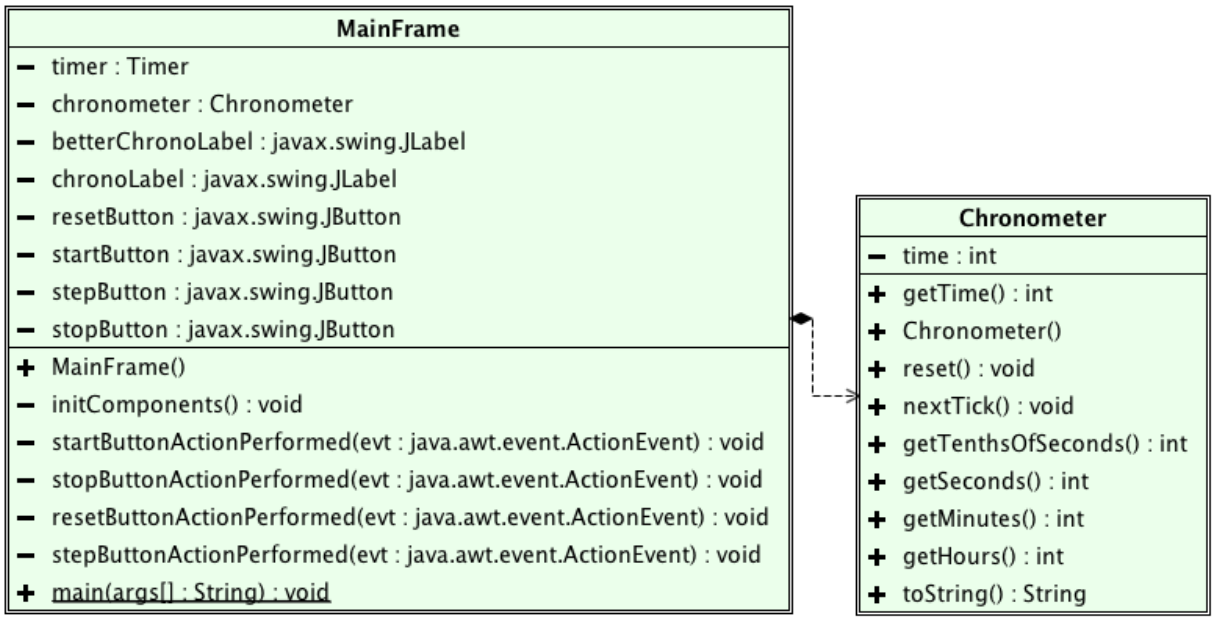
Série F : Le chronomètre

Table des matières

Exercice F.1: Chronomètre simple

Dans la suite, vous allez développer un chronomètre (*Stoppuhr*). Lors de cette première étape, vous allez développer un chronomètre qui affiche le temps de manière textuelle.



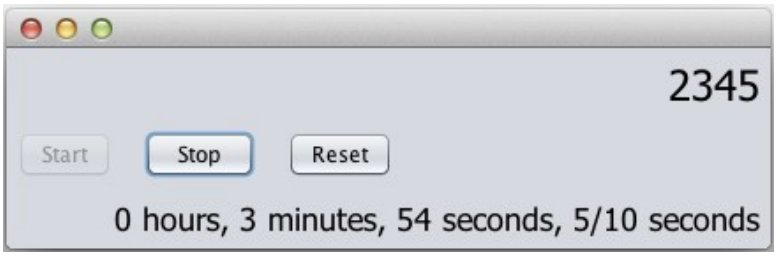
Voici quelques explications supplémentaires:

La classe **`Chronometer`** implémente un chronomètre:

- L'attribut **`time`** représente le temps écoulé en dixièmes de secondes; **`getTime`** est son accesseur.
- Les méthodes **`getHours`**, **`getMinutes`**, **`getSeconds`** et **`getTenthsOfSeconds`** calculent, à partir de la valeur de **`time`**, respectivement la durée en heures, minutes, secondes et dixièmes de secondes.
  Par exemple, pour représenter une durée de 3 minutes, 54 secondes et 5 dixièmes de secondes **`time`** vaut 2345. Dans ce cas **`getMinutes`** retourne 3, **`getSeconds`** retourne 54 et **`getTenthsOfSeconds`** retourne 5.
- La méthode **`nextTick`** fait avancer le temps d'une dixième de seconde.
- La méthode **`toString`** retourne le temps écoulé sous forme textuelle comme représenté sur la capture d'écran ci-dessous.

L'interface graphique de l'application se présente de la manière suivante:

Veillez à ce que les boutons soient désactivés s'ils sont inutiles (p.ex. Lorsque le chronomètre est en arrêt, le bouton 'Stop' est désactivé.)



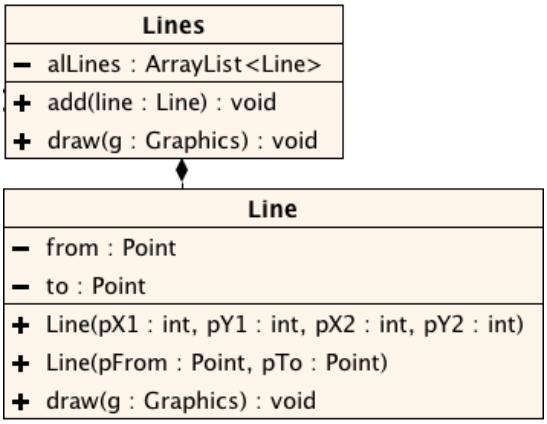
Notions requises : chronomètre, division entière avec reste

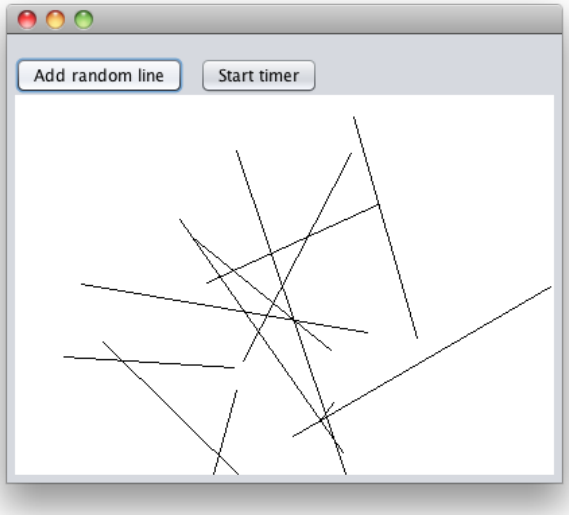
Exercice F.2: Lignes aléatoires

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

Principe de fonctionnement

La fiche principale **`MainFrame`** possède une liste de lignes qu'elle transmet au panneau de dessin **`drawPanel`** (instance de la classe **`DrawPanel`**). La classe **`DrawPanel`** affiche cette liste de lignes sur le canevas à l'intérieur de la méthode **`paintComponent(Graphics g)`**. Une liste de lignes est représentée par la classe **`Lines`**, une ligne par la classe **`Line`**.





Remarques

- Le bouton **`stepButton`** permet d'ajouter une ligne aléatoire. Les extrémités de la nouvelle ligne doivent impérativement se trouver à l'intérieur des limites du canevas.
- En ce qui concerne le bouton **`timerButton`** :
  - Affiche 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

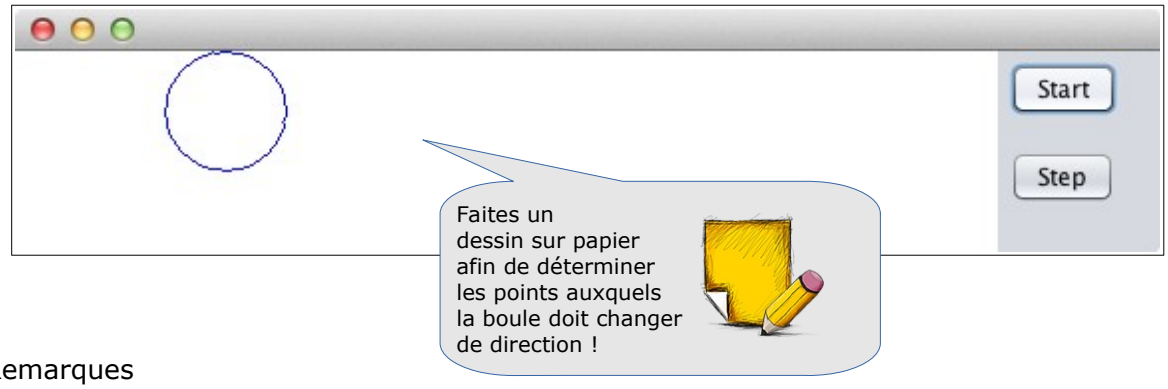
1. Faites en sorte que les lignes prennent des couleurs aléatoires. Tuyau: Utilisez la classe **`Color`**.
2. Modifiez votre code de manière à ce que les lignes prennent des couleurs aléatoires parmi la liste des couleurs suivantes : rouge, vert, bleu, jaune, noir et rose.

Notions requises : chronomètre, dessiner sur un canevas

Exercice F.3: Animation simple - Une boule en mouvement

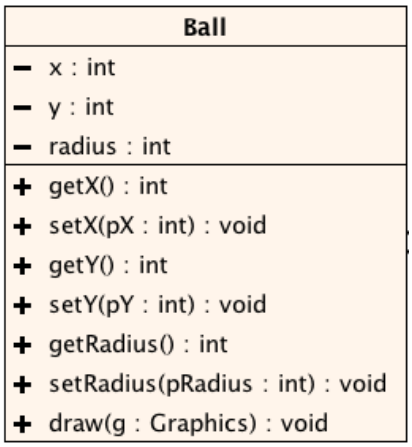
Étape 1

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



Remarques

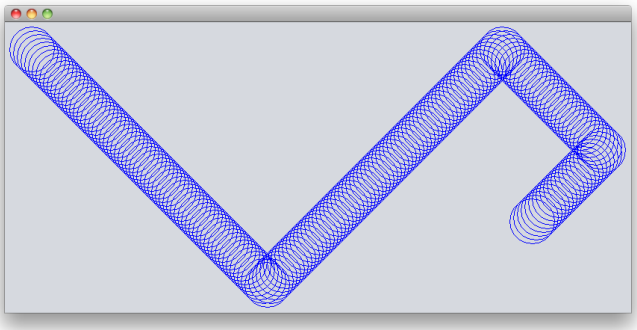
- Pour tester, le bouton **`stepButton`** permet de déplacer la boule d'une étape lorsque le chronomètre n'est pas encore actif. Ceci nous permet de vérifier exactement les mouvements de la boule. Plus tard, il peut être rendu invisible à l'aide de la méthode **`setVisible(boolean)`**.
- L'animation peut être lancée et arrêtée par le bouton **`startStopButton`**.
- La boule peut être représentée par la classe suivante, (x,y) étant les coordonnées du **centre** de la boule :



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.



Question à discuter:

Que peut-on dire de ce programme du point de vue encapsulation et modélisation MVC?

Notions requises : chronomètre, dessiner sur un canevas

Exercice F.4: Animation - Plusieurs boules en mouvement

Étape 1: Améliorations

- En fait, du point de vue encapsulation, il vaudrait mieux disposer d'un constructeur pour initialiser les coordonnées et de supprimer les manipulateurs.
- D'autre part, du point de vue modélisation ce serait mieux de disposer d'une boule qui sache
  - se déplacer soi-même avec ses propres attributs,
  - se dessiner soi-même.
- Pour effectuer des déplacements plus précis, les coordonnées seront mémorisées comme réels (double) et ce sera seulement lors du dessin qu'on les convertira en entiers.

Nous allons donc changer la classe **`Ball`** comme décrit ci-dessous et créer une nouvelle classe **`MovingBall`** qui saura se déplacer et se dessiner soi-même.

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

Cette nouvelle classe représente donc une boule en mouvement. Elle 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.

La balle possède deux attributs **`xStep`** et **`yStep`** (type **`double`**) qui définissent les pas à effectuer en horizontale et en verticale à chaque appel de **`doStep`**. Ensemble avec le délai du **`timer`**, ces deux attributs définissent ainsi les vitesses horizontale et verticale des balles à l'écran.

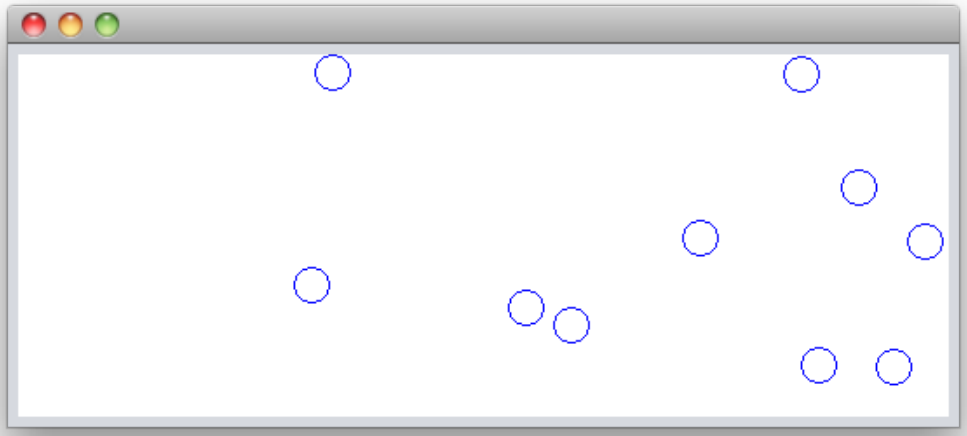
`xStep`	positif ↔ déplacement vers la droite;	négatif ↔ vers la gauche
`yStep`	positif ↔ déplacement vers le bas;	négatif ↔ vers le haut

Comme **`xStep`** et **`yStep`** sont des réels, les balles peuvent avoir une infinité de trajectoires différentes. Désormais, afin de disposer d'une meilleure précision, les coordonnées de la boule seront aussi des nombres réels.

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

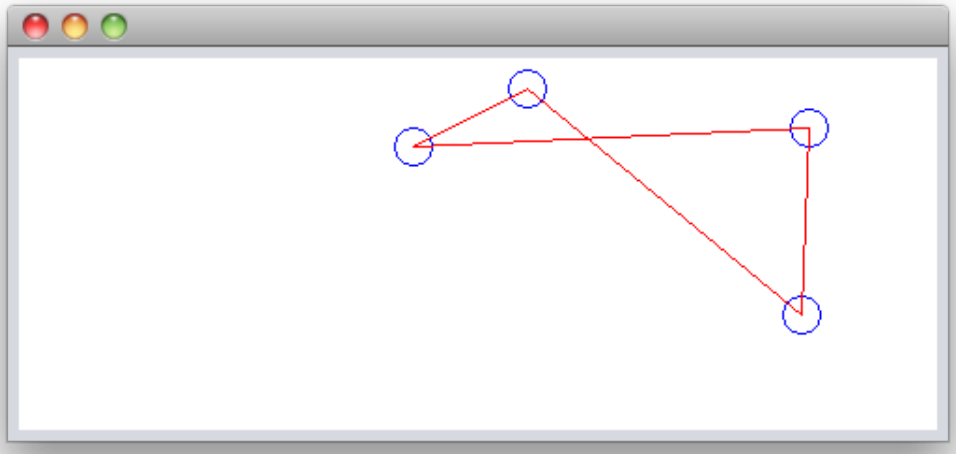
Étape 2

Modifiez ensuite votre programme de manière à ce qu'il devienne possible d'ajouter plusieurs boules en mouvement, p.ex. 30 boules de rayon 40 avec 33 déplacements /seconde. 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`**.



Étape 3

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 !

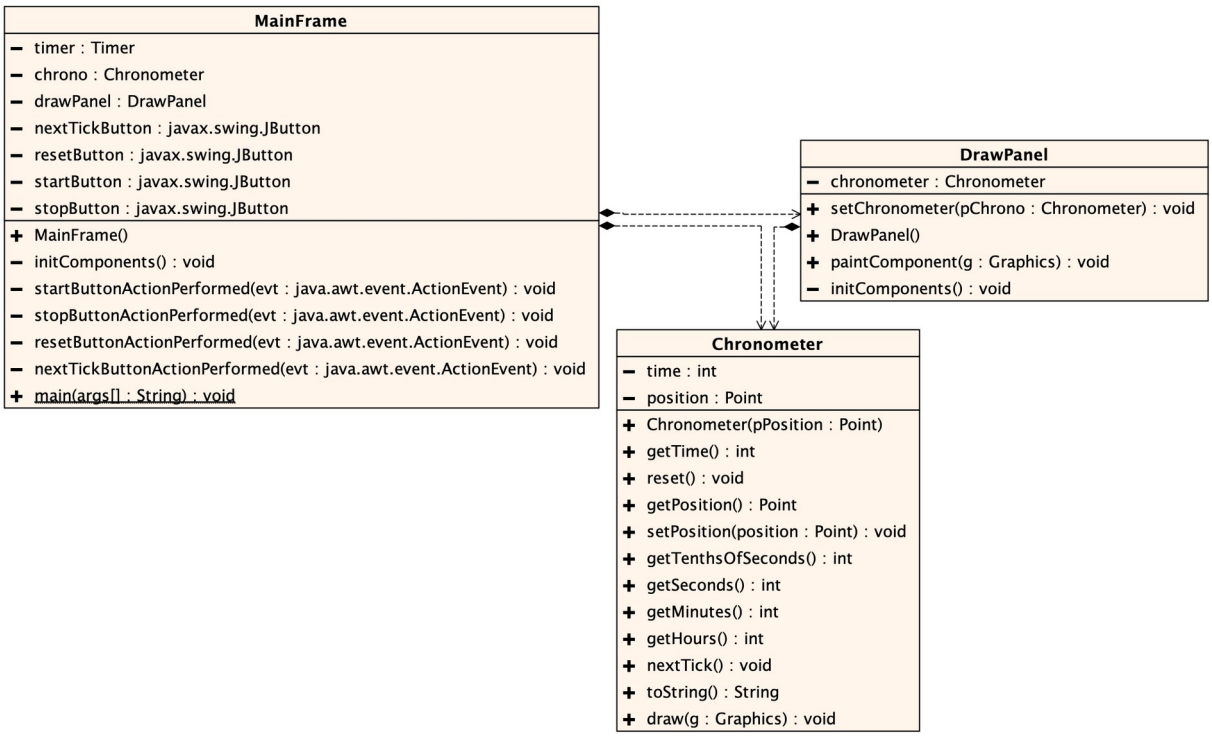
Essayez aussi de modifier la taille de la fenêtre !

Notions requises : chronomètre, dessiner sur un canevas, listes

Exercice F.5: Chronomètre

Reprenez l'application qui implémente un chronomètre (*Stoppuhr*). Modifiez le programme de façon à ce que l'heure puisse être affichée sous forme de texte à une position donnée sur un canevas.

Voici le schéma UML simplifié de l'application (uniquement les attributs et les méthodes les plus importantes sont représentées).



Voici quelques explications supplémentaires:

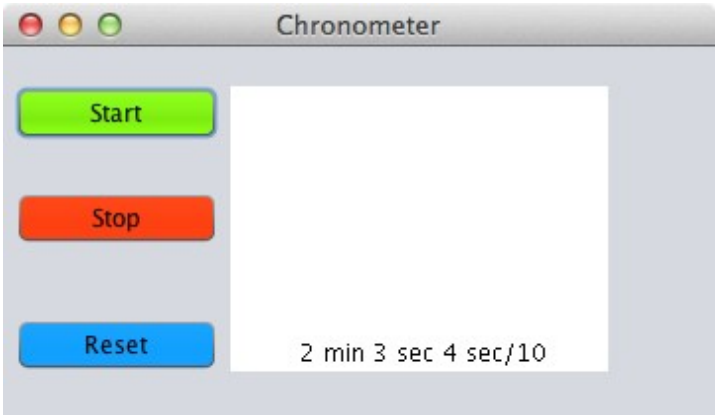
La classe **`DrawPanel`** permet d'afficher un chronomètre sur un panneau:

- L'attribut **`chronometer`** contient le chronomètre à afficher.
- La méthode **`paintComponent`** affiche le chronomètre sur le panneau.

La classe **`Chronometer`** implémente un chronomètre:

- L'attribut **`position`** représente sa position sur un canevas.
- La méthode **`draw`** affiche sur le canevas de dessin le temps écoulé sous forme textuelle, comme représenté sur la capture d'écran ci-dessous.

L'interface graphique de l'application se présente de la manière suivante:

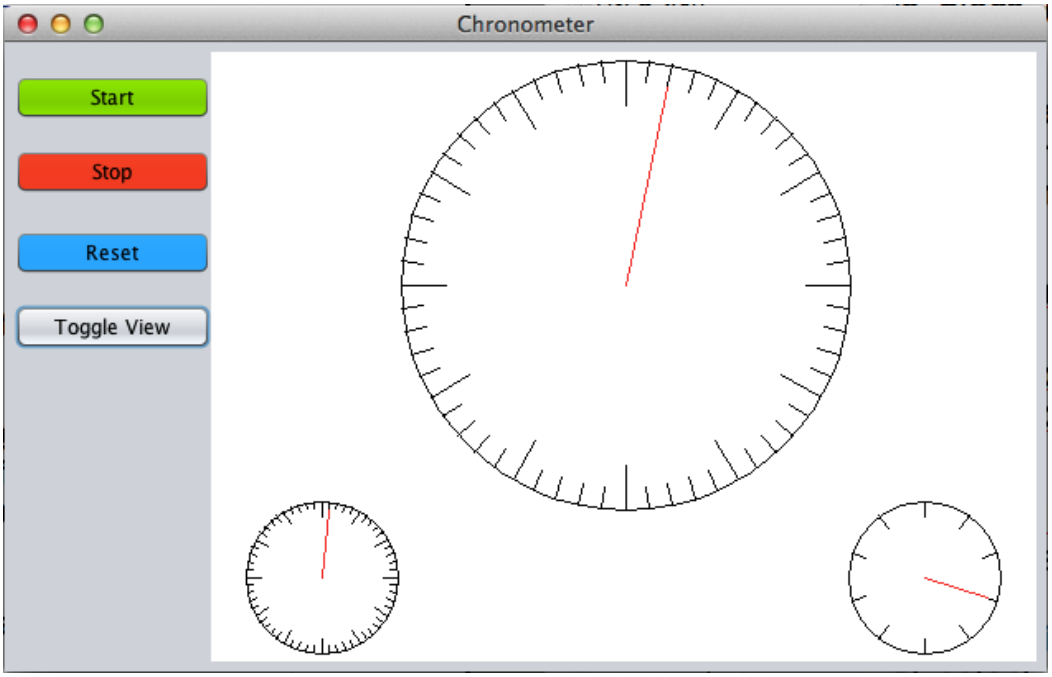


Notions requises : chronomètre, dessiner sur un canevas, listes

Exercice F.6: Chronomètre graphique (pour avancés)

Lors de cette étape vous aller modifier la classe **`Chronometer`** de façon à ce qu'elle puisse afficher le chronomètre aussi sous forme analogique.

Voici l'interface utilisateur:



Un bouton supplémentaire permet de basculer (*"toggle"*) entre un affichage analogique et un affichage sous forme textuelle.

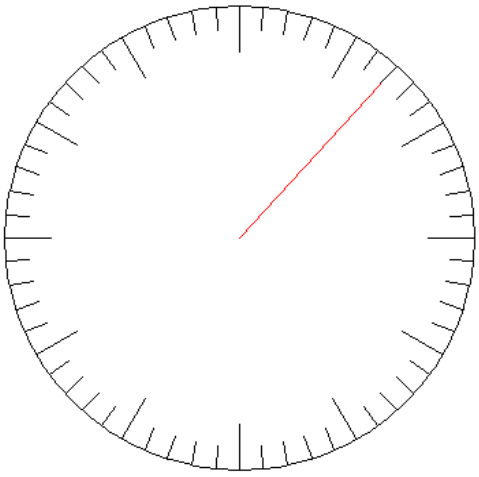
La classe **`Chronometer`** contient maintenant un attribut booléen **`analogView`** qui est **true** si le cadran doit être affiché sous forme analogique, **false** pour l'affichage digital. La méthode **`toggleView`** permet de basculer entre les deux modes d'affichage.

Le chronomètre est affiché à l'aide de trois cadrans (EN : *dial* : un grand cadran (de taille ¾ du canevas de dessin) pour les secondes et deux petits (de taille ¼ du canevas de dessin) pour les minutes et les dixièmes de secondes.

La classe **`Dial`** implémente un cadran:

- Les attributs **`x`**, **`y`**, **`radius`** définissent l'emplacement et le rayon du cadran sur le canevas de dessin.
- L'attribut **`numberOfTicks`** définit le nombre de valeurs affichées sur le cadran sous forme de graduations.
- L'attribut **`majorTick`** définit la fréquence des graduations « majeures » (représentées à l'aide d'un trait plus long)
- L'attribut **`currentTick`** définit la valeur affichée par l'aiguille. Cet attribut peut être accédé à l'aide des accesseurs **`getCurrentTick`** et **`setCurrentTick`**.

- Exemple:

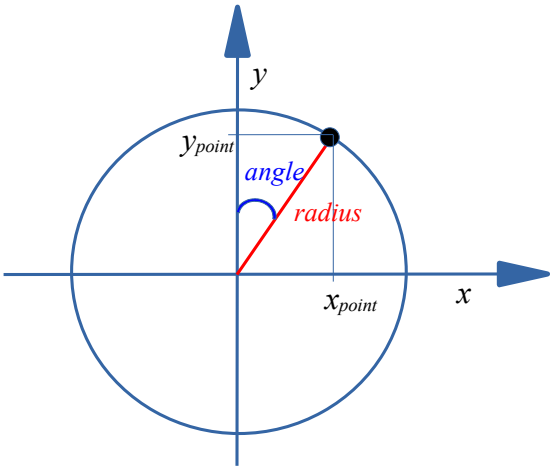


numberOfTicks = 60
majorTick = 5
currentTick = 7

- La méthode **draw** affiche le cadran sur le canevas de dessin. Les graduations sont représentées par un trait dont la longueur est de un dixième du rayon du cadran. Pour les graduations majeures, la longueur est un cinquième du rayon. Pour dessiner les graduations et l'aiguille, il est utile de définir et d'utiliser la classe **PolarVector**.

La classe **PolarVector** représente un vecteur. Lors de la construction, on indique le point de départ, son angle et son rayon. Après, on a accès aux coordonnées du point d'arrivée (**getEndX**, **getEndY**).

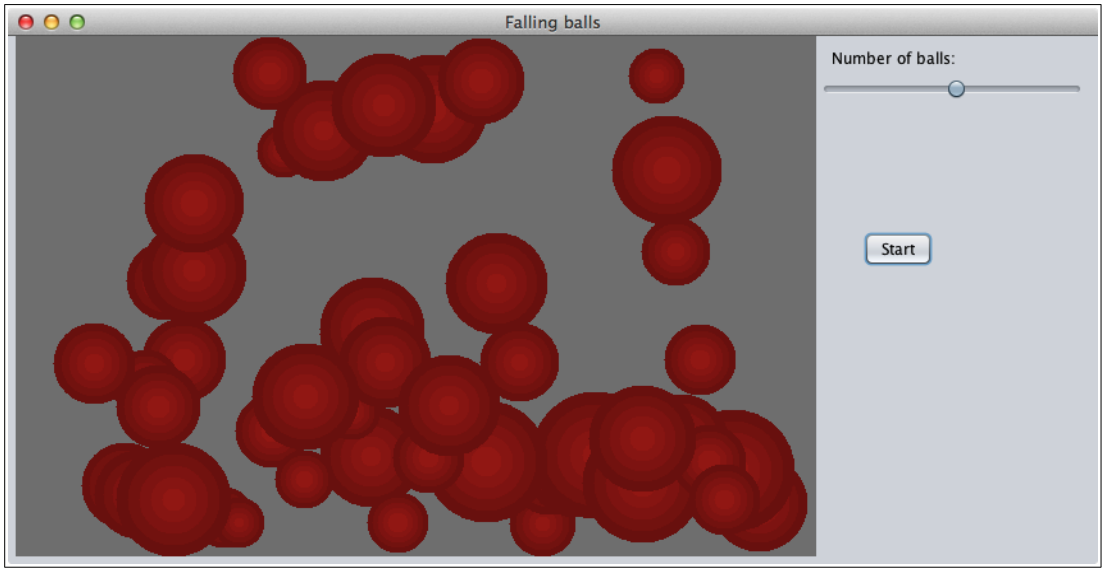
*Attention, lors du calcul de **getEndY**, tenez compte du fait que l'axe des y est inversé.*



Notions requises : dessin, timer, application simple de fonctions trigonométriques

Exercice F.7: 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 dans le domaine [**`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.