

- les candidats au scrutin,
- les bulletins de vote,
- le nombre de personnes pouvant voter,
- le nb de votes valides,
- la date du scrutin.

Le vote se fait à l'aide de bulletins qui peuvent être :

- sur un papier signé et déposé dans une urne ledit jour,
- envoyés par courrier (papier signé) et arrivés au plus tard le jour du scrutin,
- ou sous forme de bulletins électroniques qui doivent arriver au minimum 2 jours avant.

Un bulletin est caractérisé par :

- l'homme politique choisi,
- la date du vote,
- un booléen qui indique si le bulletin est valide (selon les types de vote en fonction de la date et de la signature).

Pour simplifier, chaque date est représentée par un **int**, une signature par un **boolean**.

Le diagramme UML de l'application est proposé en annexe en fin de document.

# TP POO-Java N° 1 – 2 à 4h

## Concepts d'Abstraction et d'Encapsulation

### Notions de Classe, Objet, Attribut et Méthode

#### Préambule

Définir les classes qui serviront à stocker et manipuler les données relatives aux candidats.

- Classe `HommePolitique` : stocke les informations relatives à un homme politique indépendamment de toute élection.
- Classe `CandidatScrutin` : stocke les informations relatives à un candidat pendant un scrutin. Cette classe permet d'ajouter des voix au candidat lorsque les bulletins de vote sont dépouillés.
- Classe `Candidat` : stocke les informations relatives à un candidat à l'issue d'un scrutin. Cette classe permet d'exploiter les résultats d'un scrutin sans possibilité de mise à jour des données.

#### Question 1 : constructeurs, accesseurs, mutateurs, toString()

1. Créez une classe `HommePolitique` (dans un fichier `HommePolitique.java`)
  - a) ses attributs sont :
    - La civilité dont les valeurs sont FEMME ou HOMME (l'enum `Civilite` est donnée sur le e-campus dans le répertoire "Fichiers pour TP"),
    - Un nom + prénom (String),
    - Un nom de parti (String).
  - b) Munissez-la d'un constructeur paramétré de manière à initialiser les 3 attributs.
  - c) Dans la fonction `main`, ajoutez les instructions suivantes et testez (il ne se passera rien mais il ne devrait pas y avoir d'erreur ☺) :
 

```
HommePolitique h1, h2, h3;
h1 = new HommePolitique(Civilite.FEMME, "nom1", "parti1");
h3 = new HommePolitique(Civilite.HOMME, "nom3", "parti3");
```
  - d) Surchargez la méthode `toString()` de la classe `HommePolitique` de manière à ce que l'instruction `System.out.println("h1: " + h1);` ait pour résultat :
 

```
[civilité = FEMME, nom = nom1, parti = parti1]
```
2. Ajoutez à la classe `HommePolitique` les accesseurs et mutateurs permettant de manipuler les attributs.
  - a) ajoutez les instructions suivantes dans le programme de test.
 

```
System.out.println("Est ce que civilite est correct?" + h1.getCivilite());
h1.setCivilite(Civilite.HOMME);
System.out.println("Est ce que civilite a changé? " + h1.getCivilite());
```
  - b) Idem pour tester le nom-prénom et le parti.
  - c) Essayez d'affecter la civilité `ENFANT` à l'homme politique référencé par `h1`. Pourquoi est-ce impossible ?

## Question 2 : égalité de références /égalité sémantique, comparaison de 2 objets.

### 1. Egalité de références

- Vérifiez que `h1 == h3` retourne la valeur `false`. Pourquoi ?
- Ajoutez l'instruction `h1 = h3`; et vérifiez que `h1 == h3` retourne la valeur `true`. Pourquoi ?
- Vérifiez que `h1.equals(h3)` retourne la valeur `true`. Pourquoi ?
- Changez la civilité de l'homme politique référencé par `h1`. Réaffichez les valeurs de `h1` et `h3`. Que constatez-vous, pourquoi ?

### 2. Egalité sémantique :

- Ajoutez l'instruction  

```
h2 = new HommePolitique(h1.getCivilite(), h1.getNom(), h1.getParti());
```
- Vérifiez que `h1 == h2` retourne la valeur `false`. Pourquoi ?
- Vérifiez que `h1.equals(h2)` retourne la valeur `false`. Pourquoi ?
- Redéfinissez la méthode `equals()` de manière à ce que 2 objets soit égaux si **tous** leurs attributs le sont. Mettez-la en commentaires et faites générer par votre IDE dans la classe `HommePolitique` les méthodes `hashCode()` et `equals()`:
  - En quoi la méthode générée est-elle différente de celle que vous avez écrite ?
  - Pourquoi votre IDE a-t-il inséré l'annotation `@Override` devant les méthodes ?
  - A quoi sert le test `if (this == obj) ?`
  - A quoi sert le test `if (getClass() != obj.getClass()) ?`
  - A quoi sert l'instruction `HommePolitique other = (HommePolitique) obj ?`
  - Pourquoi fallait-il aussi redéfinir la méthode `hashCode()` ?
- Vérifiez maintenant que `h1.equals(h2)` retourne la valeur `true`. Pourquoi ?
- Changez la civilité de l'homme politique référencé par `h1`. Vérifiez que `h1.equals(h2)` retourne la valeur `false`. Pourquoi ?

### 3. Comparaison de 2 objets : Ajouter à la classe `HommePolitique` une méthode de comparaison de 2 instances de la classe `HommePolitique` selon l'ordre naturel :

- Modifiez la signature de votre classe :  

```
public class HommePolitique implements Comparable<HommePolitique>
```
- Implémenter la méthode `compareTo()` de l'interface `Comparable`, en comparant l'ensemble des attributs (inspirez-vous de la méthode `hashCode()`).
- Vérifiez que `h1.compareTo(h2)` retourne une valeur différente de `0`.
- Modifiez les valeurs des objets référencés par `h1` ou `h2` de manière à ce que `h1.equals(h2)` retourne la valeur `true`.
- Vérifiez que `h1.compareTo(h2)` retourne `0`.

### Question 3 : composition et encapsulation

1. **Si votre progression est lente, ne pas traiter cette question et passer à la suivante.**

Ajouter dans la classe `HommePolitique` une méthode `clone()` :

- a) Il s'agit d'implémenter la méthode `clone()` de l'interface `Cloneable`. Transformez la signature de votre classe ainsi :

```
public class HommePolitique implements Comparable<HommePolitique> Cloneable
```

```
    Testez avec h2 = (HommePolitique) h1.clone();
```

- b) Pourquoi faut-il transtyper le résultat ?
- c) Vérifiez que `h1 == h2` retourne la valeur `false`. Pourquoi ?
- d) Vérifiez que `h1.equals(h2)` retourne la valeur `true`. Pourquoi ?
- e) Vérifiez que `h1.compareTo(h2)` retourne 0.
- f) Modifiez le parti de `h1`. Que constatez-vous dans `h2`? Pourquoi?

2. Créez une classe `CandidatScrutin` (dans un fichier `CandidatScrutin.java`) qui permettra d'incrémenter le nombre de voix d'un `HommePolitique` lorsqu'un vote sera en sa faveur lors d'un scrutin donné. La future classe `Scrutin` manipulera des instances de cette classe.

- a) Ses attributs sont :
- o Un `HommePolitique`,
  - o Un nombre de voix,
  - o Une date de scrutin (stockée dans un `int` pour simplifier).
- b) Munissez-la d'une méthode `toString()`.
- c) Ajoutez un constructeur paramétré avec un `HommePolitique` et une date de scrutin.  
Ce constructeur :
- o Initialise le nb de voix à 0 ;
  - o Initialise la date du scrutin avec celle passée en paramètre.
  - o Crée (ou Clone?) un **nouvel** `HommePolitique` à partir de celui passé en paramètre.
- d) Testez.

3. Dans la classe `CandidatScrutin` faites générer et/ou ajoutez les méthodes suivantes et testez au fur et à mesure :

- o `hashCode()` et `equals()` de manière à ce que 2 objets soit égaux si tous leurs attributs sont égaux.
- o Des méthodes `get` sur les attributs nb de voix et date de scrutin.
- o Des méthodes `get` sur les attributs de l'objet `HommePolitique` enveloppé (civilité, etc.)
- o Une méthode pour incrémenter le nb de voix.
- o Une méthode pour vérifier si les attributs de l'objet `HommePolitique` enveloppé (civilité, etc.) sont les mêmes que celle d'un `HommePolitique` passé en paramètre. Cette méthode sera utilisée dans la classe `Scrutin` pour identifier le candidat référencé sur un bulletin de vote.
- o Une méthode pour comparer 2 instances de `CandidatScrutin` selon l'ordre naturel en ne tenant compte que de l'attribut `HommePolitique`.

- a) Pourquoi ne faut-il pas créer de méthode `set` sur le nb de voix et la date de scrutin ?
- b) Créer une méthode `get` publique sur l'attribut `HommePolitique` nous aurait évité de créer des méthodes `get` pour tous ses attributs. Pourquoi n'est-ce pas une bonne idée ?
- c) Pourquoi faut-t-il cloner l'`HommePolitique` passé en paramètre et non pas écrire le constructeur simplement ainsi ?

```
public CandidatScrutin(HommePolitique hommePolitique, int dateScrutin) {
    this.hommePolitique = hommePolitique;
}
```

- d. Quels auraient été les avantages et les inconvénients si l'on avait défini la date de scrutin comme un attribut de classe et non pas un attribut d'instance ?
4. Créez une classe `Candidat` (dans un fichier `Candidat.java`) qui sera une simple vue d'un `CandidatScrutin`. Des instances de cette classe seront créées lorsque l'on dépouillera un scrutin et seront manipulées par toutes les classes qui manipuleront les résultats sans possibilité de les mettre à jour. Ses attributs sont :
- o Un `CandidatScrutin`,
  - o Un pourcentage de voix (double).
- a) Munissez la d'une méthode `toString()` et d'un constructeur paramétré avec un `CandidatScrutin` et un nb de votes valides. Ce constructeur :
- o Initialise le `CandidatScrutin` avec celui passé en paramètre,
  - o Initialise le pourcentage de voix à partir du nb de voix du `CandidatScrutin` enveloppé et du nb de votes valides passé en paramètre ;
- b) Testez.
5. Dans la classe `Candidat` faites générer et/ou ajoutez les méthodes suivantes et testez au fur et à mesure :
- o `hashCode()` et `equals()` de manière à ce que 2 objets soit égaux si tous leurs attributs sont égaux.
  - o Une méthode `get` sur l'attribut pourcentage de voix.
  - o Des méthodes `get` sur les attributs de l'objet `CandidatScrutin` enveloppé (civilité, etc.)
  - o Une méthode pour comparer 2 instances de `Candidat` selon l'ordre naturel en ne tenant compte que de l'attribut `CandidatScrutin`.
- a) Pourquoi ne faut-il pas créer de méthode `set` dans cette classe ?
- b) Pourquoi est-il inutile de cloner le `CandidatScrutin` dans le constructeur dans cette classe contrairement à ce que l'on avait fait dans la classe `CandidatScrutin` ?

## Annexe

```
public enum Civilite {
    HOMME, FEMME
}
```

## Trucs et astuces

- Affichage d'un réel avec 2 décimales :  
`System.out.format("Mon réel présenté avec 2 décimales : %2.1f ", monReel);`  
ou  
`String str = String.format(" -> %2.1f", monReel);`  
`System.out.print("Mon réel présenté avec 2 décimales : %2.1f " + str);`

## Documentation à consulter (autre poly et/ou annexes)

### Doc de référence :

- <http://docs.oracle.com/javase/7/docs/api/>
  - o Interfaces `java.lang.Comparable` (méthode `compareTo()`), `java.lang.Cloneable` (`clone()`).
  - o Classe `Object`.

Tutoriaux : <http://docs.oracle.com/javase/tutorial/java/index.html>



# TP POO-Java N° 2 - 4h

## Concepts d'Héritage et de Polymorphisme

### Notions de Classe Abstraite, Classe Dérivée, Interface

#### Préambule

Il s'agit de définir les classes permettant de simuler une élection. La classe de simulation `Election` est elle-même fournie sur le e-campus ainsi que les solutions proposées pour les classes `HommePolitique`, `CandidatScrutin` et `Candidat` (*package* `tp1`).

- La classe `Scrutin` stocke les informations relatives à un scrutin :
  - les candidats au scrutin,
  - les bulletins de vote,
  - le nombre de personnes pouvant voter,
  - le nb de votes valides,
  - la date du scrutin.
- Toutes les classes implémentent l'interface `Vote` pour stocker les différents types de bulletins.

```
public interface Vote {
    public boolean estInvalide();
    public HommePolitique getHommePolitique() ;
    public int getDate();
}
```

- Le vote se fait à l'aide de bulletins qui peuvent être :
  - sur un papier signé et déposé dans une urne ledit jour (classe `BulletinPapier`),
  - envoyés par courrier (papier signé) et arrivés au plus tard le jour du scrutin (classe `BulletinCourrier`),
  - ou sous forme de bulletins électroniques qui doivent arriver au minimum 2 jours avant (classe `BulletinElectronique`).
- Les classes dont la validité dépend d'une date doivent implémenter l'interface `CheckDateBulletin` :

```
public interface CheckDateBulletin {
    public boolean checkDate(int DateScrutin);
}
```

- Celles dont la validité dépend d'une signature doivent implémenter l'interface `CheckSigneBulletin`:

```
public interface CheckSigneBulletin {
    public boolean checkSigne();
}
```

Un bulletin est caractérisé par :

- l'homme politique choisi,
- la date du vote,
- la date du scrutin (pour vérifier la validité du bulletin).

#### Recommandations

- Faites générer par votre IDE (Eclipse ou NetBeans) dans chaque classe autant de méthodes que possible, y compris la fonction `main()` pour les tests unitaires.
- Consultez les documentations recommandées et les trucs et astuces en fin de sujet.

## Question 1 : hiérarchie de classes qui implémentent des comportements différents

1. Etudiez (et comprenez) le code de la classe `Election` qui vous servira à tester la simulation (CF. Trucs et Astuces sur l'utilisation des listes). Cela vous permettra de savoir comment déclarer les méthodes des bulletins de Vote.
2. Définissez sur papier la hiérarchie de classes permettant de modéliser les différents bulletins de vote (Cf. préambule). Pensez factorisation (classe abstraite) et réfléchissez bien au niveau hiérarchique dans lequel il est opportun de déclarer et/ou définir les méthodes. Faites valider votre modèle avant de coder.

3. Codez vos différentes interfaces et classes :

- o Ecrivez une méthode `toString()` qui indique le type de vote, l'HommePolitique concerné et si le vote est valide ou non.

Exemple (Cf. Trucs et Astuces) :

```
Vote par BulletinCourrier pour [civilité = HOMME, nom = Vlad Imirboutine, parti = parti3]
-> invalide
```

```
Vote par BulletinElectronique pour [civilité = HOMME, nom = Tarek Oxlama, parti = parti1]
-> invalide
```

```
Vote par BulletinPapier pour [civilité = HOMME, nom = Tarek Oxlama, parti = parti1] ->
valide
```

- o Prévoyez un tout petit programme de test pour créer des instances des classes concrètes et testez.
- o Répondez aux questions suivantes :
  - a) Faut-il créer des mutateurs (méthodes `setXxx()`) sur les attributs privés des bulletins de vote ?
  - b) Quelle méthode ne peut pas être définie dans la classe de plus haut niveau d'abstraction ?
  - c) Faut il redéfinir la méthode `toString()` dans les classes dérivées ?

## Question 2 : réaliser une application utilisant toutes les classes créées précédemment.

1. Etudiez (et comprenez) le code de la classe `Election` qui vous servira à tester la simulation (CF. Trucs et Astuces sur l'utilisation des listes).
2. Créez une classe `Scrutin` (dans un fichier `Scrutin.java`) et munissez-la des attributs suivants :
  - les candidats au scrutin (une liste de `CandidatScrutin`, classe définie au TP1, implémentée dans une `LinkedList`) :
 

```
private List<CandidatScrutin> candidatScrutins ;
```
  - les bulletins de vote (une liste de `Vote` implémentée dans une `LinkedList`) :
 

```
private List<Vote> votes;
```
  - le nombre de personnes pouvant voter,
  - le nb de votes valides,
  - la date du scrutin.
3. Faites générer et/ou ajoutez les méthodes indiquées en prenant les précautions suivantes :
  - Ecrivez d'abord le squelette de toutes les méthodes pour garantir une bonne compilation.
  - Codez chaque méthode et testez au fur et à mesure en utilisant la classe `Election` que vous commenterez et dé-commenterez au fur et à mesure.
  - o Deux constructeurs :
    - ```
public Scrutin(int population, int dateScrutin) { ... }
```

- un autre constructeur qui initialise la liste de CandidatScrutin à partir d'une liste de HommePolitique passée en paramètre :  

```
public Scrutin(List<HommePolitique> hommesPolitiques,int population,
int dateScrutin) { ... }
```

La méthode **de création et d'ajout** de CandidatScrutin à partir de la liste de HommePolitique passée en paramètre vous est donnée dans les trucs et astuces.

Ces 2 constructeurs **initialisent** les listes de candidats et de bulletins comme suit :

```
this.votes = new LinkedList<Vote>();
this.candidatScrutins = new LinkedList<CandidatScrutin>();
```

- Une méthode addCandidat() qui permet l'ajout d'un HommePolitique à la liste de candidats au scrutin.
- Une méthode addBulletin() qui permet l'ajout d'un bulletin de vote en faveur d'un homme politique. Cette méthode se contente d'ajouter le Vote à la liste des Vote sans se soucier de vérifier la validité du bulletin.
- Une méthode toString() qui retourne toutes les infos relatives au scrutin et, pour chaque candidat, le nb de voix obtenues.
- Les accesseurs et mutateurs nécessaires et pertinents.
- Une méthode countTheVotes () qui parcourt la liste des Vote. Pour chaque bulletin, elle vérifie sa validité et si oui, incrémente le nb de voix du candidat et le nb de votes valides.
- Une méthode tauxParticipation () qui retourne le taux de participation en fonction du nb de votes valides et de la population.
- Quels accesseurs et mutateurs faut-il éviter de créer ? Pourquoi ?

4. Pour aller plus loin : pourquoi la liste des HommePolitique est-elle implémentée dans une ArrayList dans la classe Election alors que la liste des CandidatScrutin est implémentée dans une LinkedList dans la classe Scrutin ?

## Trucs et astuces

- Affichage du nom de la classe :

```
System.out.print(monObjet.getClass().getSimpleName());
```

- Affichage d'un réel avec 2 décimales :

```
System.out.format("Mon réel présenté avec 2 décimales : %2.1f ", monReel);
ou
String str = String.format(" -> %2.1f", monReel);
System.out.print("Mon réel présenté avec 2 décimales : %2.1f " + str);
```

- Utilisation des méthodes de l'interface List :

- Déclaration :

```
List< HommePolitique> hommePolitiques;
```

- Implémentation dans un tableau (création) :

```
hommePolitiques = new ArrayList< HommePolitique>();
```

- Ajout d'un élément dans la liste :

```
hommePolitiques.add(new HommePolitique(Civilite.HOMME, "Tarek Oxlama",
  "partil"));
```

- Accès au ième élément de la liste :

```
hommePolitiques.get(i) ;
```

- Taille de la liste :

```
hommePolitiques.size() ;
```

- Fonction d'initialisation de la liste des CandidatScrutin à partir de la liste des HommePolitique :

```
private void initListCandidatScrutins(List<HommePolitique> hommesPolitiques) {
    if (hommesPolitiques != null){
        for (HommePolitique hommePolitique : hommesPolitiques ) {
            CandidatScrutin candidatScrutin = null;
            candidatScrutin =
                new CandidatScrutin(hommePolitique, this.getDateScrutin());
            this.candidatScrutins.add(candidatScrutin);
        }
    }
}
```

## Documentation à consulter (autre poly et/ou annexes)

### Doc de référence :

- <http://docs.oracle.com/javase/7/docs/api/>

### Tutoriaux :

- <http://docs.oracle.com/javase/tutorial/java/index.html>
- <https://docs.oracle.com/javase/tutorial/collections/implementations/list.html>

# TP POO-Java N° 3 – 8h

## Notions de Collection (List et Set) et de Map

### Découverte

#### Préambule

Ce sujet de TP est à destination des débutants.

Il s'agit dans ce TP d'enrichir la classe `Scrutin` d'une méthode qui retourne le résultat de l'élection et de manipuler ce résultat dans la classe `Election`, toujours en mode console.

#### Recommandations

- Codez et testez au fur et à mesure les instructions ou fonctions.
- Pour les questions 1 à 3, sauf précision contraire, les réponses tiennent en 1 instruction (hors déclaration des variables et affichage du résultat). Il s'agit la plupart du temps d'identifier la bonne fonction existant dans une des classes du package `java.util`.
- Consultez les documentations recommandées et les trucs et astuces en fin de sujet.
- Repartez des corrections des TP1&2 disponibles sur le e-campus.

#### Question 0 : travail préparatoire

1. Dans la classe `Scrutin` (celle de la correction du TP2) ajoutez une méthode qui retourne le résultat du scrutin sous la forme d'une liste de `Candidat` implémentée dans une `ArrayList`. Quel est l'intérêt de retourner une liste de `Candidat` plutôt que directement la liste de `CandidatScrutin` ?
2. En vue de trier le résultat selon l'ordre des pourcentages de voix obtenues par les candidats, créez une nouvelle classe pour y définir un comparateur de `Candidat` selon l'ordre décroissant de pourcentage (Cf. documentation à consulter).
3. Dans la classe `Election` (celle de la correction du TP2), relancez une simulation et récupérez le résultat dans une `List<Candidat>`. prévoyez une fonction d'affichage d'une telle liste, elle sera appelée plusieurs fois.

#### Question 1 : utilisation des fonctions de la classe Collections

- 1.0 Dans la classe `Election`, récupérez le résultat sous forme de liste de `Candidat`.
- 1.1 Affichez le résultat du scrutin par ordre d'insertion dans la liste.
- 1.2 Affichez le résultat du scrutin trié (fonction `sort()`) selon l'ordre naturel (par ordre alphabétique tel que prévu dans la méthode `compareTo()` de la classe `Candidat`).
- 1.3 Affichez le résultat du scrutin trié par ordre décroissant de pourcentage en utilisant le comparateur précédemment créé.

1.4 Affichez le résultat du scrutin trié par ordre croissant de pourcentage de 2 manières différentes.  
Attention, il ne faut pas redéfinir explicitement un nouveau comparateur.

1.5 Affichez le candidat qui a obtenu le meilleur résultat de 2 manières différentes.

## Question 2 : utilisation des méthodes de l'interface List

2.0 Dans la classe `Election`, récupérez le résultat sous forme de liste de `Candidat`.

2.1 Affichez l'élément qui se trouve à la 2ème position.

2.2 Affichez la position du candidat qui a obtenu le meilleur résultat.

2.3 Clonez la liste de `Candidat` en une nouvelle liste.

2.4 Supprimer de la liste clonée tous les candidats dont le score est inférieur à 20 % en utilisant une boucle `for-each` puis en utilisant un itérateur (clonez à nouveau la liste entre les 2 opérations©) – plusieurs lignes pour chaque méthode.

2.5 Ne conservez de la liste initiale que les éléments communs à la liste clonée.

2.6 Vérifiez que les 2 listes contiennent les mêmes éléments de 2 manières différentes.

2.7 Supprimez tous les éléments de la liste clonée en 1 instruction.

2.8 Vérifiez que la liste est bien vide de 2 façons différentes.

## Question 3 : utilisation des méthodes des interfaces Map et Set

3.0 Dans la classe `Election`, récupérez le résultat sous forme de liste de `Candidat`.

3.1 A partir des données issues de la liste des candidats, écrivez une fonction permettant d'alimenter une `Map` dont les clés seront les 2 civilités et les valeurs une liste de noms de candidats ayant la bonne civilité (`Map<Civilite, List<String>> map ;`). Cette `Map` sera implémentée par un `TreeMap`. Plusieurs lignes sont nécessaires pour cette fonction.

3.2 Affichez le contenu de la `Map` en utilisant la méthode `toString()` de la classe `TreeMap`.

3.3 Récupérez une vue de la `Map` dans un `Set` et affichez le contenu du `Set` en utilisant la méthode `toString()` de la classe `TreeSet`.

3.4 Ecrire une fonction pour afficher le contenu de la `Map` de manière plus stylisée (retour ligne, espaces, etc.). Plusieurs lignes sont nécessaires pour cette fonction.

3.5 Affichez le nombre d'hommes.

3.6 Créez un `Set` à partir des clés de la `Map` et affichez-le.

3.7 Supprimez les femmes de la `Map` et réaffichez la `Map`.

- 3.8 Vérifiez que le Set a bien été modifié. Pourquoi ?
- 3.9 Vérifiez que le Set crée à partir des clés a lui aussi été modifié. Pourquoi ?
- 3.10 Créez un nouveau Set à partir de celui des clés qui ne soit pas une vue de la Map. Implémentez le dans un TreeSet.
- 3.11 Vérifiez la possibilité d'ajouter une civilité inexistante et l'impossibilité d'ajouter une civilité existante. Pourquoi ?
- 3.12 Créez une vue inversée du Set et l'afficher.
- 3.13 Supprimez un élément sur le Set inversé et vérifiez que le Set initial est bien aussi modifié. Pourquoi ?

## Documentation à consulter (autre poly et/ou annexes)

### Doc de référence :

- <http://docs.oracle.com/javase/7/docs/api/>
  - Interfaces Collection, List, Set, SortedSet, Map, Comparator dans package java.util ;
  - Classes AbstractCollection, AbstractSequentialList, ArrayList, LinkedList, TreeSet, Collections, HashMap, TreeMap dans package java.util ;

### Tutoriaux :

- <http://docs.oracle.com/javase/tutorial/java/index.html>
- <http://docs.oracle.com/javase/tutorial/collections/algorithms/index.html> : tri et comparateur.
- <http://www.java2s.com/Tutorial/Java/CatalogJava.htm> chap 9.40 : comparateur avec exemples simples.
- [http://roger.neel.free.fr/langages/cours\\_htm/coursjava/collection.html](http://roger.neel.free.fr/langages/cours_htm/coursjava/collection.html) : cours un peu ancien mais d'approche simple pour la réalisation du TP.

# TP POO-Java N° 3 bis – 8h

## Notions de Collection (List et Set) et de Map

### Exercices

#### Préambule

Ce sujet de TP est à destination des élèves plus avancés.

Il s'agit dans ce TP d'enrichir la classe `Scrutin` d'une méthode qui retourne le résultat de l'élection et de manipuler ce résultat dans la classe `Election`, toujours en mode console.

#### Recommandations

- Codez et testez au fur et à mesure les instructions ou fonctions.
- Consultez les documentations recommandées et les trucs et astuces en fin de sujet.
- Repartez des corrections des TP1&2 disponibles sur le e-campus.

#### Question 0 : travail préparatoire

1. Dans la classe `Scrutin` (celle de la correction du TP2) ajoutez une méthode qui retourne le résultat du scrutin sous la forme d'une liste de `Candidat` implémentée dans une `ArrayList`. Quel est l'intérêt de retourner une liste de `Candidat` plutôt que directement la liste de `CandidatScrutin` ?
2. Dans la classe `Election` (celle de la correction du TP2), écrivez une fonction qui :
  - lance une simulation
  - compte les votes
  - récupère le résultat dans une `List<Candidat>`.
  - Affiche le résultat (prévoyez une fonction d'affichage d'une telle liste, elle sera appelée plusieurs fois).
3. Dans la classe `ElectionLauncher` (celle du package `fichierPourTp.TP3`) invoquez cette méthode (supprimez la fonction `main()` de la classe `Election`). Tous vos tests se feront désormais dans cette classe.

#### Question 1 : afficher le résultat selon différents critères de tri

- 1.1 En vue de trier le résultat selon l'ordre des pourcentages de voix obtenues par les candidats, créez une nouvelle classe pour y définir un comparateur de `Candidat` selon l'ordre décroissant de pourcentage (Cf. documentation à consulter).
- 1.2 Dans la classe `Election`, écrivez une fonction qui selon le type d'ordre passé en paramètre (alphabétique ou pourcentage) retournera le résultat (issu de l'invocation de la méthode définie à la question 0.2) trié selon le bon critère.
- 1.3 Affichez le résultat (dans fonction `main()`).



## Question 2 : créer une Map associant des hommes politiques avec une image

2.1 Dans la classe `Election` écrivez une fonction qui crée cette Map implémentée dans une `TreeMap` à partir de 2 List : 1 contenant des `HommePolitique`, l'autre des noms de fichiers d'images (à récupérer dans les fichierPourTp par exemple).

- Vous pouvez vous inspirer de la construction de la liste d' `HommePolitique` de la classe `Election`.
- Attention au chemin pour accéder aux images.

2.2 Affichez cette map en utilisant la méthode `toString()` de la classe `TreeMap` (dans `main()`).

2.3 Dans la classe `Election` écrivez une fonction d'affichage de la map qui affiche 1 ligne par entrée dans la map. Pour mémoire, une Map ne peut pas être parcourue (n'implémente pas l'interface `Iterable`) mais est capable de produire une `Collection` qui elle peut l'être...

2.4 Affichez cette map en utilisant la méthode précédente (dans `main()`).

## Question 3 : créer une Map associant des candidats avec l'image de l'homme politique correspondant

3.1 Dans la classe `Election` écrivez une fonction qui crée cette Map implémentée dans une `HashMap` à partir de la liste des candidats (question 0.2) et de la map `HommePolitique-Image` précédemment créée (question 2.1).

3.2 Affichez cette map (dans `main()`) en utilisant la méthode définie précédemment ((question 2.3).

## Question 4 : créer une Map associant pour chaque parti le pourcentage de voix obtenues

4.1 Dans la classe `Election` écrivez une fonction qui crée cette Map implémentée dans une `TreeMap` à partir de la liste des candidats (question 0.2).

4.2 Affichez cette map (dans `main()`) en utilisant la méthode définie précédemment ((question 2.3).

## Question 5 : créer une Map associant pour chaque parti la liste des candidats du parti

5.1 Dans la classe `Election` écrivez une fonction qui crée cette Map implémentée dans une `TreeMap` à partir de la liste des candidats (question 0.2).

5.2 Affichez cette map (dans `main()`) en utilisant la méthode définie précédemment ((question 2.3).

## Question 6 : jouer avec les méthodes des set et des map

5.1 Dans la classe `Election`, à partir de la liste des candidats (question 0.2), écrivez une fonction qui crée une Map implémentée dans une `TreeMap` dont les clés seront les 2 civilités et les valeurs une liste de noms de candidats ayant la bonne civilité.

5.2 Affichez cette map (dans `main()`) en utilisant la méthode définie précédemment ((question 2.3)).

5.3 Dans la classe `Election` créez une fonction de jeu sur les méthodes des `Set` et des `Map` et testez-la au fur et à mesure (dans `main()`).

- a. Affichez le nombre d'hommes (dans `main()`).
- b. Créez un `Set` à partir des clés de la `Map` et affichez-le (dans `main()`).
- c. Supprimez les femmes de la `Map` et réaffichez la `Map`.
- d. Vérifiez que le `Set` a bien été modifié. Pourquoi ?
- e. Vérifiez que le `Set` crée à partir des clés a lui aussi été modifié. Pourquoi ?
- f. Créez un nouveau `Set` à partir de celui des clés qui ne soit pas une vue de la `Map`. Implémentez le dans un `TreeSet`.
- g. Vérifiez la possibilité d'ajouter une civilité inexistante et l'impossibilité d'ajouter une civilité existante. Pourquoi ?
- h. Créez une vue inversée du `Set` et l'afficher.
- i. Supprimez un élément sur le `Set` inversé et vérifiez que le `Set` initial est bien aussi modifié. Pourquoi ?

## Documentation à consulter (autre poly et/ou annexes)

### Doc de référence :

- <http://docs.oracle.com/javase/7/docs/api/>
  - Interfaces `Collection`, `List`, `Set`, `SortedSet`, `Map`, `Comparator` dans package `java.util` ;
  - Classes `AbstractCollection`, `AbstractSequentialList`, `ArrayList`, `LinkedList`, `TreeSet`, `Collections`, `HashMap`, `TreeMap` dans package `java.util` ;

### Tutoriaux :

- <http://docs.oracle.com/javase/tutorial/java/index.html>
- <http://docs.oracle.com/javase/tutorial/collections/algorithms/index.html> : tri et comparateur.
- <http://www.java2s.com/Tutorial/Java/CatalogJava.htm> chap 9.40 : comparateur avec exemples simples.
- [http://roger.neel.free.fr/langages/cours\\_html/coursjava/collection.html](http://roger.neel.free.fr/langages/cours_html/coursjava/collection.html) : cours un peu ancien mais d'approche simple pour la réalisation du TP.

# TP POO-Java N° 4 - 8h

## Concept de délégation et d'écouteurs (Observer)

### Notions de Composants graphiques, évènements, librairies

#### Préambule

Il s'agit dans ce dernier TP de réaliser une IHM graphique pour gérer le résultat d'une élection et éventuellement de gérer quelques préférences d'affichage (couleur, polices, ordre d'affichage, etc.).

Une classe `Election` est fournie sur le e-campus. Elle gère le traitement des données, charge à votre classe d'IHM (`ElectionGUI` à compléter) d'en assurer le rendu (respect du principe de Responsabilité Unique).

#### Recommandations

- Etudiez bien contenu de la classe `Election` fournie avant de l'utiliser. Les questions 1&2 ne l'utilisent pas.
- Vous pourrez travailler sur vos propres images (ne perdez cependant pas de temps à en chercher) ou utilisez celles fournies sur le e-campus.
- Consultez au fur et à mesure les docs recommandées, les trucs et astuces et l'annexe en fin de sujet.

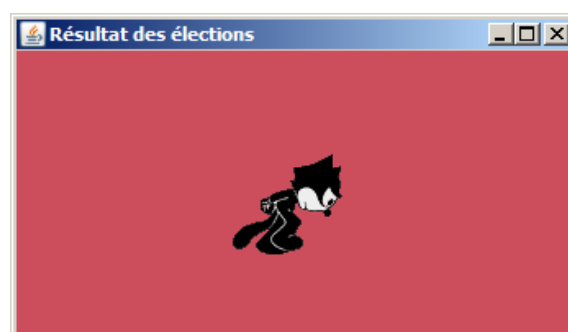
#### Question 1 : premiers composants graphiques

1. Etudiez le code du lanceur de l'application `ElectionLauncher`. Modifiez éventuellement le chemin d'accès à vos images.
2. Munissez la classe `ElectionGUI` d'un constructeur qui permet d'initialiser les attributs à partir des paramètres. Faites pour cela une fonction d'initialisation `private` qui sera appelée par le constructeur (elle aura vocation à être enrichie par la suite).

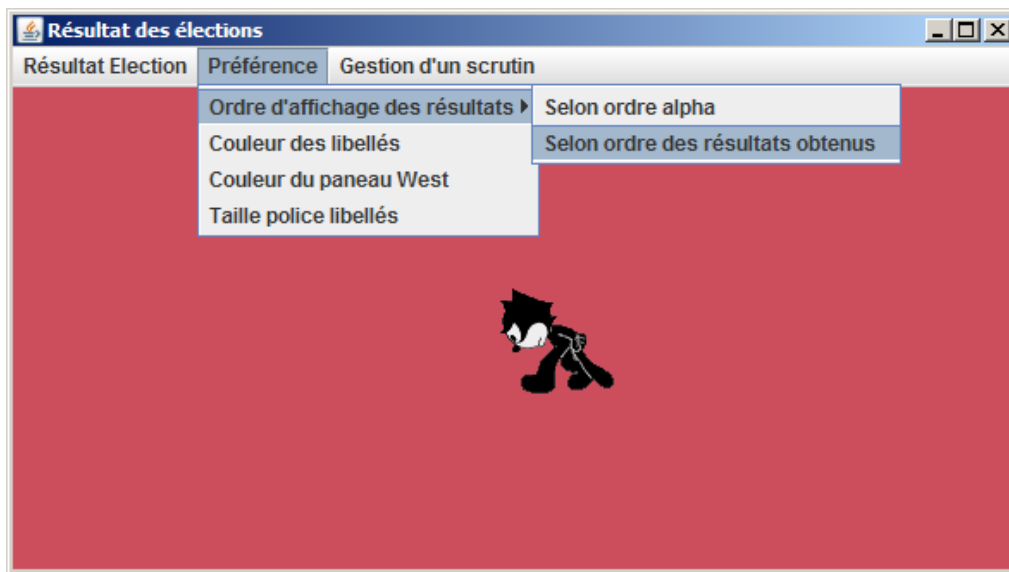
Exécutez l'application : a minima vous obtenez une fenêtre vide que vous pouvez redimensionner, mettre en icône, etc.



3. Complétez le constructeur par l'appel à une fonction `private` de mise en forme de la page d'accueil (à écrire de manière à changer la couleur du fond de la fenêtre et d'y ajouter au milieu une image d'accueil de l'application) et testez.



4. Complétez le constructeur par l'appel à une fonction *private* qui permet d'ajouter un menu à la fenêtre et testez.



Les options du menu et des sous-menus peuvent être les suivantes (celles en italique ne sont pas indispensables dans un 1<sup>er</sup> temps) :

- Résultat Election
  - Après simulation
  - *Après gestion d'un scrutin*
- Préférences
  - Ordre d'affichage des résultats
    - Par ordre alphabétique
    - Par ordre décroissant de résultat
  - *Couleur des libellés*
  - *Etc.*

Prévoyez de regrouper les menus principaux (JMenu) dans une barre de menus (JMenuBar). Les sous menus peuvent n'être que des JMenuItem pour simplifier ou des JMenu s'ils sont encore décomposés.

- a. Pour l'instant les menus sont affichés mais il ne se passe rien lors d'une sélection. Pourquoi ?

## Question 2 : écouteurs et mise en page du panneau de résultats

Il s'agit maintenant d'écouter le sous menu « Résultat Election / Après simulation » et de mettre en page le panneau d'affichage des résultats.

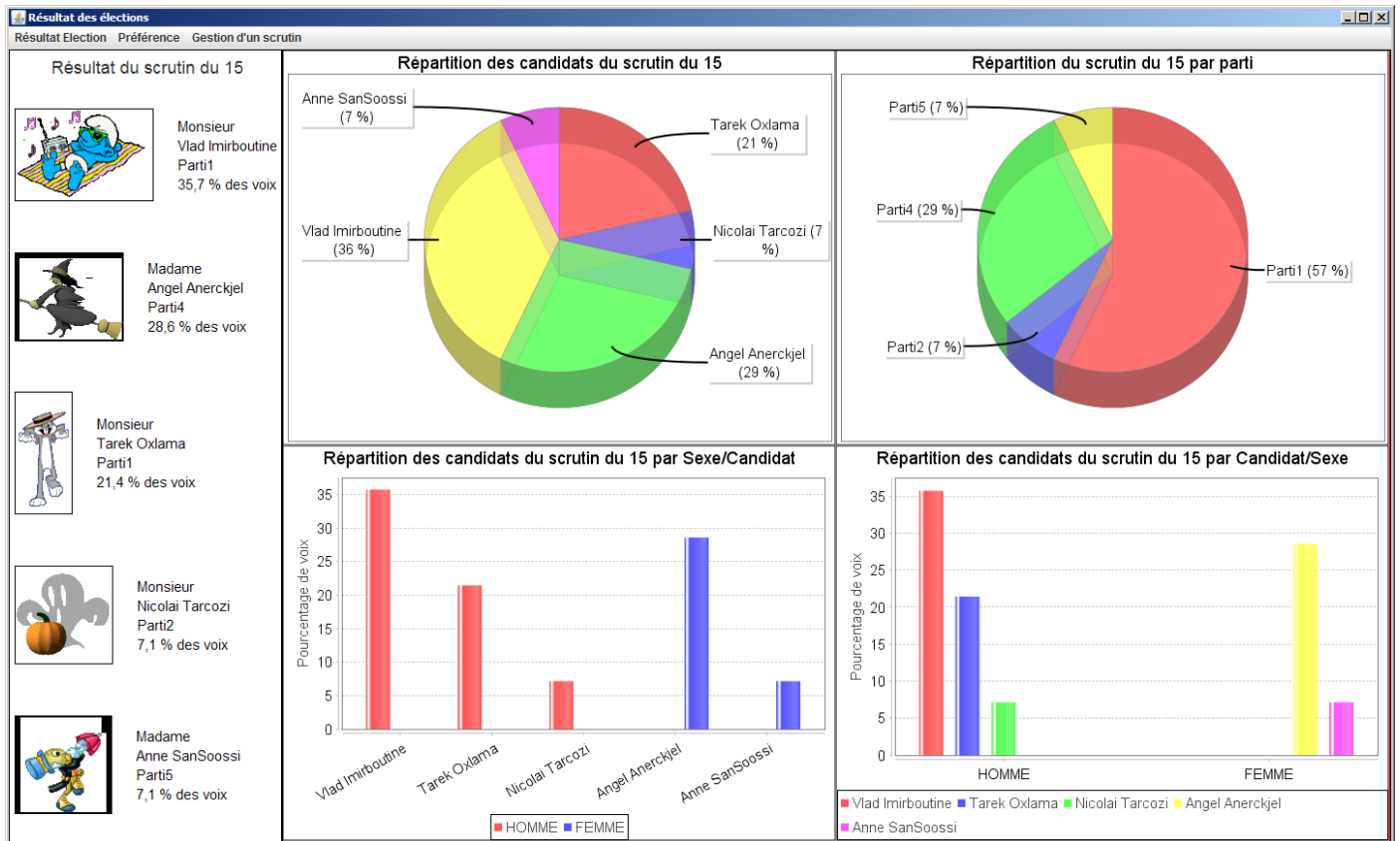
Comme sur l'exemple ci-dessous, le panneau doit se découper en une partie gauche (BorderLayout.*WEST*) et une partie centrale (BorderLayout.*CENTER*), elle-même découpée en 4 parties à l'aide d'un GridLayout de 2 lignes de 2 colonnes.

Lorsque l'item de menu est sélectionné :

- Une simulation est lancée,
- Les différentes collections de données sont mises à jour (liste de Candidat, map HommePolitique/image, map Candidat/Civilite, etc.),
- Les graphiques sont produits et affichés.

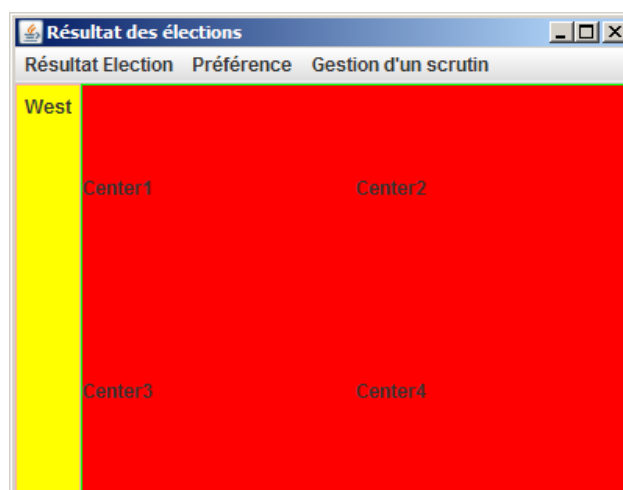
Le traitement de données est géré par la classe Election, la classe ElectionGUI doit se contenter d'invoquer ses méthodes, d'alimenter les graphiques avec les bonnes données et d'en gérer la mise en forme.

La classe `ElectionGUI` délègue à la classe `Election` le soin des traitements donc ses autres attributs ne doivent servir qu'à gérer le rendu. En revanche ses méthodes pourront définir localement des variables de manipulation des données en fonction des besoins (liste de Candidat, etc.).



1. Complétez le constructeur par l'appel à une méthode `private` qui permet de mettre en page le panneau d'affichage des résultats.

- Ce panneau doit être constitué d'un panneau `West` et d'un panneau `Center`. Créez les 3 `JPanel` et affectez leur les `Layout Manager` adéquats.
- Colorez chaque panneau avec une couleur différente (`setBackground()`). Eventuellement, prévoyez de pouvoir changer de couleur en cours d'exécution.
- Encadrez chaque panneau avec une couleur différente (`setBorder()`).
- Ajoutez 1 `JLabel` dans le panneau `West` et 4 `JLabel` dans le panneau `Center`.
- Testez.

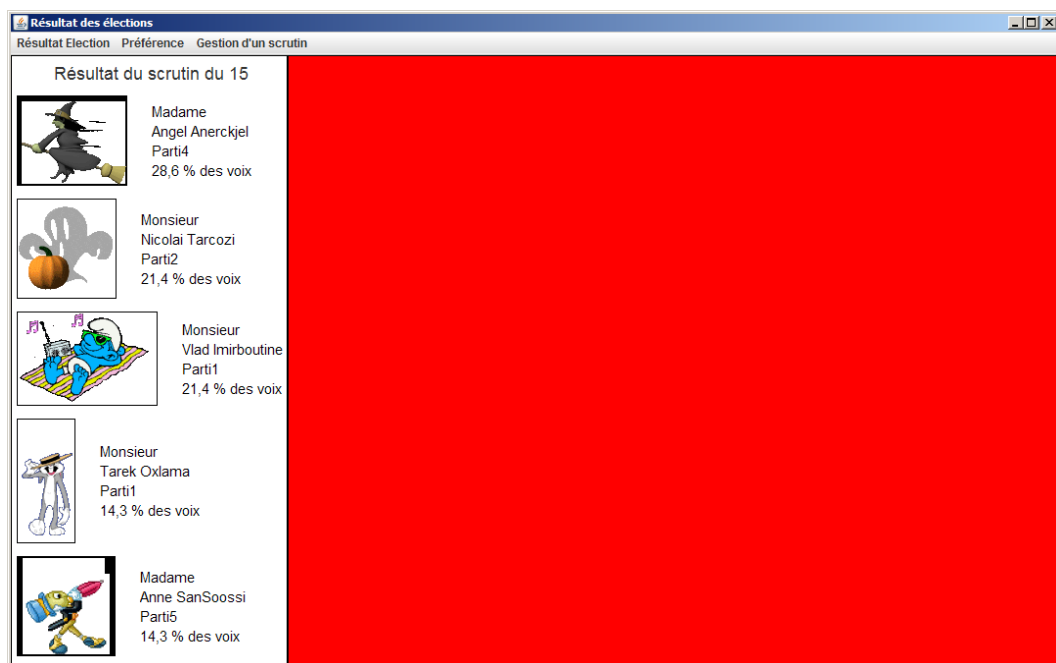


- Avez-vous vu votre page d'accueil ?
- Pourquoi ?

2. Substituez dans le constructeur l'appel à la méthode précédente par l'appel à une nouvelle méthode `private` qui permet d'écouter les items de menu.
  - Créez cette nouvelle fonction. Elle pourra regrouper tous vos écouteurs.
  - Faites écouter l'item de menu correspondant à « Résultat Election / Après simulation » par un écouteur d'évènements Action (`ActionListener`).
  - Faites lui appeler votre méthode de mise en page du panneau de résultat.
- a. Avez-vous vu votre page d'accueil ?
- b. Pourquoi ?

### Question 3 : délégation et mise en forme de données

Il s'agit maintenant de lancer la simulation et de mettre en forme les résultats obtenus dans le panneau West, le panneau Center restant inchangé.



1. Etudiez (et comprenez) le code de la classe `Election` (en particulier la méthode `simulation()`).
2. Complétez l'écouteur précédant en déléguant à la classe `Election` le soin de lancer la simulation (méthode `simulation()`).
  - Cette délégation doit se faire avant l'appel à votre méthode de mise en page du panneau de résultats.
  - La signature de cette dernière devra être modifiée de manière à récupérer en paramètre la collection de données obtenue à l'issue de la simulation.
  - Testez : il ne doit rien se passer de plus au niveau visuel que précédemment mais ça ne doit pas « planter ».
3. Enrichissez la méthode de mise en page par l'appel d'une méthode `private` qui remplit le panneau West et testez.
  - Cette méthode doit admettre comme paramètres :
    - La liste des candidats,
    - La map associant les candidats avec leur image (`election.newMapCandidatImage()`),
    - La date du scrutin.

- Pour la mise en forme, des images et des données : faites dans le très simple, inutile de perdre du temps en mise en forme. Le plus rapide pour mettre un peu en forme est d'utiliser des Box :
  - 1 box verticale (`Box.createVerticalBox()`) qui contient autant de box horizontales (`Box.createHorizontalBox()`) que de candidats.
  - Chaque box horizontale contient 1 image et 1 box verticale.
  - Box verticale dans laquelle sont ajoutées successivement la civilité, le nom, le parti, le pourcentage du candidat.
- 4. Ajoutez un écouteur (dans la bonne fonction...) pour écouter le menu Préférences / Ordre d'affichage des résultats / Par ordre alphabétique » et vérifiez que votre affichage se fait bien par ordre alphabétique.
  - Attention, la simulation ne doit pas être relancée et les résultats doivent être les mêmes qu'avant.
  - Utilisez les constantes définies dans l'Enum `DisplayOrder` disponible sur le e-campus.
- 5. Faites de même pour écouter le menu Préférences / Ordre d'affichage des résultats / Par ordre décroissant de résultat » et testez.

## Question 4 : utilisation d'une nouvelle librairie pour réaliser des graphiques

Il s'agit maintenant de compléter le panneau Center. 2 choix s'offrent à nous :

- Programmer les différents diagrammes (histogrammes, camemberts, etc.).
- Utiliser les classes d'une API déjà écrite par d'autres programmeurs avant nous. C'est évidemment ce choix que nous allons préférer.

JFreeChart est une API Java qui permet de créer des graphiques et des diagrammes de très bonne qualité. Cette API est open source et sous licence LGPL. En revanche, la documentation est payante.

1. Ajoutez les librairies JCommon et JFreeChart aux « Build Path » de votre projet (Cf. annexe).
2. Enrichissez la méthode de mise en page du résultat d'une élection par l'appel d'une méthode `private` qui remplit la 1<sup>ère</sup> case du panneau Center avec un camembert qui illustre les résultats par candidats et testez.
  - Cette méthode doit admettre comme paramètres :
    - La map associant les candidats avec leur image (`election.newMapCandidatImage()`).
    - La date du scrutin.
    - Le titre du diagramme.
    - + à réfléchir (Cf. question 4.3.).
  - Pour la mise en forme d'un camembert, étudiez les exemples relatifs aux Pie Chart ou aux Pie Chart 3D sur le lien référencé.
  - Pour occuper la 1<sup>ère</sup> case du panneau Center, un simple `add` du `ChartPanel` dans le `JPanel` suffit.
3. Enrichissez la méthode de mise en page du résultat d'une élection par l'appel d'une méthode `private` qui remplit la 2<sup>ème</sup> case du panneau Center avec un camembert qui illustre les résultats par parti et testez.
  - Cette méthode doit admettre comme paramètres :
    - La map associant les partis avec la somme des pourcentages obtenus (`election.newMapPartiPourcent()`).
    - La date du scrutin.
    - Le titre du diagramme.
    - + à réfléchir (Cf. question 4.2.).
  - Pour occuper la 2<sup>ème</sup> case du panneau Center, un simple `add` du `ChartPanel` dans le `JPanel` suffit.
  - Avec un peu de paramétrage, l'adaptation de la méthode précédente éviterait un copier/coller...

4. Enrichissez la méthode de mise en page du résultat d'une élection par l'appel d'une méthode `private` qui remplit la 3<sup>ème</sup> case du panneau Center avec un histogramme qui représente la répartition des résultats par candidats/sexe et testez.
  - Cette méthode doit admettre comme paramètres :
    - La map associant les sexes avec la liste des candidats de ce sexe (`election.newMapCiviliteCandidats()`).
    - La date du scrutin.
    - Le titre du diagramme.
    - + à réfléchir (Cf. question 4.5.).
  - Pour la mise en forme d'un histogramme, étudiez les exemples relatifs aux Bar Chart (Vertical) ou aux Bar Chart 3D sur le lien référencé.
  - Pour occuper la 3<sup>ème</sup> case du panneau Center, un simple `add` du `ChartPanel` dans le `JPanel` suffit.
  
5. Enrichissez la méthode de mise en page du résultat d'une élection par l'appel d'une méthode `private` qui remplit la 4<sup>ème</sup> case du panneau Center avec un histogramme qui représente la répartition des résultats par candidats/sexe et testez.
  - Cette méthode doit admettre comme paramètres :
    - La map associant les sexes avec la liste des candidats de ce sexe (`election.newMapCiviliteCandidats()`).
    - La date du scrutin.
    - Le titre du diagramme.
    - + à réfléchir (Cf. question 4.4.).
  - Pour occuper la 4<sup>ème</sup> case du panneau Center, un simple `add` du `ChartPanel` dans le `JPanel` suffit.
  - Avec un peu de paramétrage, l'adaptation de la méthode précédente éviterait un copier/coller...

## Question 5 : customisation

Au choix, vous pouvez programmer des écouteurs pour changer la couleur, la taille, des polices, des panneaux, etc.

Vous pouvez aussi prévoir la gestion d'un scrutin sans passer par une simulation (mais c'est un très gros travail...).

## Trucs et astuces

- Changer le Layout Manager d'un Container : `monContainer.setLayout(...)`
- Changer la couleur du fond d'un Component : `monComponent.setBackground(...)`
- Entourer un composant avec une bordure : `monComponent.setBorder(...)`  
Ex : `setBorder(BorderFactory.createLineBorder(Color.BLACK))` ;
- Changer la police d'un JComponent : `monJComponent.setFont(...)`
- L'alignement d'un JComponent : `monJComponent.setAlignmentX(...)`
- Mettre une image dans une étiquette : `monLabel = new JLabel(new ImageIcon(imageAccueil))`
- Ecouter un `AbstractButton` : `monAbstractButton.addActionListener(...)`
- Utiliser des menus : des `JMenu` composés de `JMenuItem` le tout stocké dans une `JMenuBar`
- Ouvrir une boîte de dialogue pour choisir une couleur : `JColorChooser.showDialog(...)`
- Ouvrir une boîte de dialogue pour saisir une valeur : `JOptionPane.showInputDialog(...)`



## Documentation à consulter (outre poly et/ou annexes)

### Doc de référence :

- <http://docs.oracle.com/javase/7/docs/api/>
  - Package javax.swing, java.awt
- <http://www.jfree.org/jfreechart/api/javadoc/index.html>
  - Package org.jfree.chart, org.jfree.data

### Tutoriaux :

- <http://docs.oracle.com/javase/tutorial/java/index.html>
- <http://www.java2s.com/Code/Java/Chart/CatalogChart.htm> : exemples d'utilisation des différents graphiques de l'API JFreeChart.
- <http://www.java2s.com/> : d'une manière générale, une foule de tutos et d'exemples sur ce site.

# Annexes

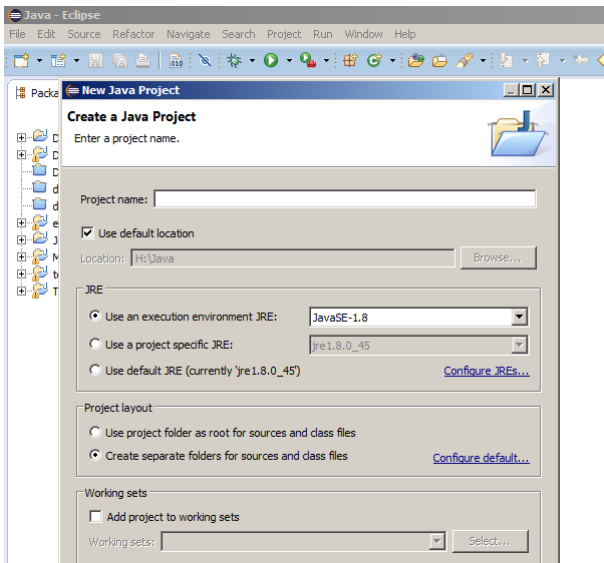
## Prise en main d'Eclipse :

Extrait de : <http://www.eclipse totale.com/articles/premierPas.html>

et de : <http://www.jmdoudoux.fr/java/dejae/index.htm>

## Une première application Java :

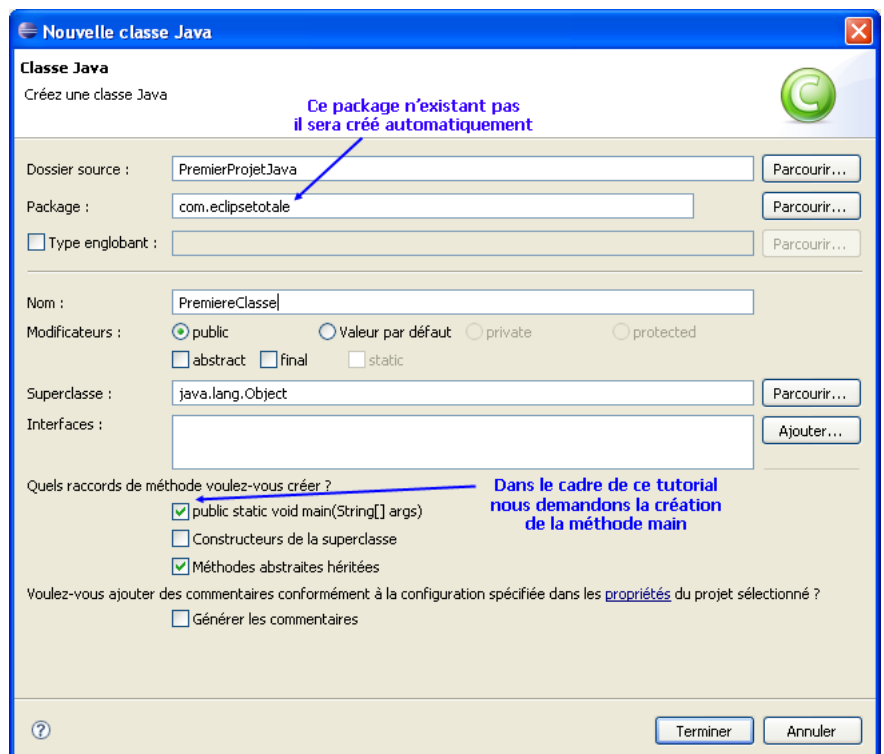
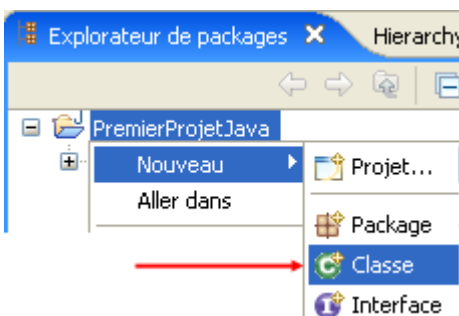
- lancer Eclipse et sélectionner le répertoire qui contiendra le répertoire de travail.  
Pour réduire les temps de réponse le workspace d'Eclipse sera placé en local dans le répertoire /var/tmp.
- créer un 'Projet Java' (Onglet "New", choisir "Java Project").



La seule information nécessaire pour l'instant est le nom du projet.

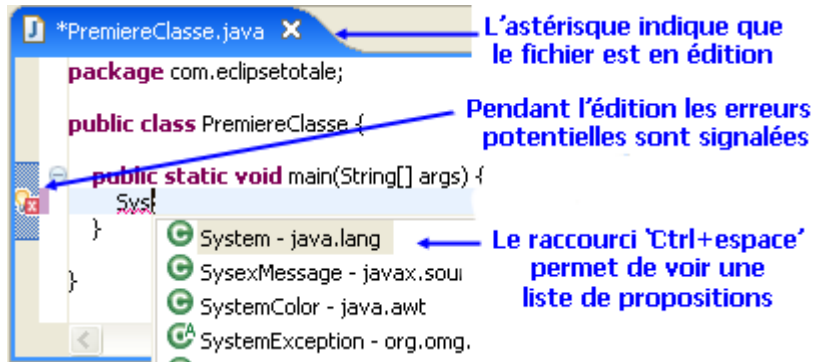
Après la création, le projet apparaît dans la vue 'Explorateur de projets' :


- créer une Classe :



Les informations importantes sont le nom de la classe et le nom de son package.

- Dans l'éditeur, compléter la classe (consultez <http://thierry-leriche-dessirier.developpez.com/tutoriels/eclipse/raccourcis/>) :



- Le raccourci Ctrl+Shift+F déclenche le formatage soit de tout le fichier soit des lignes sélectionnées. (Les options de formatage sont configurables dans Préférences->Java->Style de Code->Formater.)
- Une fois le code saisi, demander l'enregistrement (Ctrl+S ou menu Fichier), la classe est enregistrée et compilée.
- Pour demander l'exécution de la classe : déplier le menu associé au bouton  et sélectionner l'option '**Exécuter en tant que...**' -> '**Application Java**'.

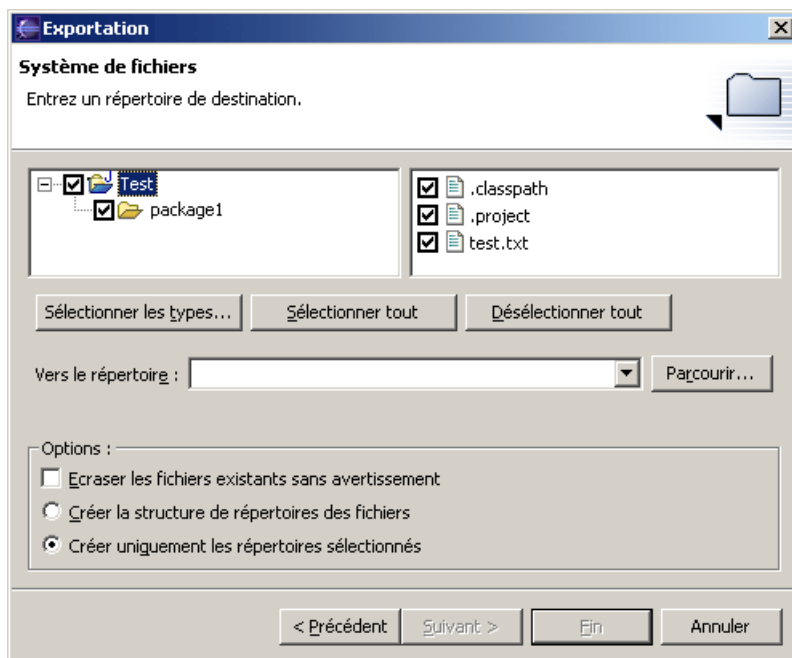
## Sauvegarder vos fichiers :

Pour exporter un projet vers votre espace personnel, utiliser le menu "**Fichier/Exporter**".

L'assistant demande de sélectionner le format d'exportation. Choisir le format "Système de fichiers".

Sélectionner TOUT le projet et indiquer le répertoire de destination.

Attention : pour que la structure des répertoires soit conservée dans la cible, il faut obligatoirement que les répertoires soient sélectionnés.



## Importer un projet (ou la copie d'un projet) :

Pour restaurer un projet dans le workspace d'Eclipse, utiliser le menu "Fichier/Importer/General/Existing Projects into workspace".

L'assistant vous demande d'indiquer le répertoire source où se trouve le projet à importer (bouton "Browse" (Parcourir) ). Pour travailler sur une copie de votre projet, cocher la case "copy projects into ws" (conseillé!).

## Tests unitaires

- Pour paramétrer le chemin d'accès vers les bibliothèques de test JUnit, utiliser le menu Project/Properties/Java\_Build\_Path/Libraries et cliquez sur le bouton "Add Library". Sélectionnez JUnit.
- Indiquez le chemin de la classe de test dans l'instruction d'appel des tests unitaires :  

```
public static void main(String[] args){
    org.junit.runner.JUnitCore.main("tp1.HommePolitiqueTest");
}
```

## Ajout de nouvelles librairies dans le « Build Path » d'un projet

1. Récupérez les librairies nécessaires (Exemple avec les bibliothèques graphiques).
  - Récupérez sur le e-campus les archives « jcommon-1.0.23.zip » et « jfreechart-1.0.19.zip » et les extraire dans le répertoire de votre projet (sur disque).
  - Ou bien, téléchargez la dernière version des librairies sur le site <http://sourceforge.net/projects/jfreechart/files/> .
2. Intégrez les librairies dans Eclipse.
  - Lancez Eclipse.
  - Sélectionnez Window puis Preferences. Dans le menu organisé sous forme d'arbre à droite de la fenêtre, sélectionnez Java > Build Path > User Librairies
  - Appuyez ensuite sur le bouton New... et entrez JCommon 1.0.23 comme nom de nouvelle librairie puis validez.
  - La librairie devrait désormais apparaître dans la liste à l'écran. Appuyez ensuite sur le bouton Add JARs... et sélectionnez l'archive jcommon-1.0.23.jar dans le répertoire JCommon téléchargé précédemment.
  - Double-cliquez sur la ligne Source attachment : (None) puis sur External Folder... et sélectionnez le sous-répertoire « source » dans le répertoire JCommon.
  - Procédez de même pour la librairie Jfreechart : Appuyer donc sur le bouton New... et entrez JFreeChart 1.0.19 comme nom de nouvelle librairie puis validez.
  - La librairie devrait désormais apparaître dans la liste à l'écran. Appuyez ensuite sur le bouton Add JARs... et sélectionnez l'archive jfreechart-1.0.19.jar dans le sous-répertoire « lib » du répertoire JFreeChart téléchargé précédemment.
  - Double-cliquez sur la ligne Source attachment : (None) puis sur External Folder... et sélectionnez le sous-répertoire « source » dans le répertoire JFreeChart.
  - Validez cette configuration en appuyant sur ok.
3. Ajoutez les librairies au Build Path de votre projet.
  - Positionnez-vous sur votre projet (dans le navigateur d'Eclipse) et faites un clic droit pour obtenir le menu contextuel.
  - Sélectionnez le menu Build Path puis Add Library...
  - Choisissez ensuite User Library puis cliquez sur Next...
  - Cochez les 2 librairies JCommon 1.0.23 et JFreeChart 1.0.19.
  - Cliquez sur Finish pour valider l'ajout des librairies.
  - Elles devraient apparaitre dans votre projet (dans le navigateur d'Eclipse) sous « JRE System Library ».

Diagramme UML de l'application :

