Série E : Le temps et la date

Table des matières

Exercice E.1: Mesure du temps - chronomètre

Pour vous familiariser avec les méthodes de mesure de temps, développez d'abord un programme qui peut servir de chronomètre. Le programme possède un bouton [Start/Stop] et un libellé qui affiche le temps écoulé entre le démarrage et l'arrêt aussi précisément que possible.

a) Utilisez **System.nanoTime()**.

b) Ajoutez un deuxième libellé et utilisez **now().getNano()** ou **now().toNanoOfDay()** pour calculer la différence de temps.

c) Comparez les deux méthodes. Que constatez-vous?

d) Créez un nouveau projet et utilisez un **Timer** pour mesurer le temps écoulé. Calculez le temps écoulé à partir de la durée et du nombre des 'tics' du **Timer**. Affichez la durée écoulée constamment dans un libellé lorsque le chronomètre est actif.

e) Lorsque **Timer** est actif, affichez en plus (dans un autre libellé) la différence de temps qu vous obtenez avec **System.nanoTime()**. Que constatez-vous?

Comparez les résultats que vous obtenez et tirez-en des conclusions.

Exercice E.2: Performance des algorithmes de recherche

Développez une classe **`RandomNumbers`** qui contient une liste de **`n`** nombres entiers aléatoires entre **`min`** et **`max`**. Les nombres sont aléatoires, mais organisés par ordre croissant. **`n`**, **`min`** et **`max`** sont des attributs du type **`long`** qui sont initialisés lors de la construction d'une instance.

La méthode **`fill`** remplit la liste par **`n`** nombres entre **`min`** et **`max`**.

Méthode 1 (précise mais lente) :
Chaque nouveau nombre aléatoire est automatiquement inséré à la bonne position.

Méthode 2 (moins précise mais rapide) :
En partant de **`min`**, ajouter des valeurs aléatoires choisies dans un intervalle à ce que les dernières valeurs se trouvent autour de la valeur **`max`**.

La classe possède deux méthodes qui recherchent la position d'un nombre donné dans la liste (-1 si le nombre ne se trouve pas dans la liste.) :

La méthode **`linearSearch`** recherche le nombre en parcourant la liste du début vers la fin.

La méthode **`binarySearch`** recherche le nombre par la méthode de la recherche dichotomique.

Développez un programme qui sait tester la classe en recherchant des nombres dans la liste. Remplissez la liste par un grand nombre de valeurs (n>10000).

Déterminez le temps de recherche pour un seul nombre, pour 100 nombres, 1000 nombres, 10000 nombres,... avec les deux algorithmes.

Laissez maintenant la quantité de nombres à rechercher inchangée (p.ex 10000 valeurs) :

- Etablissez dans un tableur un graphique représentant les temps de recherche pour les deux algorithmes de façon à ce qu'on puisse comparer leur efficacité (temps de recherche).
- Soit n le nombre de valeurs dans la liste : Déterminez les temps de recherche pour 2n, 4n, 5n, 10n, … pour chacun des deux algorithmes.

Exercice E.3: Algorithmes de tri

Étendre la classe **`RandomNumbers`** pour trier des nombres aléatoires par les algorithmes suivants :

- tri par sélection *(selection sort)* → voir exercice du cours
- tri par insertion *(insertion sort)* → voir ci-dessous
- tri par propagation *(bubble sort)* → voir ci-dessous
- tri rapide *(quicksort)* → voir ci-dessous
- tri prédéfini des collections *(merge sort)* → voir ci-dessous

Ecrire un programme pour déterminer le temps de tri pour différents nombres de données.

Etablir des graphiques comparatifs dans un tableur (efficacité des algorithmes par rapport à leur initialisation, efficacité des algorithmes les uns par rapport aux autres). Testez la réaction des algorithmes pour différents états initiaux des données :

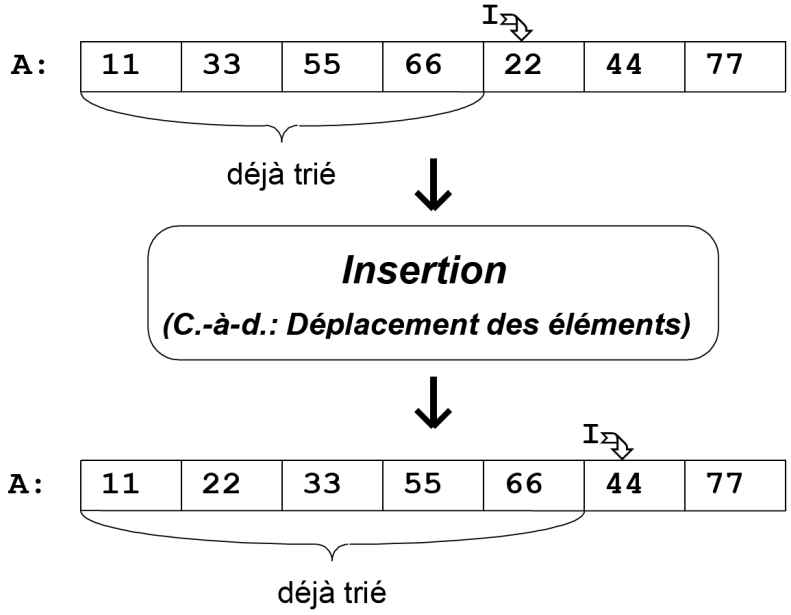
- Tableau aléatoire
- Tableau trié à l'envers au départ *('worst case')*
- Tableau déjà presque trié (p.ex. les derniers 10% du tableau sont des nombres aléatoires)

Tri par insertion - EN : Insertion sort - DE : Sortieren durch einfügen

Méthode: Le tableau est trié de gauche à droite. L'élément à droite de la partie déjà triée est inséré à la bonne position dans la partie déjà triée. L'insertion se fait en déplaçant les éléments plus grands que l'élément à placer. (Beaucoup de personnes utilisent une méthode pareille pour trier leurs cartes dans les jeux de cartes.)

Exemple (4e passage):

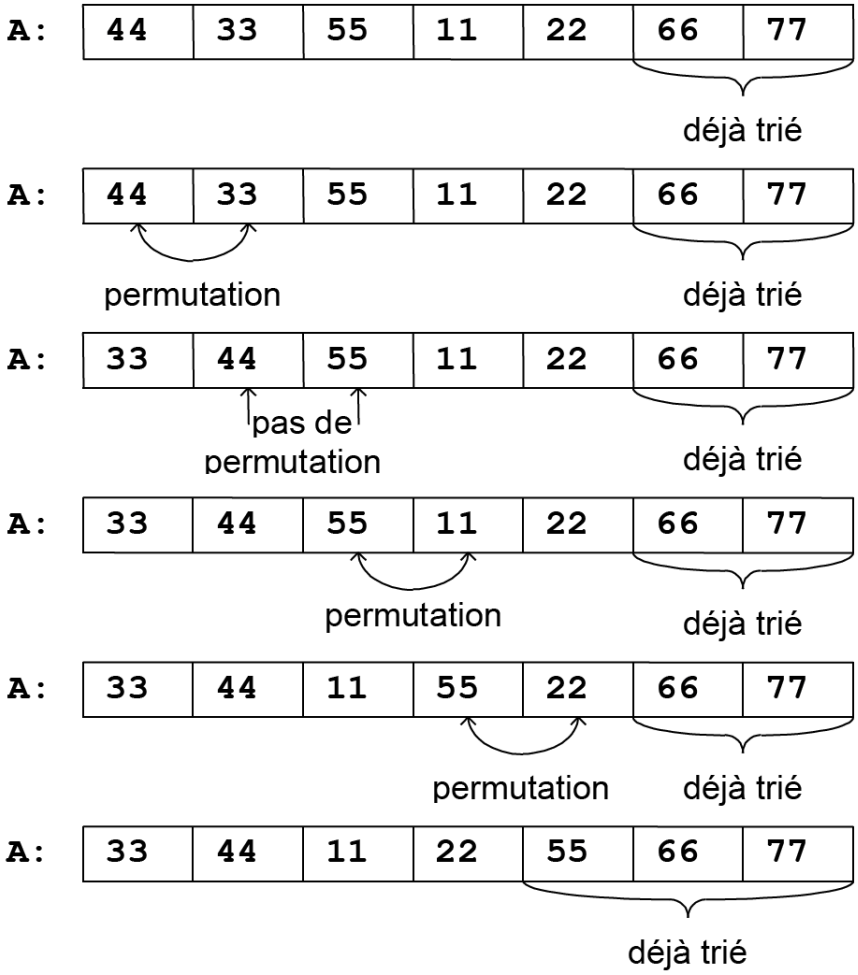
Chaque nouveau nombre aléatoire est automatiquement inséré à la bonne position.



Tri par propagation "Tri de la bulle" - EN : Bubble sort

Méthode: En recommençant chaque fois au début du tableau, on effectue à plusieurs reprises le traitement suivant: Par permutations successives, on fait avancer le plus grand élément du tableau vers la fin du tableau (comme une bulle qui remonte à la surface d'un liquide).

Exemple (3e passage):



Tri prédéfini des collections (MergeSort)

Une **`ArrayList`** est une '**Collection**' (→ voir plus tard dans le cours). Les collections ont une possibilité de trier une liste automatiquement, si elles connaissent un '**comparateur**' (→ voir plus tard dans le cours) pour comparer les éléments. Pour des éléments du type **`Long`**, un tel comparateur est prédéfini et ainsi, nous pouvons trier une **`ArrayList<Long> numbers`** en écrivant simplement :

```
Collections.sort(numbers);
```

L'algorithme utilisé est *MergeSort* (→ voir Wikipedia ou livre de Sedgewick : '*Algorithms*')

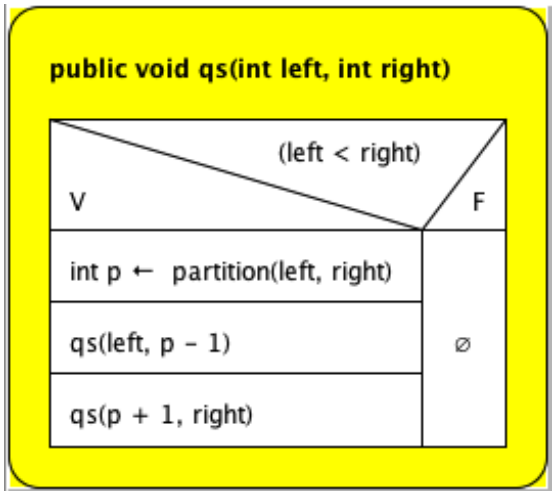
Tri rapide - EN : Quicksort

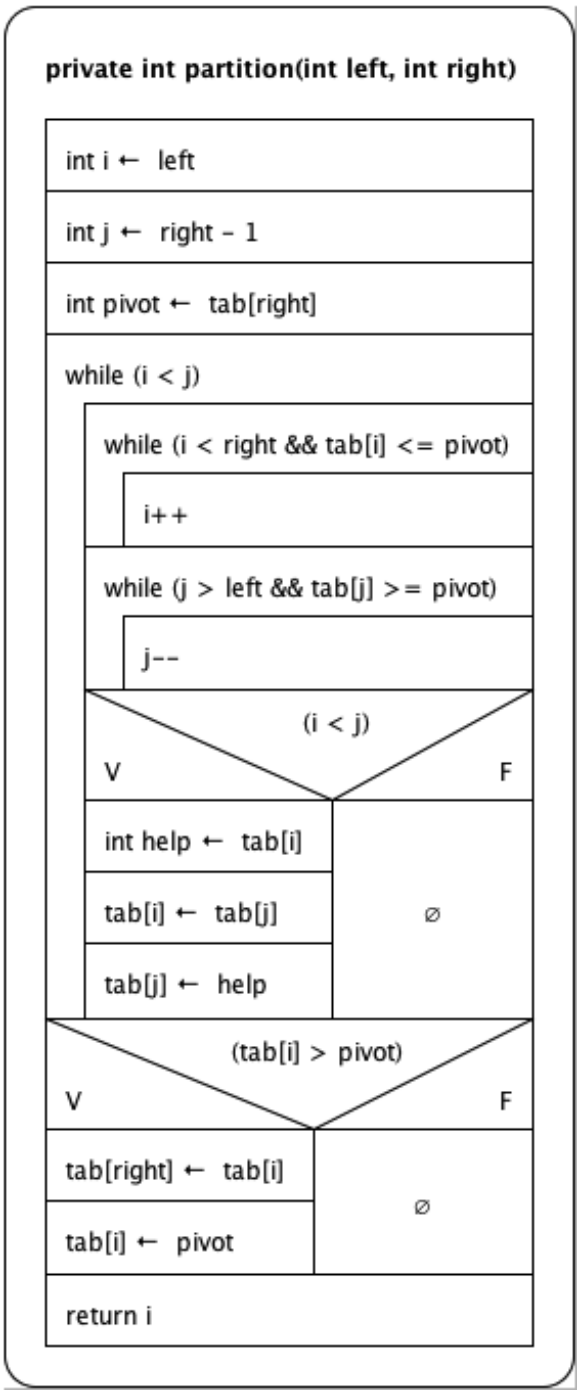
Méthode: algorithme récursif à voir en 1GIN (En résumé : Le tableau est divisé en deux parties et les éléments sont réarrangées dans chaque partie autour d'une valeur pivot - sans quand même les trier. Cette méthode est répétée pour chaque division du tableau jusqu'à la plus petite...)

Structogramme :

`tab[i]` correspond à l'élément à la position i de la liste (à adapter pour votre **`ArrayList`**).

Appel initial : **`qs(0, numbers.size()-1) ;`**





Exercice E.4: La date actuelle

Créez une application qui affiche la date et le temps actuel dans deux libellés distincts lors d'un clic sur un bouton. Employez les formats suivants :

<heures>:<minutes>:<secondes>

et

<jour>/<mois>/<année>

Actualisez l'affichage 5 fois par seconde.

Exercice E.5: Montre analogique

Développez une application qui affiche le temps actuel sous forme d'une montre analogique. Le cadran de la montre est toujours circulaire avec des marques pour les heures et possède trois aiguilles pour les heures, les minutes et les secondes. Le cadran s'adapte au panneau sur lequel il est dessiné. Les aiguilles des minutes et des heures avancent progressivement avec le passage du temps (l'aiguille des minutes est actualisée pour chaque seconde, l'aiguille des heures est adaptée pour chaque minute).
La date est affichée sur le panneau sous forme de texte.

Définir les classes **AnalogWatch** et **DrawPanel** avec les méthodes et attributs nécessaires en respectant bien le modèle MVC et le principe de l'encapsulation (c.-à-d. uniquement les éléments nécessaires sont accessibles à l'extérieur de chaque classe). Les noms utilisés doivent respecter les conventions du cours.

Définissez et utilisez dans **AnalogWatch** la méthode

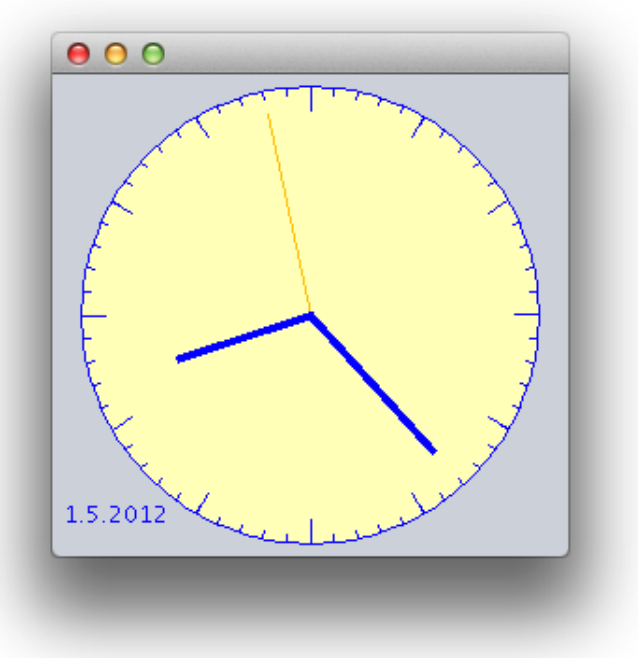
public Point polar2cart (Point center, double radius, double angle)

qui retourne le point (type prédéfini **Point**) correspondant aux coordonnées polaires données par center, radius, angle. L'angle est donné en radians ou en degrés selon votre préférence. Ajoutez cependant un commentaire qui indique l'unité que vous utilisez dans votre programme !

Placez le panneau sur une fiche et actualisez l'affichage 5 fois par seconde.

Amélioration :

En gardant les mêmes fonctionnalités, personnalisez la présentation de votre montre selon votre goût personnel.

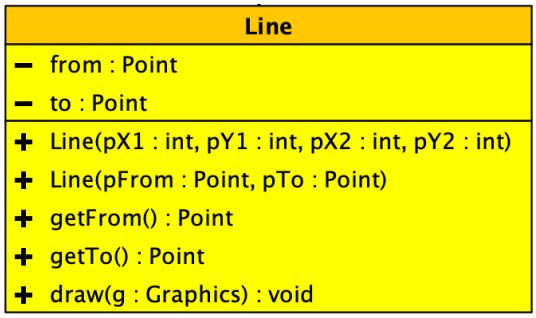


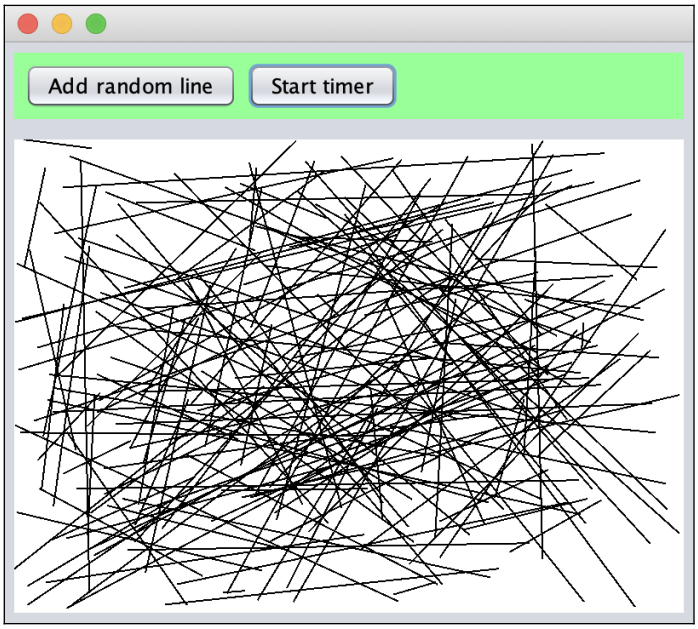
Exercice E.6: Lignes aléatoires

Développez un programme permettant de tracer automatiquement des lignes de manière aléatoire sur un canevas.

Principe de fonctionnement

Définissez une classe **`Line`** et une classe **`Lines`**. La classe **`Line`** est définie comme montré ci-dessous. La classe **`Lines`** possède une méthode **`addRandomLine(int width, int height)`** qui ajoute une ligne aléatoire qui se trouve entièrement dans la surface définie par **`width`** et **`height`**.





Remarques

- Le bouton **`stepButton`** permet d'ajouter une ligne aléatoire. Les extrémités de la nouvelle ligne doivent impérativement se trouver en dedans les limites du canevas.
- En ce qui concerne le bouton **`timerButton`** :
  - Affichez le texte « Start timer » si le chronomètre n'est pas actif et le texte « Stop timer » dans le cas contraire.
  - Le chronomètre s'exécute toutes les 10 millisecondes.

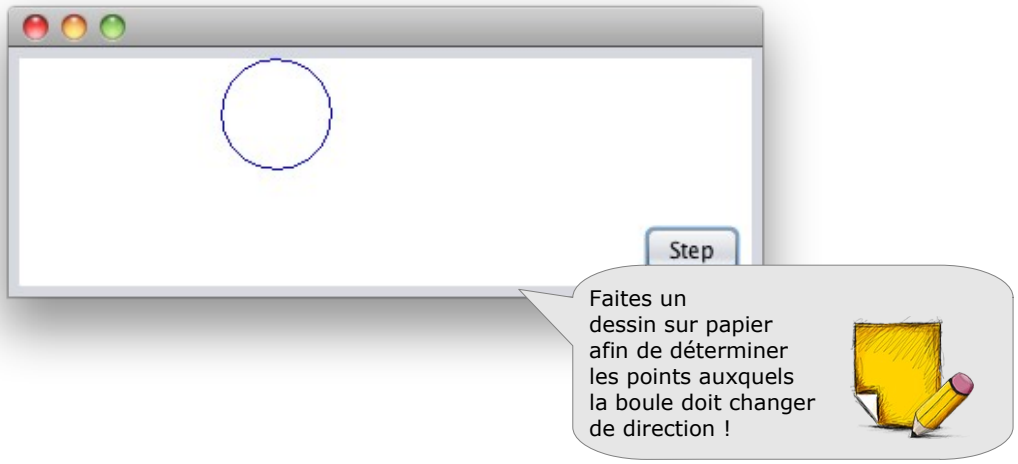
Pour les plus rapides

Faites en sorte que les lignes prennent des couleurs aléatoires.

Exercice E.7: Animation simple

Étape 1

Écrivez un programme qui fait bouger une boule du côté gauche vers le côté droit et vice-versa.



Remarques

- L'animation est lancée tout de suite lors du démarrage du programme.
- Pour tester, le bouton **`stepButton`** permet de déplacer la boule d'une étape lorsque le chronomètre n'est pas encore actif. Par après, ce bouton peut être rendu invisible à l'aide de la méthode **`setVisible(boolean)`**.
- La boule peut être représentée par la classe **`MovingBall`**, (x,y) étant les coordonnées du **centre** de la boule.
- La balle possède un attribut **`xStep`** qui définit le pas à effectuer en horizontale à chaque appel de **`doStep`**. Ensemble avec le délai du **`timer`**, cet attribut définit ainsi la vitesse horizontale de la balle à l'écran.

  `xStep` positif ↔ déplacement vers la droite;
  négatif ↔ déplacement vers la gauche.

  C'est la méthode **`doStep`** qui change le signe de **`xStep`** avant que la balle ne touche les bords à gauche ou à droite.

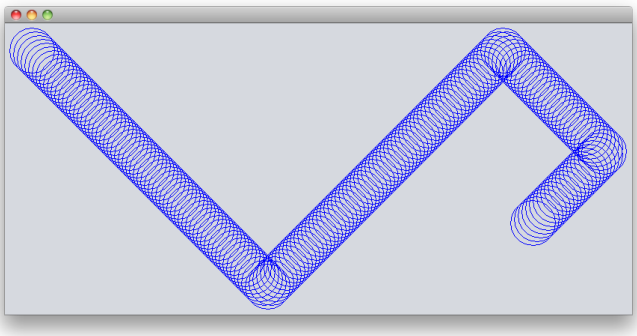
MovingBall
− x : int
− y : int
− radius : int
− xStep : int
+ getX() : int
+ getY() : int
+ getRadius() : int
+ getXStep() : int
+ doStep(width : int) : void
+ draw(g : Graphics) : void

Veillez à ce que la balle retourne automatiquement à l'intérieur du canevas lorsqu'on rétrécit brusquement le canevas.

Étape 2

Modifiez votre programme de manière à ce que la boule ne fasse pas uniquement un mouvement horizontal mais aussi un mouvement vertical !

Voici une capture d'écran truquée afin de vous permettre de mieux saisir le principe de fonctionnement de la modification à apporter au programme.



Exercice E.8: Animation : Plusieurs boules en mouvement

Etape 3:

Pour effectuer des déplacements plus précis et pour avoir plus de choix pour les trajectoires des balles, les coordonnées et les pas à effectuer seront mémorisés comme réels (**double**) et ce sera seulement lors du dessin qu'on les convertira en entiers.

Nous allons donc changer la classe **`MovingBall`** comme décrit ci-dessous. Nous ajoutons aussi un constructeur pour initialiser les attributs.

MovingBall
− x : double − y : double − radius : int − xStep : double − yStep : double
+ MovingBall(pX : double, pY : double, pRadius : int, pXStep : double, pYStep : double) + getXStep() : double + getYStep() : double + doStep(width : int, height : int) : void + getX() : double + getY() : double + getRadius() : int + draw(g : Graphics) : void

En résumé:

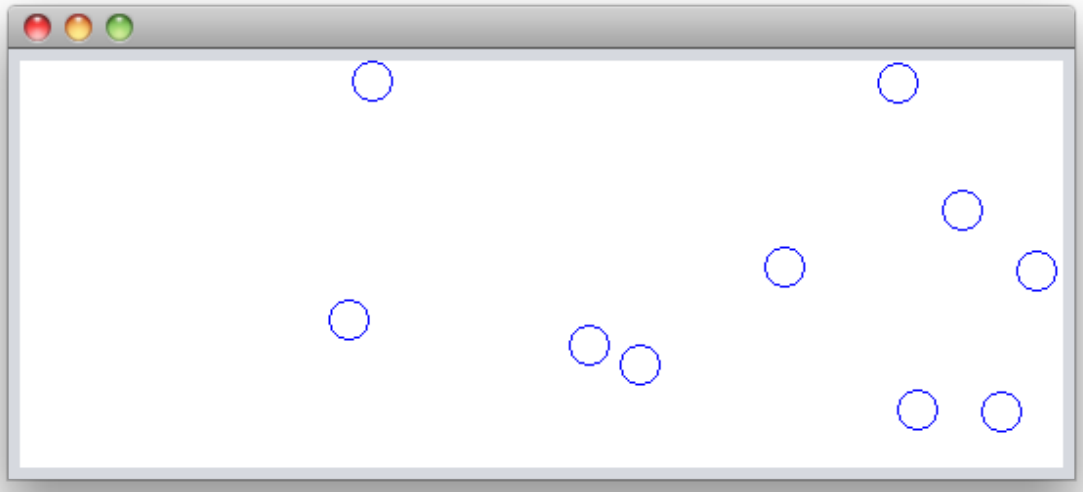
Cette classe représente une boule en mouvement qui sait se déplacer elle-même à l'aide de la méthode **`doStep(...)`**. En lui envoyant la largeur et la hauteur du canevas sur lequel elle se trouve, elle peut rebondir sur les bords sans jamais sortir du canevas.

Comme **`xStep`** et **`yStep`** sont des réels, les balles créées peuvent avoir une infinité de trajectoires différentes.

Dans la plupart de nos animations et simulations, nous allons veiller à ce que tous les calculs se font avec la meilleure précision possible. (C'est seulement pour le dessin où les valeurs sont arrondies sans quand même changer les coordonnées mémorisées.)

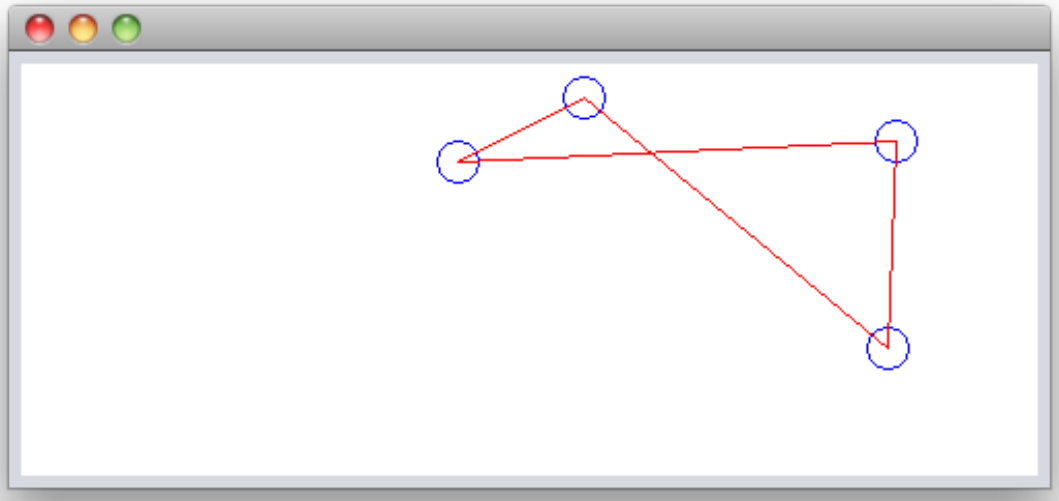
Modifiez maintenant votre programme de manière à ce qu'il devienne possible d'ajouter plusieurs boules en mouvement. Leurs positions de départ sont aléatoires. Leurs trajectoires sont aléatoires (`-5.0 ≤` **`xStep`** `≤ 5.0 et -5.0 ≤` **`yStep`** `≤ 5.0`).

La liste des balles est gérée par la classe **`MovingBalls`**.



Etape 4 :

Ensuite nous pouvons ajouter le code qui permet de tracer une ligne entre les différentes boules. Quelle classe faut-il charger de cette tâche ?



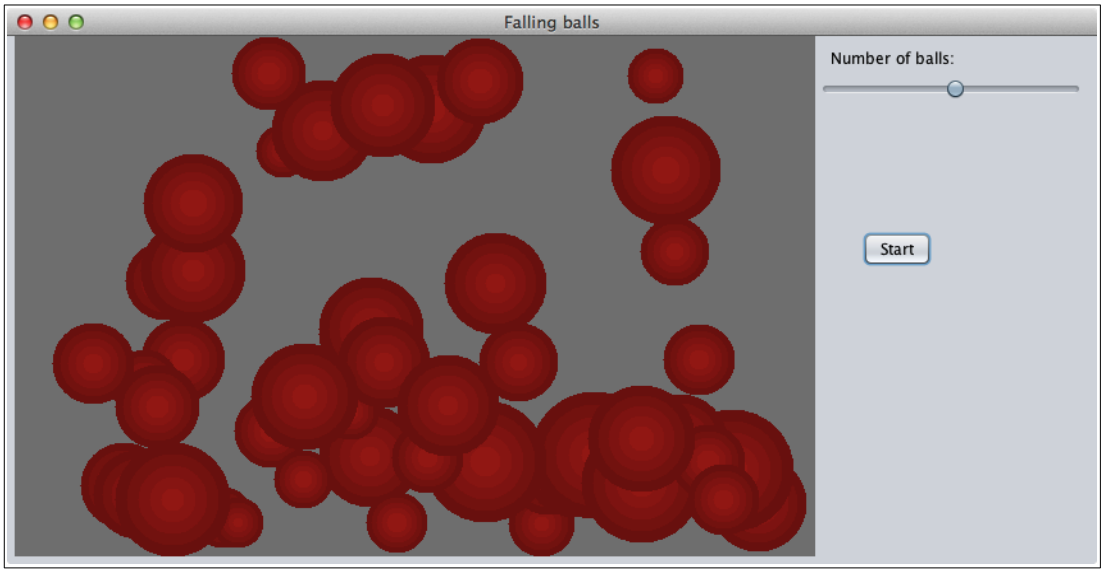
Réessayez avec une série de boules de rayon 0 !

Essayez aussi de modifier la taille de la fenêtre !

Notions requises : chronomètre, dessiner sur un canevas, listes

Exercice E.9: Balles tombantes

Il s'agit de réaliser une application pour simuler des balles tombantes qui rebondissent au sol. Réalisez le programme en suivant le diagramme UML que vous trouvez à la page suivante et en respectant les instructions et précisions données dans les remarques ci-dessous.

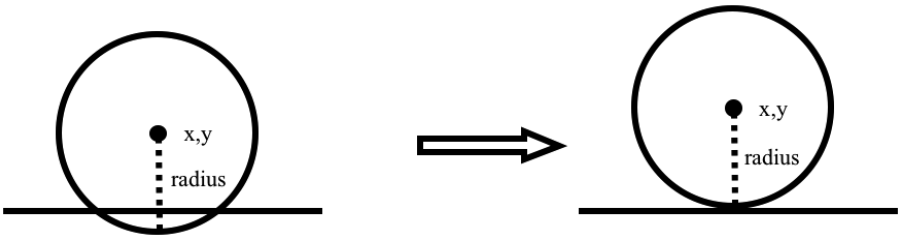


Classe **Ball :**

x, **y** : les coordonnées du centre de la balle.
dY : (delta y) définit le pas du déplacement en verticale, initialisé à 0.
friction : valeur utilisée lors d'un rebond de la balle au sol. **`friction`** est à initialiser à une valeur aléatoire réelle dans le domaine [0.65 , 0.80[.
falling : valeur booléenne initialisée à ***`false`*** qui indique si la balle est en train de tomber.
drop() : laisse tomber une balle en mettant son attribut **`falling`** à ***`true`***.
move(...) : réalise le déplacement vertical d'une balle tombante.

Pour cela :

- ***`dY`*** est incrémenté de *g* (*gravité*), valeur fixée à 0.981.
- ***`y`*** est incrémenté de ***`dY`***.
- si la balle déborde la hauteur ***`pHeight`***, alors remettez la balle sur ***`pHeight`*** et multipliez ***`dY`*** par **`(-1)*`*`friction`**.



Classe **Balls :**

random(...) : retourne un nombre aléatoire entier compris entre [**`pMin`** … **`pMax`**].

Balls(...) : sert à créer **`pN`** balles et à les ajouter à la liste **`alBalls`**. Toutes les balles doivent se trouver entièrement à l'intérieur du plan délimité par les paramètres **`pWidth`** et **`pHeight`**. Le rayon d'une balle est une valeur aléatoire de [20 , 50].

dropBall(...) : laisse tomber la première balle de la liste **`alBalls`** qui n'est pas encore en train de tomber.

move(...) : déplace toutes les balles de la liste **`alBalls`**.

Classe **MainFrame :**

La classe contient deux chronomètres :

- le chronomètre **`timerDrop`** sert à laisser tomber une balle toutes les 90 millisecondes. Le bouton y lié **`dropButton`** est invisible.
- le chronomètre **`timerMove`** sert à réaliser le déplacement vertical des balles toutes les 40 millisecondes. Le bouton y lié **`stepButton`** est invisible.

Lors d'un clic sur le bouton **`startButton`**, les balles sont créées dans le premier quart en haut de la surface de dessin disponible. Le nombre de balles peut être configuré à l'aide de la glissière.

Notions requises : chronomètre, dessiner sur un canevas, listes