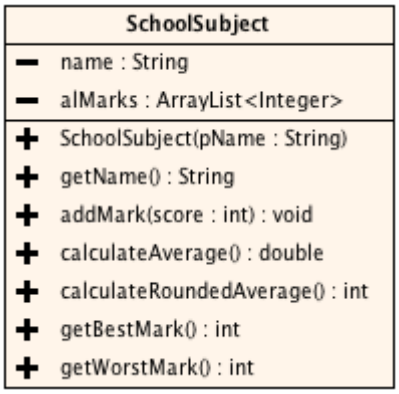
Série C : Listes

Table des matières

Série C : Listes....1
Exercice C.1: Branche....2
Exercice C.2: Liste de participants....3
Exercice C.3: Liste de nombres aléatoires....4
Exercice C.4: Envoi et réception de messages *(pour avancés !)*....4
Exercice C.5: Branche (GUI)....5
Exercice C.6: Listes de participants (GUI)....7
Exercice C.7: Liste de nombres aléatoires (GUI)....8
Exercice C.8: Nombres de Fibonacci....9
Exercice C.9: Equations du second degré....10
Exercice C.10: Evaluation de polynômes....12
Exercice C.11: GameShop....15
Exercice C.12: Liste de participants - Recherche linéaire....16
Exercice C.13: Nombres aléatoires - Suppression multiple....18
Exercice C.14: Liste de participants - Suppression multiple....18
Exercice C.15: Liste de participants - Tri par sélection directe....19
Exercice C.16: Recherche dichotomique (*pour avancés !*)....21
Exercice C.17: Recherche dichotomique (*pour avancés !*)....22

Exercice C.1: Branche

Développez le code correspondant au diagramme de classe suivant dans Unimozer. Il s'agit d'une classe permettant la saisie d'un nom d'une branche et des notes scolaires des élèves dans cette branche.



Indications:

1. **`getBestMark`** et **`getWorstMark`** retournent -1 si la liste est vide
2. **`calculateRoundedAverage`** retourne la moyenne arrondie vers le haut

Utilisez ensuite l'inspecteur d'objets afin de jeter un coup d'œil sur l'état interne de vos objets.

Etablissez un diagramme représentant les objets avec leurs relations pour deux branches (p.ex. Mathématiques et Anglais) contenant chacune quelques notes.

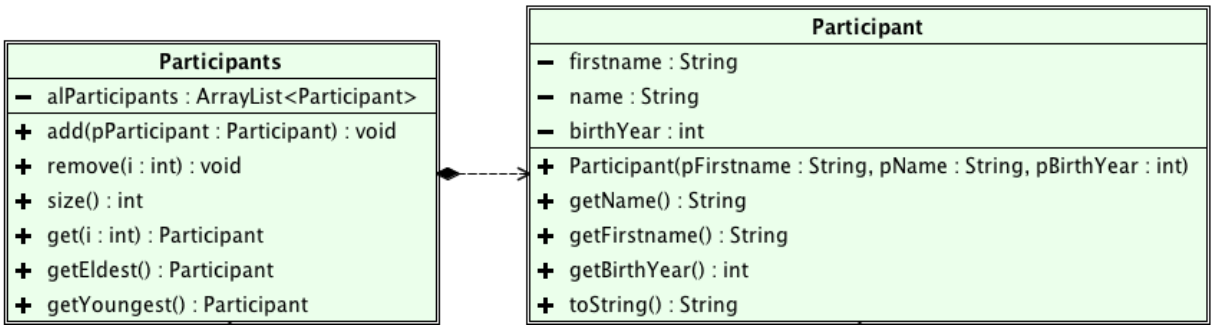
Notions requises: liste, typecast, Math
Structures requises: if, for

Exercice C.2: Liste de participants

Dans le cadre de l'organisation de cours du soir, on gère les inscriptions dans des listes de participants :

1. Un participant se compose d'un nom, d'un prénom et d'une année de naissance.
2. Un participant peut être inscrit sur une ou plusieurs listes de participants.

Implémentez dans Unimozer le code correspondant au diagramme de classe que voici :



Indications:

1. **`getEldest`** et **`getYoungest`** retournent **`null`** si la liste est vide
2. **`get`** retourne le participant qui se trouve à la position donnée par le paramètre i

Pour accélérer les tests, vous pouvez créer une classe avec une méthode **`main`**, p.ex :

```
public class Test
{
    public static void main(String[] args)
    {
        Participant josy = new Participant("Josy", "Muller", 1980);
        Participant poli = new Participant("Poli", "Klein", 1985);
        Participant mary = new Participant("Mary", "Nemo", 1978);
        Participants participants = new Participants();
        participants.add(josy);
        participants.add(poli);
        participants.add(mary);

        for(int i=0 ; i<participants.size() ; i++)
             System.out.println(participants.get(i));

        System.out.println("------------------");
        participants.remove(0);

        for(int i=0 ; i<participants.size() ; i++)
             System.out.println(participants.get(i));
    }
}
```

Représenter les objets de la méthode **main** ainsi que leurs relations dans un diagramme.

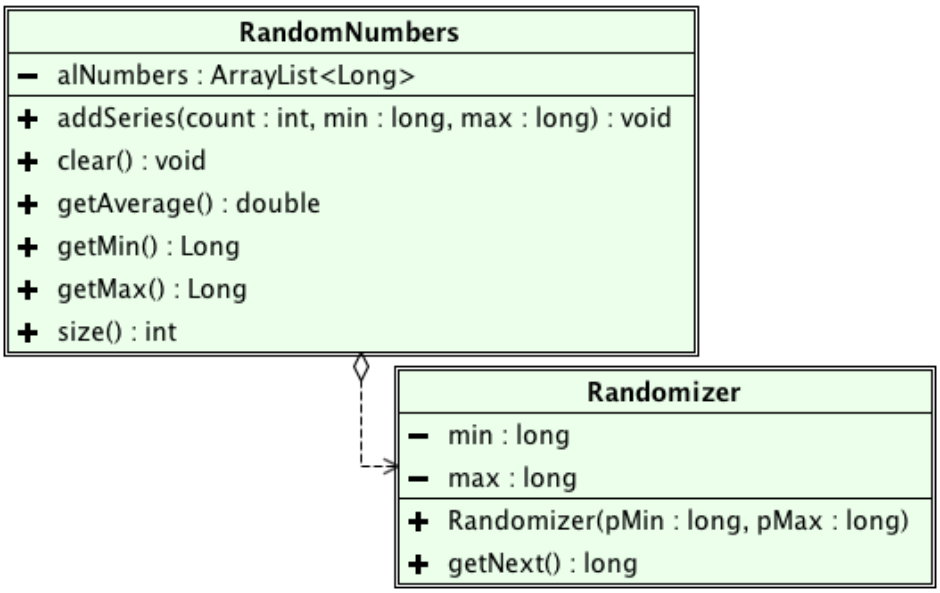
Remarque :

*Les méthodes de **Participants** qui servent uniquement à rendre publiques des méthodes qui sont déjà présentes dans **alParticipants** (ici les méthodes **add**, **remove**, **size**, **get**) se laissent facilement créer à l'aide de l'option **'Insert Code...'** - **'Delegate Method...'**. Profitez de cette fonctionnalité pour économiser du temps et pour éviter des fautes de frappe ! Ignorez le fait que **add** retourne un booléen et **remove** retourne un participant.*

Notions requises: liste, new, null
Structures requises: if, for

Exercice C.3: Liste de nombres aléatoires

En vous basant sur la classe « **`Randomizer`** » vue en classe de 3GIG, développez la classe « **`RandomNumbers`** » permettant de tester le générateur de nombres aléatoires !



En effet, en générant, par exemple, un très grand nombre de nombres aléatoires entre 0 et 10, la moyenne de tous les nombres générés devrait être proche de 5 (=milieu). En plus, on peut ainsi vérifier le minimum ainsi que le maximum.

Indications:

1. **`getMax`** et **`getMin`** retournent **`null`** si la liste est vide
2. **`size`** retourne le nombre d'éléments de la liste
3. **`clear`** efface le contenu de la liste

*[Remarquez bien que dans cet exercice il ne s'agit pas d'une liste de **Randomizer**s, mais d'une liste de nombres du type **Long.** La classe **Randomizer** sert uniquement à faciliter la création des nombres aléatoires.]*

Notions requises: liste, new, null, Math
Structures requises: if, for

Exercice C.4: Envoi et réception de messages *(pour avancés !)*

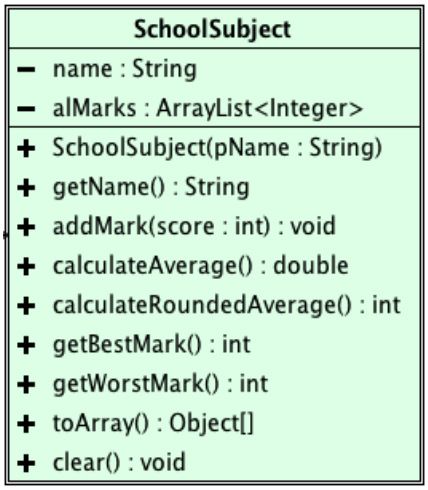
Un compte mail (classe « **`EmailAccount`** ») a un propriétaire (**`owner`**) ainsi qu'une boîte d'envoi (**`alSent`**) et de réception (**`alInbox`**). La boîte d'envoi ainsi que la boîte de réception sont des listes composées de messages (classe « **`Message`** »). Un message se compose d'un champ textuel (**`data`**), d'un émetteur (**`sender`**) et d'un récepteur (**`receiver`**).

Les intervenants (**owner**, **sender**, **receiver**) sont des personnes (contacts) avec un nom, prénom et une adresse email. Chaque compte mail possède une liste de contacts.

Créez toutes les classes nécessaires afin de pouvoir modéliser l'envoi et la réception des messages.

Notions requises: modélisation, listes, new, null

Exercice C.5: Branche (GUI)

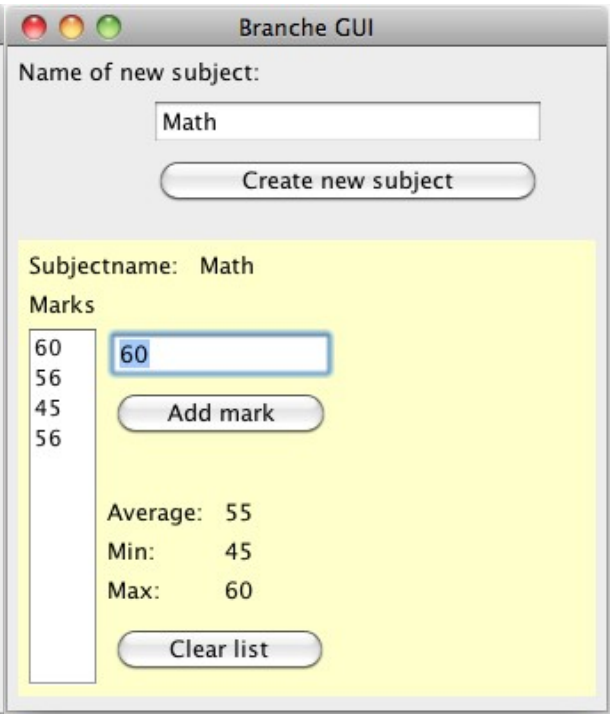


En vous basant sur la classe **`SchoolSubject`** de l'exercice C.1 ainsi que sur la capture d'écran suivante, développez une interface graphique !

Ajoutez dans la classe **`SchoolSubject`** une méthode qui retourne la liste entière et une méthode qui efface la liste.

Au démarrage, il faut d'abord créer une nouvelle branche, avant que les autres éléments ne soient visibles. Il est pratique, de placer tous les éléments à cacher sur un panneau (**`JPanel`**) et de modifier la propriété **`visible`** du panneau.

Interface graphique après le démarrage.



Interface graphique après avoir entré le nom de la branche.

Complétez le diagramme des objets que vous avez établi pour l'exercice C.1. Employez les nombres de l'exemple représenté ci-dessus.

Conseil :

*Générez les méthodes **toArray** et **clear** à l'aide de l'option **'Insert Code...'** - **'Delegate Method...'**.!*

Notions requises: MVC, new, liste, typecast, toArray
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Pour avancés (2GIN) :

Dans la suite, vous allez modifier le projet **SchoolSubject** de façon à ce qu'on puisse afficher plusieurs branches, séparément, chacune dans sa propre fiche.

Effectuez d'abord une copie du projet **SchoolSubject**, fermez l'original et ouvrez les fichiers de la copie :

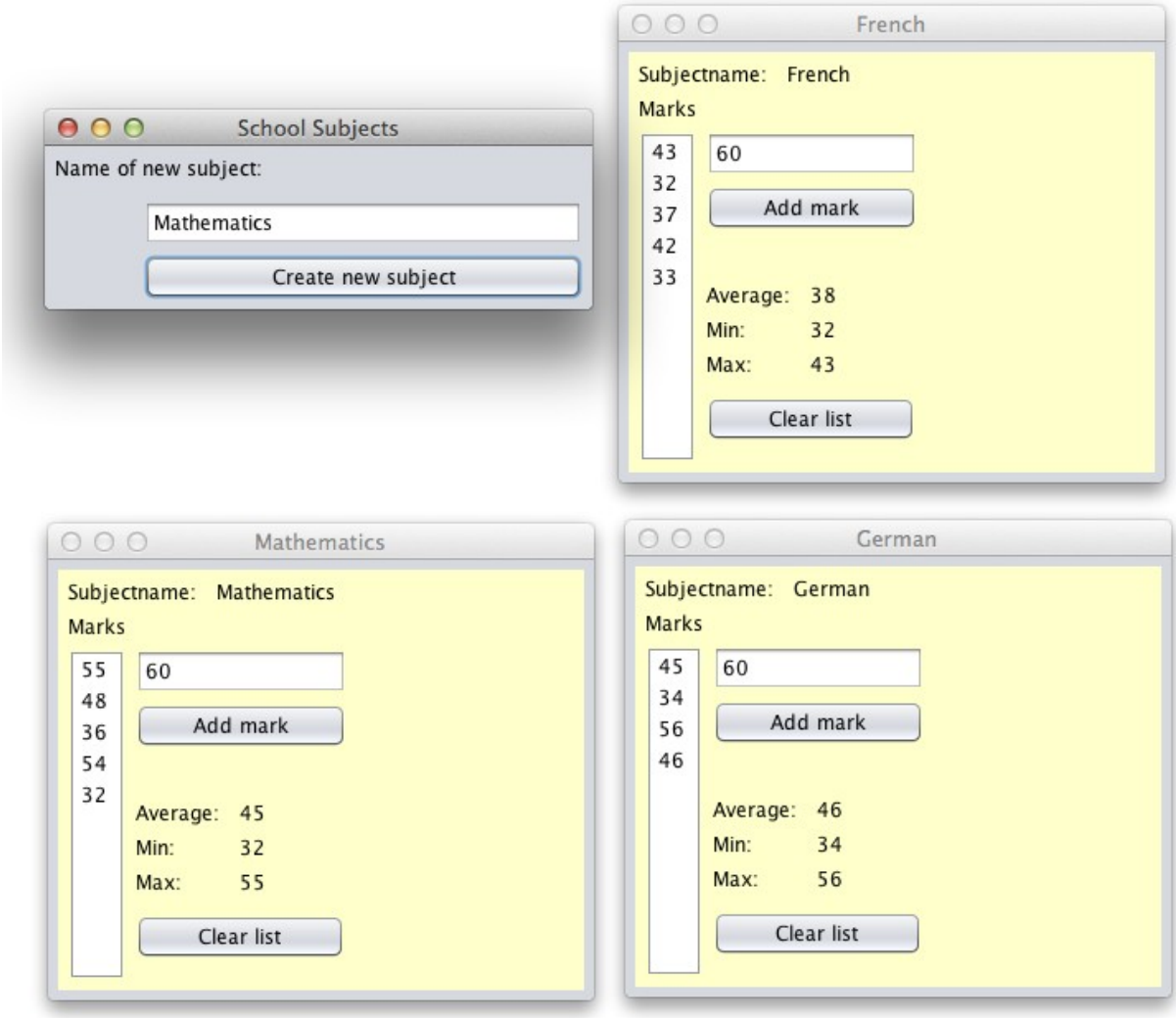
Renommez (→ *Refactor...*) la fiche **MainFrame** en **SchoolSubjectFrame.** Supprimez-en la méthode **main**().

Créez ensuite une deuxième fiche et appelez-la **MainFrame**.

Utilisez "couper & coller" pour transférer les composants **createSubjectButton, nameTextField** et le libellé correspondant de **SchoolSubjectFrame** vers **MainFrame.** (Remarquez que les méthodes de réaction sont transférées ensemble avec les composants!)

Modifiez les méthodes des deux fiches de façon à ce qu'on puisse créer dans **MainFrame** de nouvelles branches qui apparaissent alors chacune dans une nouvelle fiche du type **SchoolSubjectFrame**. Trouvez un moyen simple et logique pour passer le nom de la branche à la nouvelle fiche !

Inspirez-vous du code que vous trouvez à l'intérieur de la méthode **run** de la méthode **main** pour créer et afficher les nouvelles fiches.

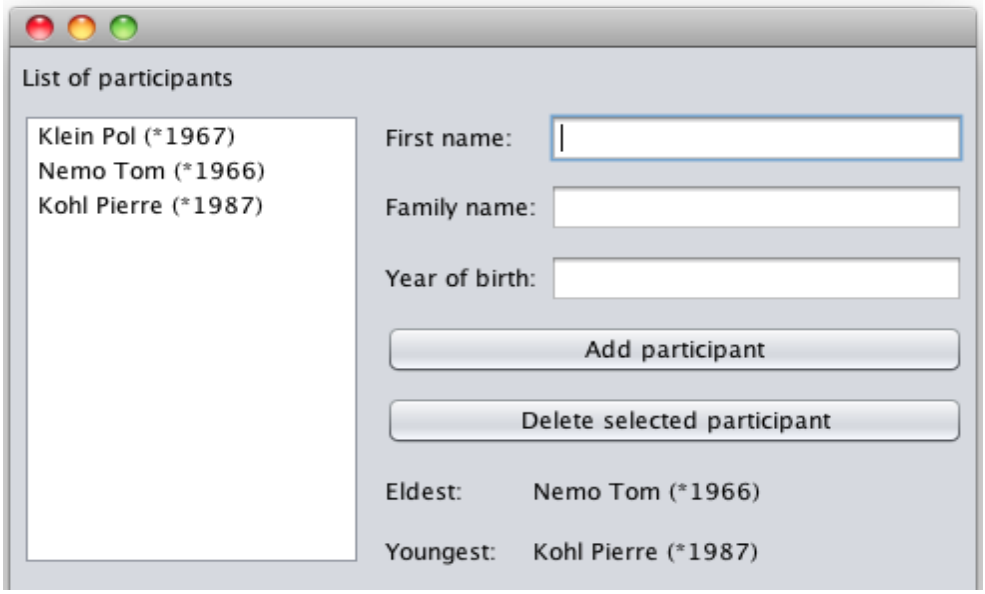


Exercice C.6: Listes de participants (GUI)

Reprenez l'exercice C.2 et développez une interface graphique pour la classe **`Participants`**. Le programme doit être capable d'ajouter et de supprimer des participants. Les données du plus jeune et du plus âgé des participants sont affichés à tout moment dans deux libellés.

Évitez les erreurs si la liste est encore vide ou si on essaie de supprimer un participant sans qu'un élément ne soit sélectionné dans la liste.

N'hésitez pas à compléter la classe **`Participants`** si vous le jugez nécessaire !



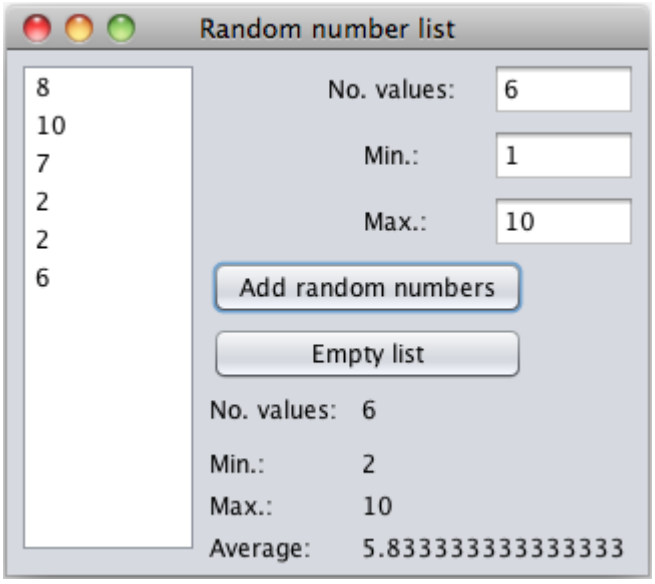
Complétez le diagramme des objets que vous avez établi pour l'exercice C.2.

Notions requises: MVC, new, liste, typecast, toArray
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.7: Liste de nombres aléatoires (GUI)

Reprenez l'exercice C.3 et développez un interface graphique pour la classe **`RandomNumbers`**. Le programme doit être capable de générer une liste de nombres aléatoires entiers entre les bornes « min » et « max » et de les supprimer. Des données statistiques (minimum, maximum, moyenne) sont affichées à tout moment.

N'hésitez pas à compléter la classe **`RandomNumbers`** si vous le jugez nécessaire !

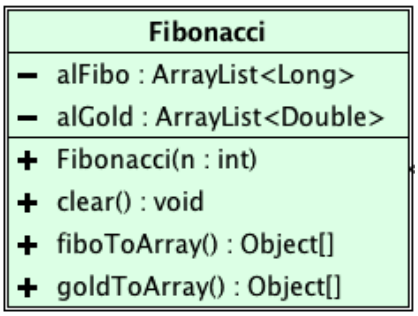


Notions requises: MVC, new, liste, typecast, toArray, Math
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.8: Nombres de Fibonacci

a) Recherchez sur Internet ce qu'on entend par **"les nombres de Fibonacci"**.

b) Créez une classe « **`Fibonacci`** » qui peut générer et mémoriser les **`n`** premiers nombres de Fibonacci. La liste s'appelle **`alFibo`** et le remplissage se fait à l'aide de la méthode **`generate(int n)`**.



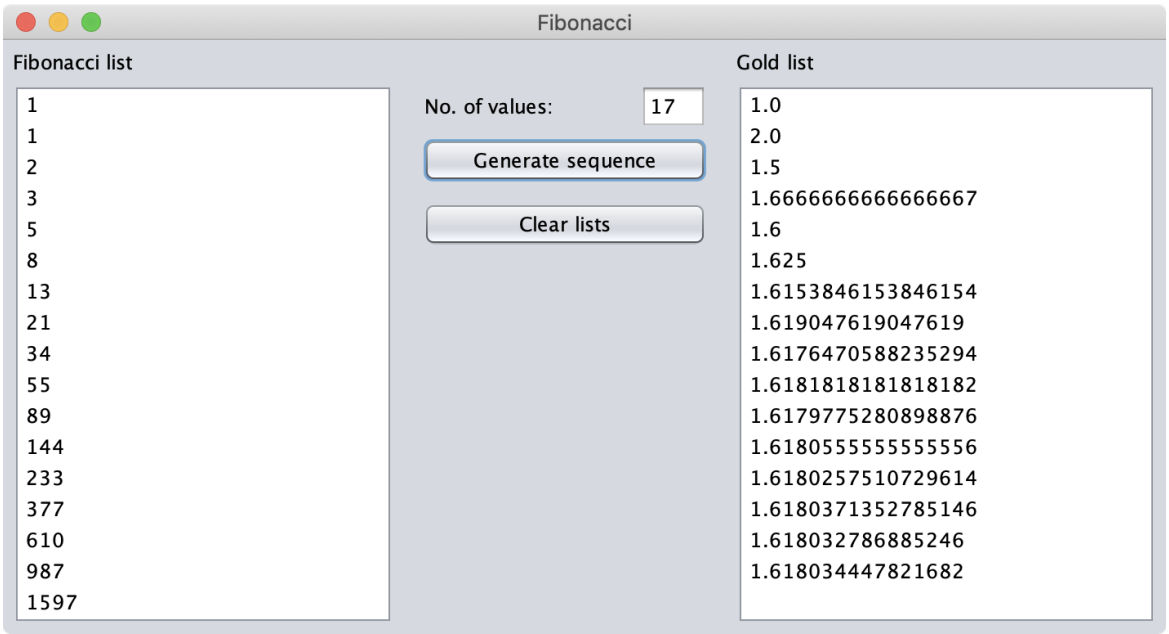
c) Développez une interface graphique pour générer et visualiser les nombres de Fibonacci ! Quelles sont les limitations pour Fibonacci, si la liste contient des nombres du type **Integer** ? du type **Long** ? du type **Double** ?

d) Ajoutez une deuxième liste du nom **`alGold`** à la classe **`Fibonacci`** et remplissez-la avec les quotients de chaque couple d'éléments consécutifs de la liste **`alFibo`**. C.-à-d. :

- le 1er nombre de **`alGold`** est le quotient du 2e élément de **`alFibo`** divisé par le 1er élément de **`alFibo`**
- le 2e nombre de **`alGold`** est le quotient du 3e élément de **`alFibo`** divisé par le 2e élément de **`alFibo`**
- le 3e nombre de **`alGold`** est le quotient du 4e élément de **`alFibo`** divisé par le 3e élément de **`alFibo`**
- etc.

e) Ajoutez une **JList** à votre interface pour représenter le contenu de **`alGold`**.

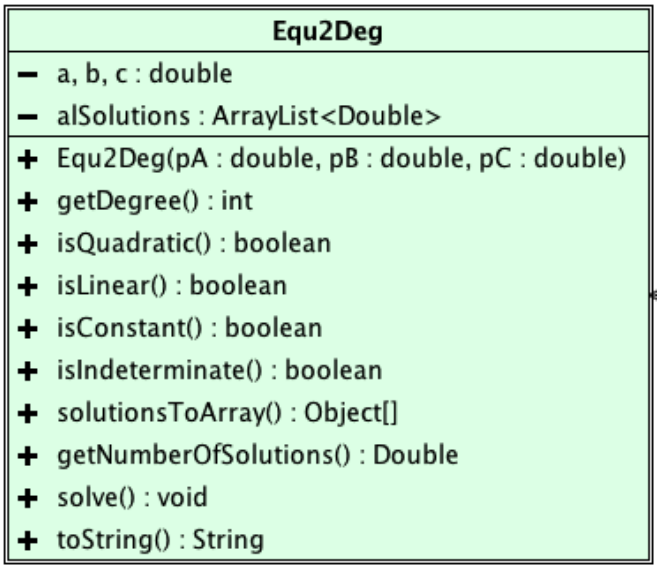
f) Que constatez-vous dans les valeurs de **`alGold`**? Que venez-vous de découvrir et quel est le rôle de votre découverte dans la nature et dans les arts ?



Notions requises: MVC, new, liste, typecast
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.9: Equations du second degré

Développez une classe permettant de résoudre une équation du second degré du type $a \cdot x^2 + b \cdot x + c = 0$. Les coefficients a, b et c sont sauvegardés dans 3 attributs.



Remarques :

Si a ≠ 0, on a une équation du second degré et il faut distinguer 3 cas en fonction de Δ (→ voir cours de mathématiques)

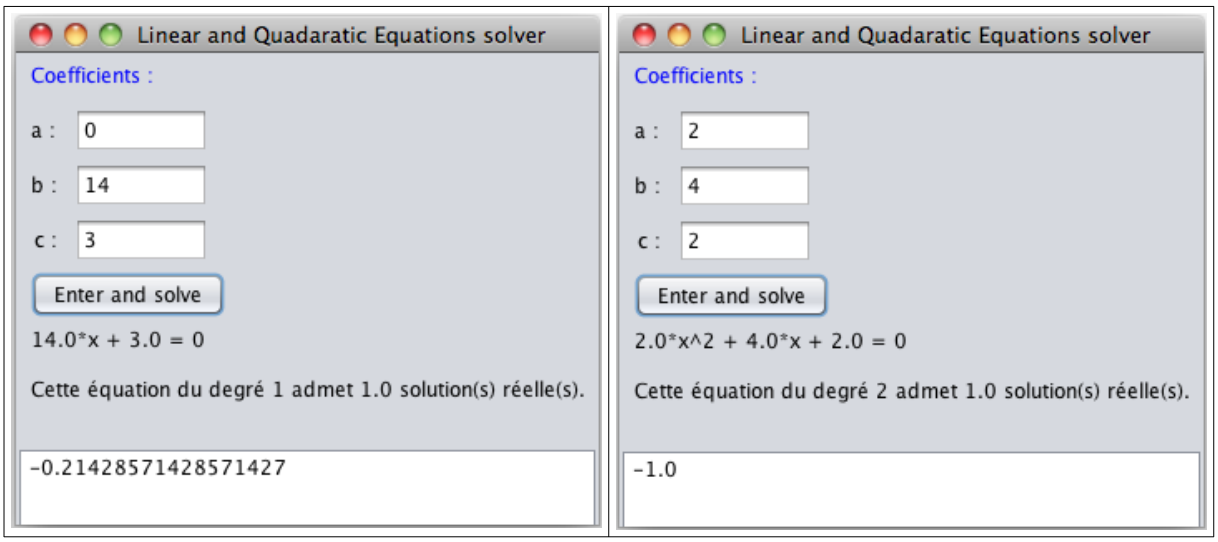
Si a = 0, on obtient : $0\ x^2 + b\ x + c = 0$

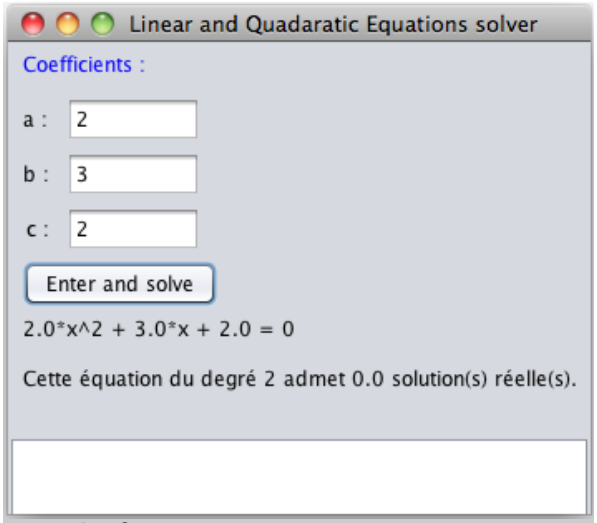
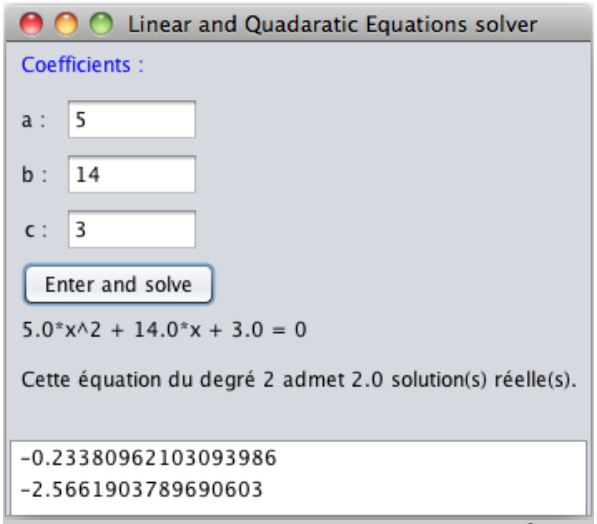
On se retrouve donc avec une équation du premier degré du type $b\ x + c = 0$.

Pour les équations du premier degré, il faut aussi distinguer trois cas. Lesquels ?

Dans l'un de ces cas on trouve une infinité de solutions (S = IR). Évidemment, nous ne pouvons pas entrer une infinité de solutions dans la liste. Pour ce cas, la méthode **`getNumberOfSolutions`** retourne la constante **`Double.POSITIVE_INFINITY.`** C'est d'ailleurs pour cette raison qu'on a défini cette méthode comme **`Double`** et non **`int`**.

Développez une interface graphique pour saisir les coefficients. Les solutions sont affichées dans une **`JList`**.



Cas où a = 0	Cas où $b^2 = 4ac$
	
Cas où $b^2 < 4ac$	Cas où a est différent de zéro et b^2 est supérieur à 4ac

Représenter dans un diagramme les objets ainsi que leurs relations comme ils se présentent pour le dernier exemple (après la résolution de l'équation).

Notions requises: MVC, new, liste, typecast, Math
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.10: Evaluation de polynômes

Considérez le problème d'évaluation d'un polynôme de degré **n** en une valeur **x**. En entrée nous avons les coefficients du polynômes : $\mathbf{a_n, a_{n-1}, ..., a_1, a_0}$ ($\mathbf{a_i}$ est le coefficient de $\mathbf{x^i}$) et la valeur de **x** pour laquelle nous voulons évaluer le polynôme. La valeur **x** et les coefficients sont des valeurs ***réelles***. On représente un polynôme **P(x)** de degré **n** sous la forme d'une liste (**`ArrayList`**). Dans la liste, le degré du coefficient correspond à l'indice dans la liste, c.-à-d. le coefficient de x^6 est sauvegardé à l'indice 6 de la liste. Le but de cet exercice est de mettre au point des algorithmes d'évaluation d'un polynôme **p** en une valeur **x** de plus en plus efficaces.

Algorithme naïf

On commence par développer l'algorithme venant naturellement à l'esprit :

On calcule la somme des exposants de chaque élément de la liste par sa position.

Exemple : $5x^2+4x^1+2x^0$ avec x = 2

Liste :

0	2
1	4
2	5

Pour l'évaluation on aura donc :

$2^0 = 1 \rightarrow 2 * 1 = 2$

$2^1 = 2 \rightarrow 4 * 2 = 8$

$2^2 = 4 \rightarrow 5 * 4 = 20$

----

somme : 30

Chaque terme de degré i nécessite i multiplications

- i-1 pour la puissance
- 1 pour la pondération

Pour une expression complète du degré **n** on aura donc

- **n** additions et
- n+(n-1)+(n-2)+...+1+0 multiplications ≅ $\mathbf{n}^2$ / 2 multiplications.

Algorithme plus efficace : méthode de Horner

On peut encore améliorer ce résultat en utilisant la façon suivante d'écrire un polynôme de degré n:

$$a_nx^n+\ldots\ldots a_2x^2 + a_1x + a_0 = (((\ldots((a_n)x + an_{-1})x \ldots)x + a_1)x + a_0)$$

Idée :

Mathématiquement, on écrit toujours un polynôme en commençant par les termes de plus haut degré. En factorisant **n** fois (mise en évidence de **x**), on peut éviter de recalculer des puissances de **x** qui ont déjà été calculées.

Par exemple :

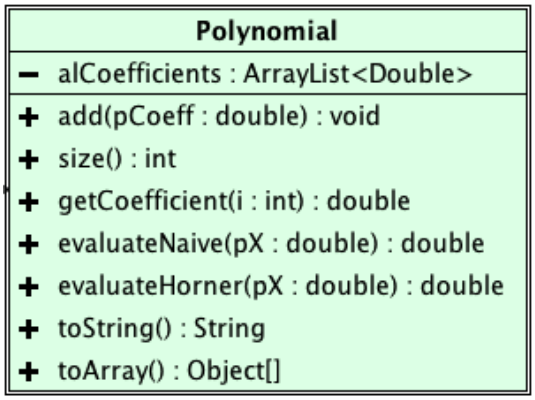
$$7x^5 + 2x^4 + 5x^3 + x^2 + 3x + 2 = ((((((7)x + 2)x + 5)x + 1)x + 3)x + 2).$$

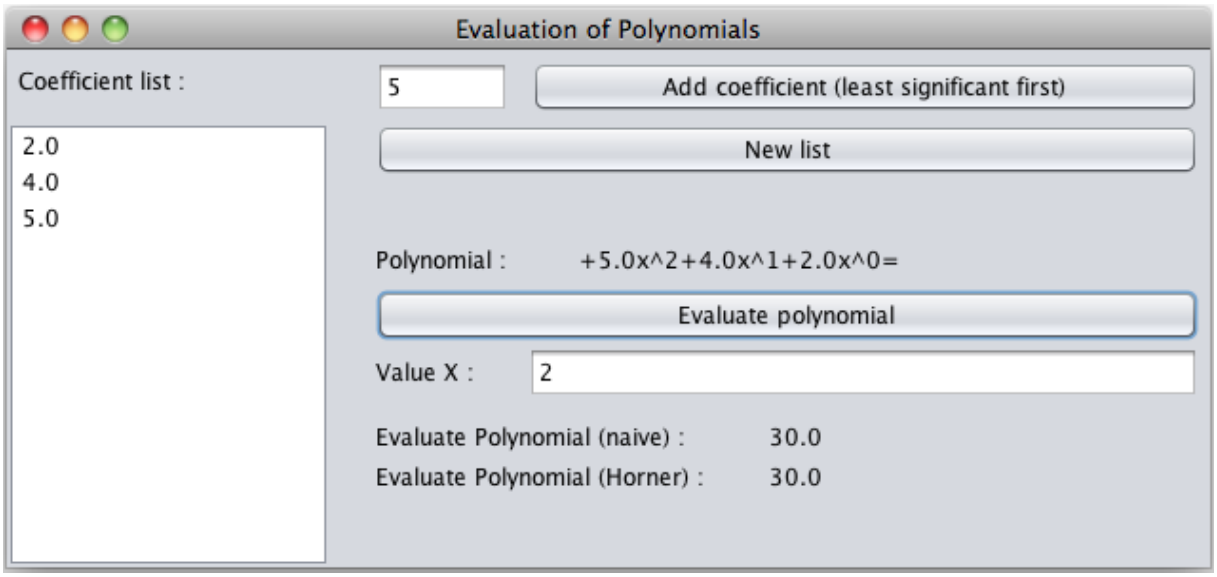
On vérifie que cet algorithme effectue :

- ***n*** additions et
- ***n*** multiplications.

L'algorithme est donc beaucoup plus effectif que l'algorithme naïf, surtout si de longs polynômes doivent être évalués avec un grand nombre de valeurs pour **x**. (Par exemple pour effectuer une représentation graphique d'un polynôme).

Développez la classe suivante et ensuite une interface graphique **`MainFrame`** :





Pour avancés /2GIN :

- Ajoutez à la classe `Polynomial` une méthode `differentiate()` qui retourne comme résultat un nouveau polynôme (instance du type `Polynomial`) qui correspond à la dérivée du polynôme actuel.
- Ajoutez à l'interface un bouton du nom **`differentiateButton`** qui ouvre une nouvelle instance de la classe **`MainFrame`** avec le polynôme dérivé du polynôme actuel. Pour voir la nouvelle instance, il faut rendre la fiche visible (à l'aide de **`setVisible(true)`**. Par défaut, la nouvelle fiche sera affichée à la même position que la fiche actuelle. A l'aide de **`getLocation/setLocation`**, vous pouvez afficher la nouvelle fiche de quelques pixels plus bas et plus à droite que la fiche actuelle.

Pour avancés^2 :

Dérivez un polynôme plusieurs fois. Fermez l'une des fenêtres qui contiennent une dérivée. Que constatez-vous ? Comment peut-on éviter cet effet indésirable (consultez le code caché de **`MainFrame`** et JavaDoc...) ?

Notions requises:	MVC, new, liste, typecast, Math
Composants requis:	JFrame, JButton, JLabel, JTextField, JList

Exercice C.11: GameShop

Dans un magasin de jeux vidéos les articles ont les propriétés suivantes :

- titre,
- catégorie (p.ex : action, stratégie, ...),
- console (p.ex : PS3, Wii, PC, …),
- limitation d'âge (p.ex : USK0, USK6, USK12, USK16, USK18),
- prix (en €).

Développez en Unimozer une classe **Game** dont chaque instance sait représenter un tel jeu. Sauvez le projet sous le nom **GameShop** dans le dossier **C11_GameShop**.

Ajoutez à la classe un constructeur initialisant tous les attributs et ajoutez les accesseurs nécessaires. Ajoutez une méthode **`toString`**.

Ajoutez à votre projet la classe **GameStock** servant à gérer une liste de jeux. Ajoutez les attributs et les méthodes nécessaires pour que vous sachiez initialiser la classe, ajouter des jeux, supprimer des jeux, exporter les jeux pour leur représentation en NetBeans, etc.

Fermez le projet et réouvrez-le en NetBeans. Ajoutez une fiche **MainFrame.** Complétez la fiche pour que vous sachiez :

- représenter les articles dans une liste (JList),
- ajouter des articles en indiquant leurs propriétés dans des champs texte,
- sélectionner un article dans la liste (JList) et représenter les données de l'article sélectionné dans les champs texte (TextField),
- supprimer l'article sélectionné,
- afficher la valeur totale de tous les articles.

Si nécessaire, complétez les classes **GameStock** et **Game.**

Réalisez une méthode **updateView** que vous appelez après chaque changement.

Représenter dans un diagramme les objets ainsi que leurs relations comme ils se présentent pour un 'mini-magasin' avec 4 articles.

Pour avancés/2GIN :

Ajoutez le nombre d'entités en stock pour chaque article.

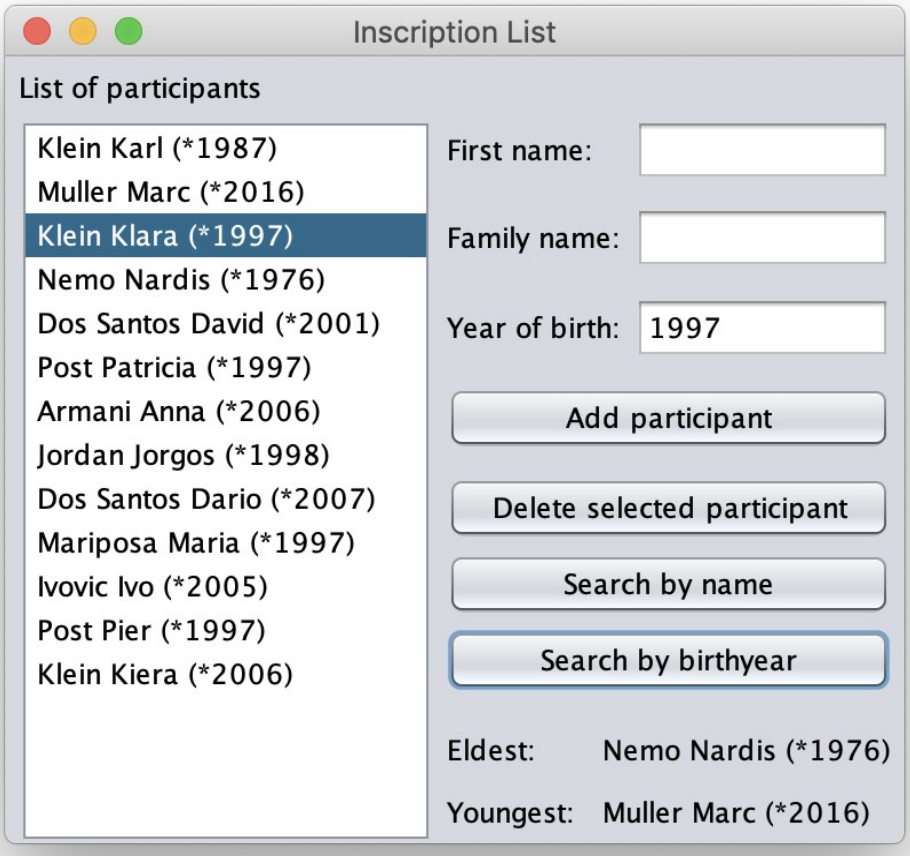
Ajoutez la possibilité d'acheter des articles (le stock diminue) et de remplir le stock pour l'article sélectionné. Adaptez les fonctionnalités existantes.

Notions requises:	planification, UML, MVC, new, liste, typecast, Math
Composants requis:	JFrame, JButton, JLabel, JTextField, JList

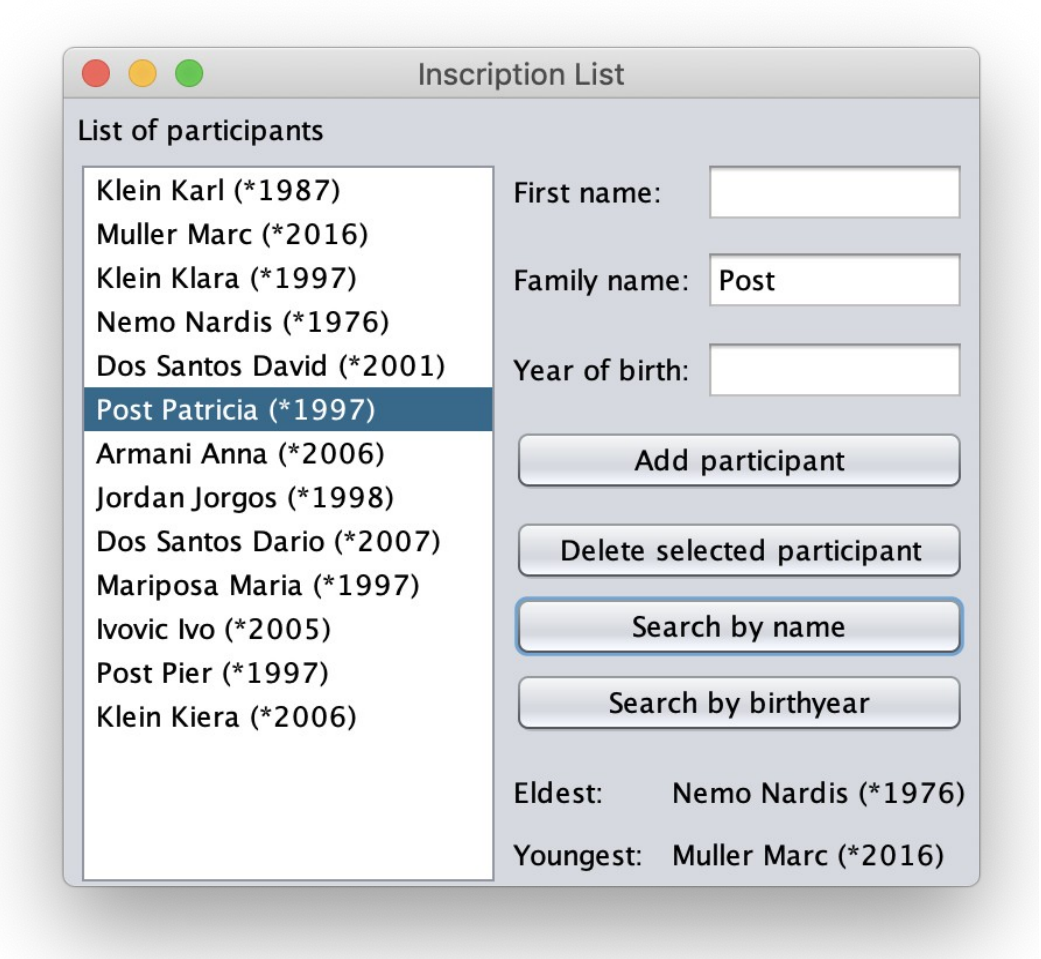
Exercice C.12: Liste de participants - Recherche linéaire

Reprenez l'exercice 'Inscription au cours du soir' (classes **`Participant`** et **`Participants`**)

1. Ajoutez à la classe **`Participants`** une méthode **`searchByBirthYear`** qui retourne comme résultat la position dans la liste du participant dont l'année de naissance est fournie comme paramètre. La recherche se fait de façon linéaire, c.-à-d. elle parcourt la liste du début à la fin et s'arrête dès que l'année recherchée a été trouvée. Par conséquent, la méthode doit retourner la position du premier participant dans la liste qui satisfait le critère de recherche. Dans l'exemple ci-dessous, **`searchByBirthYear(1987)`** retourne la valeur 2 car dans la liste le premier participant né en 1987 se trouve à la position 2. Si aucun participant, né à l'année spécifiée, n'est trouvé dans la liste, la méthode retourne **`-1`**.
2. Ajoutez dans l'interface graphique un bouton qui permet d'effectuer dans la liste des participants une recherche linéaire sur base d'une année de naissance saisie. Si un participant est trouvé alors il est marqué dans la liste affichée.



3. Ajoutez à la classe **`Participants`** une méthode **`searchByName`** qui retourne comme résultat la position dans la liste du participant dont le nom (de famille) est fourni comme paramètre. Si aucun participant, au nom spécifié, n'est trouvé dans la liste, la méthode retourne **`-1`**. La recherche se fait également de façon linéaire.
4. Ajoutez dans l'interface graphique un bouton qui permet d'effectuer dans la liste des participants une recherche linéaire sur base d'un nom de famille saisi. Si un participant est trouvé alors il est marqué dans la liste affichée.



Conseil pratique:

Pour accélérer les tests pour cet exercice et les exercices suivants, créez dans **`MainFrame`** un méthode **`initList()`** qui ajoute environ une douzaine de participants à la liste. Appelez cette méthode dans le constructeur de **`MainFrame`**.

Notions requises:	MVC, new, liste, typecast, String, boucles et indices
Composants requis:	JFrame, JButton, JLabel, JTextField, JList

Exercice C.13: Nombres aléatoires - Suppression multiple

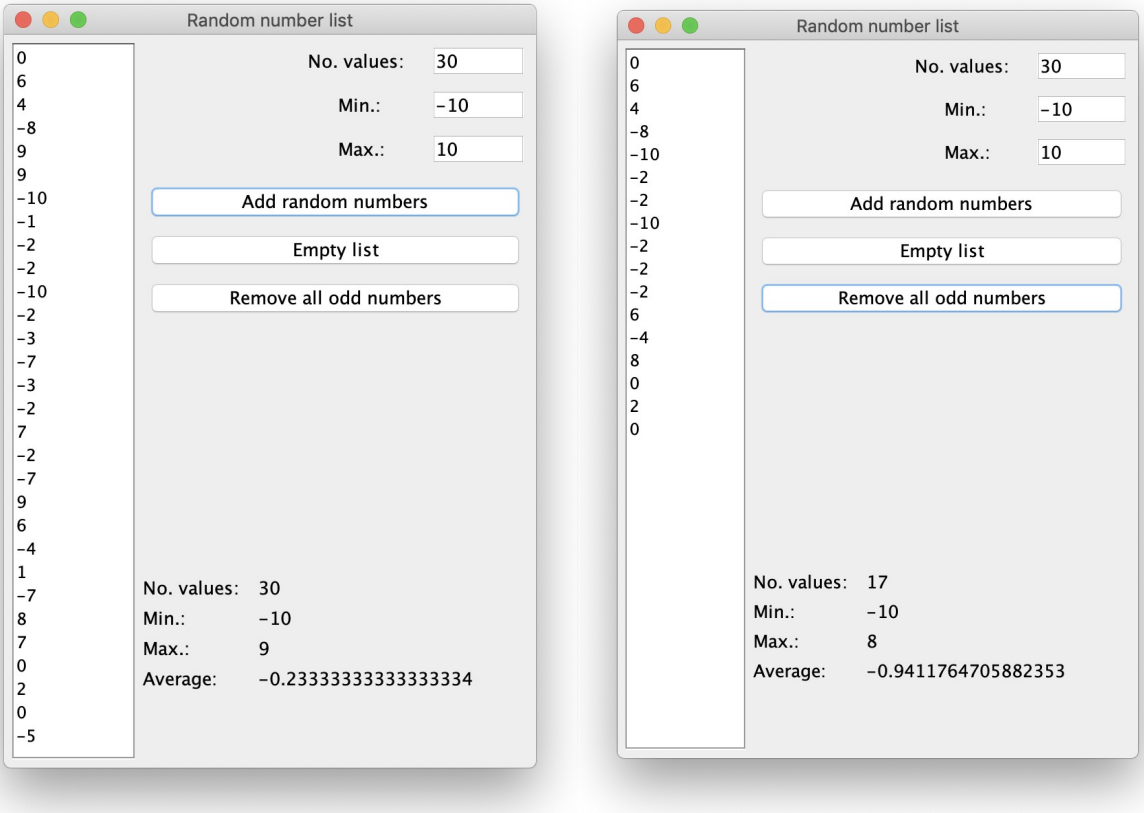
Reprenez l'exercice C.7 Nombres aléatoires.

1. Ajoutez à la classe **`RandomNumbers`** la méthode **`removeOddNumbers`** qui supprime tous les nombres impairs de la liste.
2. Ajoutez à l'interface graphique un bouton "**Remove all odd numbers**" pour tester la méthode.

Vérifiez le fonctionnement correct de la méthode!

Si la méthode ne réussit pas à supprimer tous les nombres impairs avec un seul clic, essayez de détecter dans quel cas des nombres impairs subsistent et trouvez la cause dans votre algorithme. Proposez une solution à ce problème!

Quelle conclusion générale pouvez-vous en déduire pour les méthodes qui doivent détecter et supprimer toute une série d'éléments d'une liste?



Notions requises: MVC, new, liste, typecast, toArray, Math, boucles et indices
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.14: Liste de participants - Suppression multiple

Reprenez l'exercice C.12 Liste de participants.

1. Ajoutez à la classe **`Participants`** une méthode **`removeAllMinorParticipants()`** qui supprime de la liste tous les participants mineurs (de moins de 18 ans).
2. Ajoutez un bouton "**Remove all minor participants**" pour tester la méthode.

Vérifiez le fonctionnement correct de la méthode !

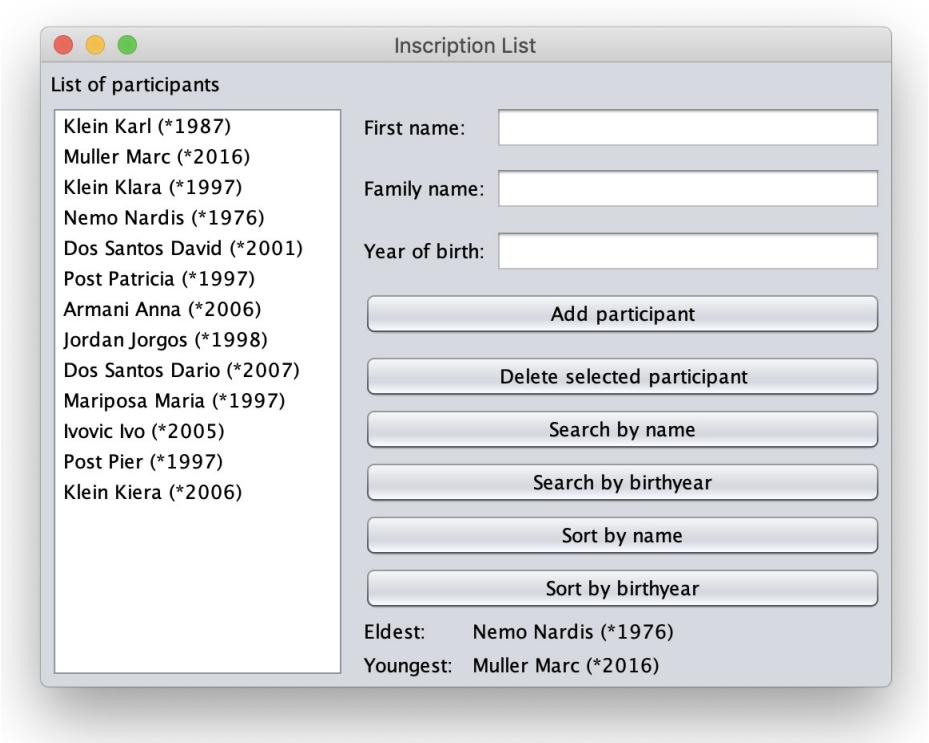
Notions requises: MVC, new, liste, typecast, toArray, Math, boucles et indices
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.15: Liste de participants - Tri par sélection directe

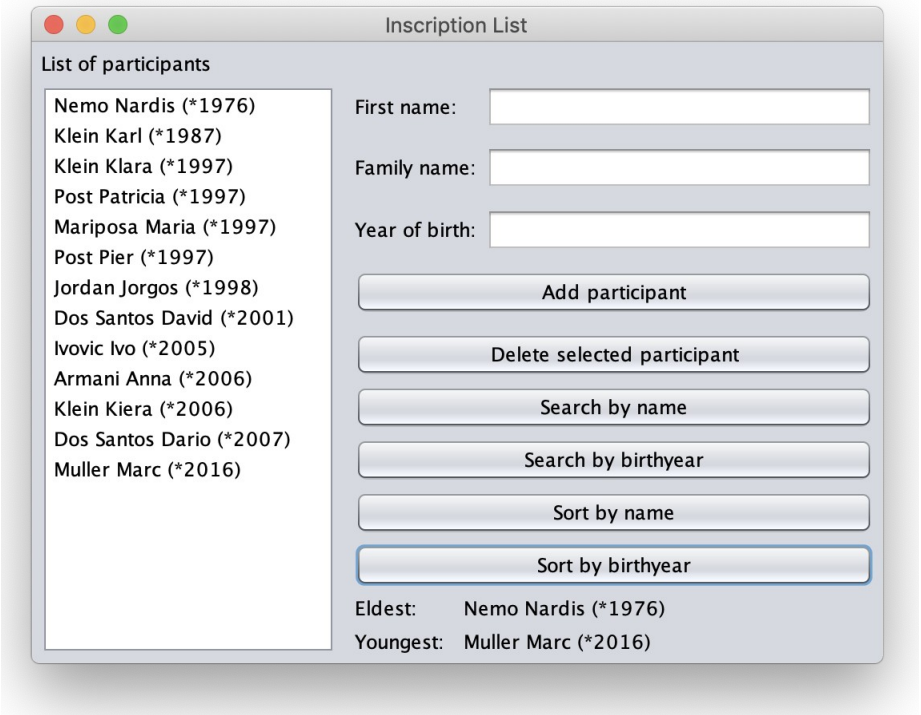
Reprenez l'exercice précédent.

1. Ajoutez à la classe **`Participants`** une méthode **`sortByBirthYear`** qui trie les participants de la liste d'inscription par année de naissance. Utilisez la méthode du tri par sélection directe (voir encadré ci-dessous).
2. Ajoutez dans l'interface graphique un bouton qui trie la liste d'inscription par année de naissance et qui affiche la liste triée.

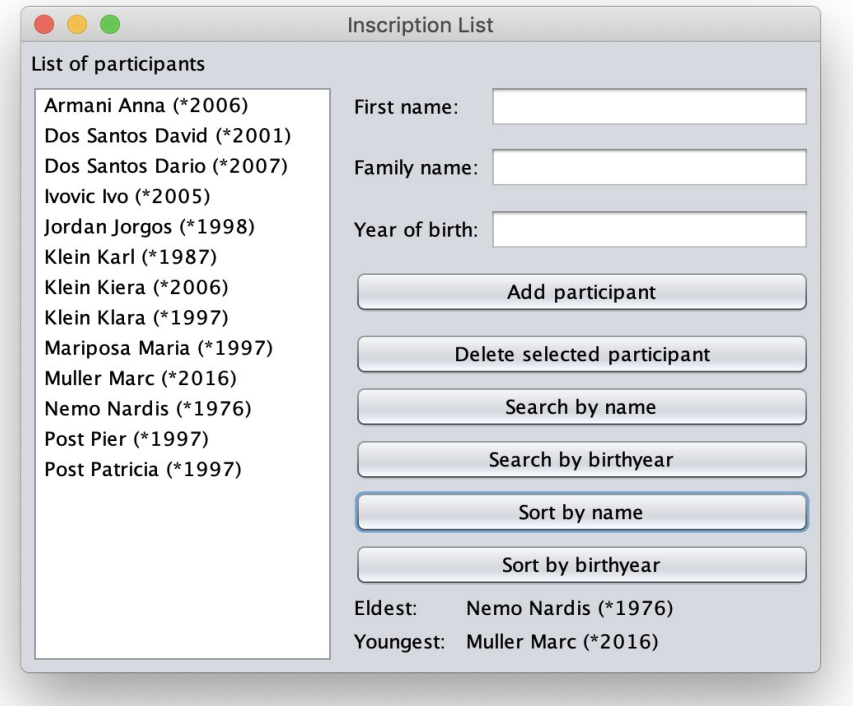
Avant le tri:



Après le tri:



3. Ajoutez à la classe **`Participants`** une méthode **`sortByName`** qui trie les participants de la liste d'inscription par leur nom de famille. Utilisez la méthode du tri par sélection directe.
4. Ajoutez dans l'interface graphique un bouton qui trie la liste d'inscription par le nom de famille et qui affiche la liste triée.

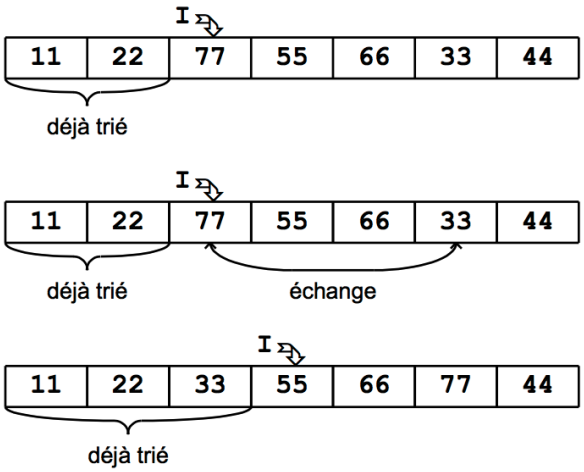


FR : Tri par sélection directe, EN : Selection Sort, DE : Sortieren durch direkte Auswahl

Problème: Classer les éléments d'une liste par ordre croissant (DE : *aufsteigend*) par la méthode du tri par sélection directe.

Méthode: Parcourir la liste à l'aide de l'indice **`I`** et échanger à chaque passage l'élément à la position **`I`** avec le plus petit élément entre la position **`I`** et la fin de la liste.

Exemple (3e passage) :



Notions requises: MVC, new, liste, typecast, Math, boucles et indices
Composants requis: JFrame, JButton, JLabel, JTextField, JList

Exercice C.16: Recherche dichotomique (*pour avancés !*)

Reprenez l'exercice précédent.

5. Modifiez la méthode **`add`** de **`Participants`** de façon à ce que les participants soient mémorisés dans la liste par ordre alphabétique de leur nom.
6. Dans la classe **`Participants`** modifiez la méthode **`searchByName`** en remplaçant la recherche linéaire par une recherche dichotomique (voir encadré ci-dessous).
7. Testez la recherche dichotomique dans l'interface graphique.
8. Quelle est l'avantage de la recherche dichotomique? Expliquez!
9. La recherche dichotomique retourne-t-elle encore le premier participant dans la liste qui satisfait le critère de recherche? Expliquez!

Principe **(FR : recherche dichotomique, EN : binary Search, DE : binäre Suche)** :

Exemple: la recherche se fait sur base du nom:

La liste doit être triée par nom.

Au départ, le domaine de recherche correspond à la totalité de la liste.

A chaque passage, le domaine de recherche est réduit à sa moitié.

A chaque passage, le nom à rechercher est comparé au nom au milieu du domaine de recherche :

- s'il y a égalité ou si la liste est épuisée, le traitement est arrêté et la position actuelle est retournée comme résultat (si la liste est épuisée, on retourne **`-1`**),
- si le nom recherché précède (lexicographiquement) le nom à la position actuelle, la recherche continue dans la moitié du domaine de recherche à gauche de la position actuelle,
- si le nom recherché suit (lexicographiquement) le nom à la position actuelle, la recherche continue dans la moitié du domaine de recherche à droite de la position actuelle.

Notions requises:	MVC, new, liste, typecast, String, boucles et indices
Composants requis:	JFrame, JButton, JLabel, JTextField, Jlist

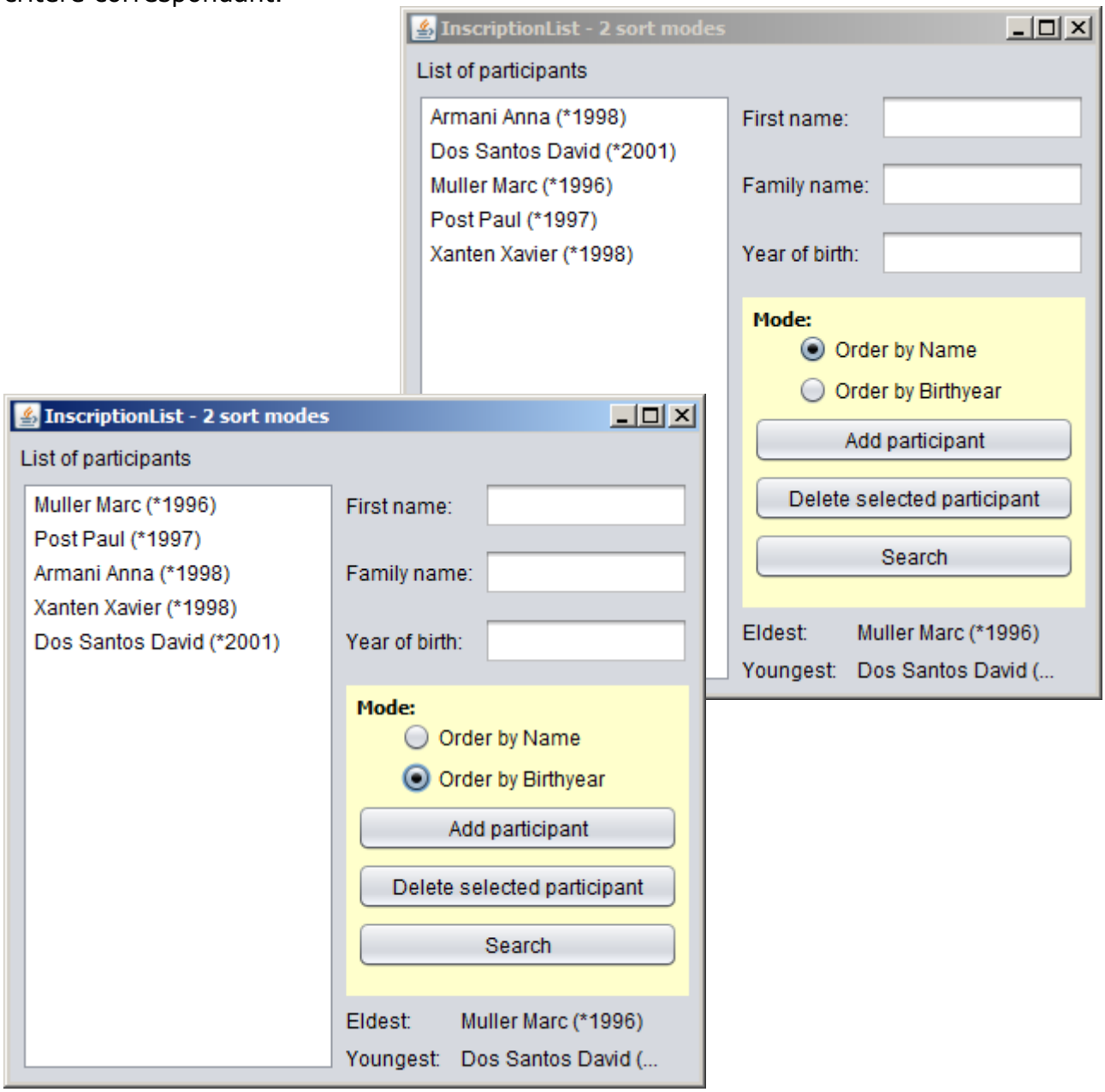
Exercice C.17: Recherche dichotomique (*pour avancés !*)

Reprenez l'exercice précédent.

Modifiez-le de façon à ce qu'il soit possible de travailler en deux modes : tri et recherche par nom ou tri et recherche par âge.

L'ajout et la recherche se font toujours selon le critère actuel. L'ajout se fait en ordre, la recherche à utiliser est la recherche dichotomique.

Lors du changement d'un mode à l'autre, la liste doit être (re)triée selon le critère correspondant.



Notions requises:	MVC, new, liste, typecast, String, boucles et indices
Composants requis:	JFrame, JButton, JLabel, JTextField, JList