Série G : La programmation orientée objet

Table des matières

Exercice G.1: Exercice de réflexion

On a les classes suivantes :

- **`Teacher`** (enseignant)
- **`Student`** (élève, étudiant)
- **`Person`** (personne)
- **`Employee`** (employé / DE: Angestellter)
- **`Client`** (patient)
- **`Doctor`** (docteur)

Effectuez les tâches suivantes :

1. Créez un nouveau projet Unimozer avec toutes ces classes.
2. Utilisez l'outil d'héritage (en haut à gauche ) pour faire hériter les classes les unes des autres. Réfléchissez bien quelle classe est une spécialisation de quelle autre !
3. Ajoutez maintenant les classes **School** et **Hospital** avec les listes qui s'imposent. Quels types de relation avez-vous créés ?
4. Ajoutez maintenant des attributs aux différentes classes. Réfléchissez bien quelle classe aura besoin de quels attributs ! Rappel : Une classe fille hérite de tous les attributs de la classe mère.
5. Que constatez-vous en comparant les classes **`Teacher`** et **`Student`** ?
6. A l'exception des accesseurs et manipulateurs, de quelles méthodes pourrait-on pourvoir ces classes ?

Notions requises : types de relations en POO

Exercice G.2: Animation : Différents objets en mouvement

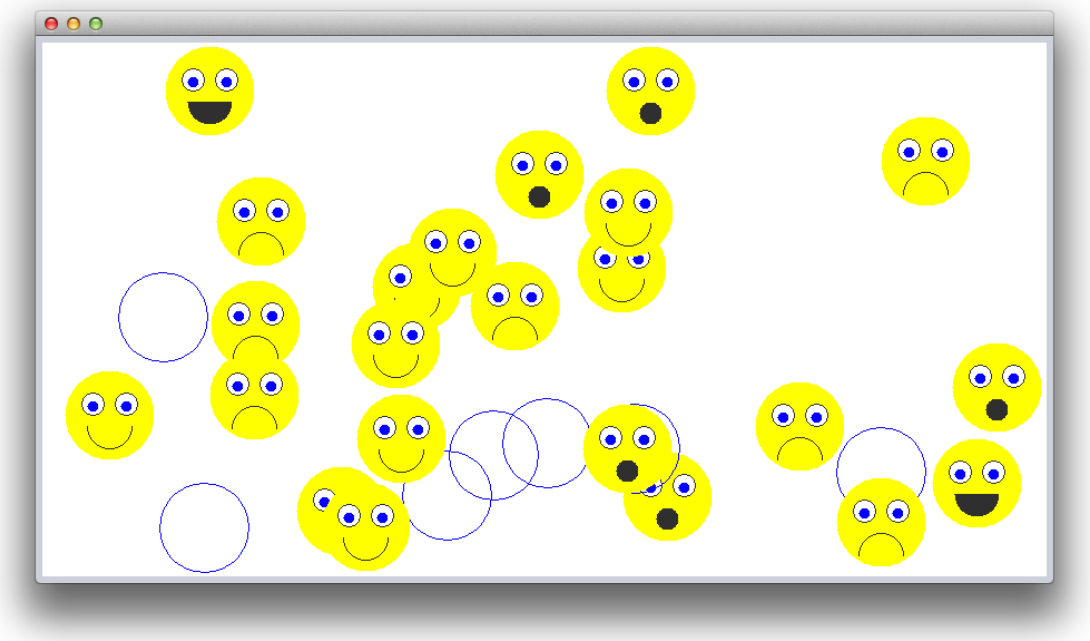
Reprenez l'exercice F4 (*Animation : Boules en mouvement*), puis modifiez-le en suivant les étapes que voici :

Étape 4

Dérivez de **`MovingBall`** les classes **`EmoSmile`**, **`EmoSad`**, **`EmoBigSmile`**, **`EmoSurprised`** qui représentent des *émoticônes* (Smileys) avec différentes expressions. Tous les émoticônes ont la même couleur de fond (jaune) et les mêmes yeux. (Soyez créatifs ! ;-)

Remarque : Consultez l'aide sur les méthodes **`drawArc`** et **`fillArc`**.

Testez votre programme en produisant aléatoirement des balles et différents types d'émoticônes. Profitez au mieux des connaissances que vous avec de l'OOP.



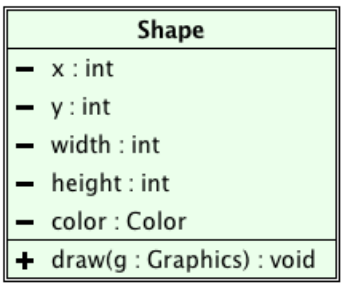
Pour avancés: Extensions possibles :

- Ajoutez **`EmoNaughty`** qui tire sa langue de temps en temps
- Modifiez **`EmoSad`** pour qu'ils disparaissent après un temps fixé lors de la création.

Notions requises : héritage, chronomètre, surcharge d'une méthode, encapsulation

Exercice G.3: Figures géométriques

Considérons la classe **Shape** (pour faciliter la lecture, les accesseurs, modificateurs et constructeurs ont été omis) :



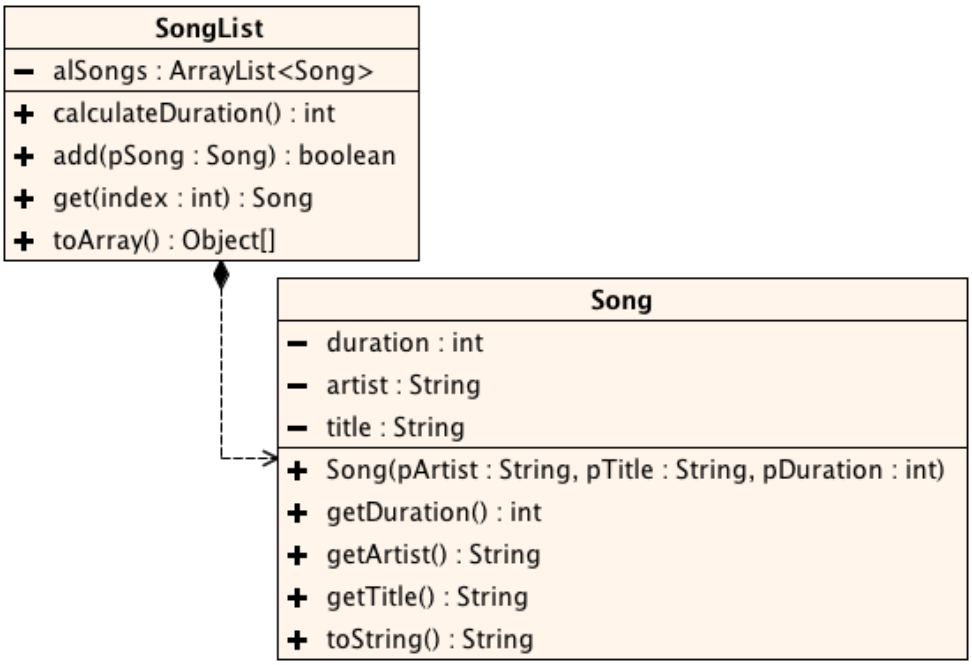
1. Créez un nouveau projet et implémentez la classe **Shape** avec tous les accesseurs et constructeurs nécessaires ! x et y sont les coordonnées du point supérieur gauche de la figure.
2. Ajoutez à votre projet les classes suivantes :
   - **`Rectangle`**
   - **`Ellipse`**
   - **`Triangle`** (Triangle isocèle)
3. Développez une classe **`MainFrame`** ainsi qu'une classe **`DrawPanel`.** Analysez la déclaration (c'est la ligne qui commence par « **`public class`** ») de votre classe **`DrawPanel`**. Que constatez-vous ? Que pouvez-vous dire de la méthode **`paintComponent`** ?
4. Ajoutez la possibilité de tracer les figures géométriques à l'aide de la souris. Par exemple :
   - tracez des rectangles avec le bouton gauche de la souris,
   - tracez des triangles isocèles avec le bouton central de la souris,
   - tracez des ellipses avec le bouton droit de la souris.

   Les figures ne sont visibles qu'après que le bouton de la souris n'a été relâché. Veillez à ce que les figures se laissent dessiner aussi en commençant par leurs points inférieurs ou par leur point supérieur droit.
5. Faites une copie du projet et ajoutez dans la copie les classes **`Square`** et **`Circle`**. Est-ce que la classe **Shape** est toujours adaptée pour en dériver ces classes ? Modifiez et améliorez les classes en conséquence. Intercalez les classes nécessaires dans la hiérarchie de l'héritage. Ouvrez le projet en Unimozer et arrangez les diagrammes UML pour représenter clairement l'héritage entre vos classes.
6. Lors du dessin les carrés et les cercles doivent se trouver à l'intérieur du rectangle tracé par la souris. Tracez des cercles au lieu d'ellipses et des carrés au lieu de rectangles, si la touche '**Shift**' est enfoncée (→ voir **`MouseEvent`**).

Notions requises : héritage, surcharge de méthodes
Classes requises : JPanel, Graphics, Color

Exercice G.4: Liste de chansons

Réalisez d'abord les classes **Song** et **SongList** décrites par le schéma UML ci-dessous :



Remarques :

- ***duration*** est indiqué en secondes*.*
- ***calculateDuration*** retourne la durée totale des chansons dans la liste (en secondes).
- ***add*** ajoute uniquement les chansons qui ne se trouvent pas encore dans la liste. La méthode retourne ***true*** ssi la chanson a été ajoutée avec succès.

Ajoutez la classe **LimitedPlayList** comme spécialisation de la classe **SongList.** Il s'agit d'une classe qui permet de compiler une liste de chansons (EN : *playlist*) pour un CD Audio, une clé USB ou un événement d'une durée déterminée :

- La classe **LimitedPlayList** doit posséder un attribut **capacity** indiquant sa capacité dont la valeur par défaut est 74*60 (74 minutes = 74*60=4440 secondes). **capacity** possède aussi un accesseur.
- La classe **LimitedPlayList** possède un constructeur permettant spécifier la capacité d'une **LimitedPlayList** en secondes.
- Ajoutez un deuxième constructeur permettant spécifier la capacité d'une **LimitedPlayList** en minutes (type **double**).
- Il n'est pas possible d'ajouter une chanson à une **LimitedPlayList** s'il n'y a plus assez de capacité libre.
- La classe **LimitedPlayList** possède une méthode **getFreeCapacity** qui indique la durée restante d'une **LimitedPlayList** (en secondes).

Interface graphique :

- Ajoutez une interface graphique qui affiche une liste de chansons dans une **JList** et qui permet d'ajouter des chansons à la liste.

- Ajoutez une deuxième **`JList`** pour représenter le contenu d'une **`LimitedPlayList`**. Les chansons d'une **`LimitedPlayList`** peuvent être ajoutées à partir de la liste de chansons.
- Ajoutez les objets et composants nécessaires (boutons, champs texte, libellés) pour pouvoir gérer les deux listes et pour pouvoir composer une **`LimitedPlayList`** à partir de chansons de la liste de toutes les chansons.
- Affichez la durée totale et le nombre de chansons de la liste des chansons ainsi que d'une **`LimitedPlayList`**. Indiquez aussi la capacité libre d'une **`LimitedPlayList`**.

Fonctions supplémentaires (révision des algorithmes de 2e) :

- Ajoutez la possibilité de trier la liste et/ou la **`LimitedPlayList`** par ordre alphabétique des artistes et des titres (=> critère de tri : concaténation de **`artist`** et **`title`**). Utilisez l'algorithme du tri par sélection.
- Ajoutez la possibilité de rechercher et de faire sélectionner (automatiquement) la chanson la plus longue de la liste.
- Ajoutez la possibilité de créer automatiquement une (nouvelle) **`LimitedPlayList`** avec toutes les chansons d'un artiste, jusqu'à ce que la capacité d'une **`LimitedPlayList`** ne suffise plus pour ajouter la prochaine chanson.

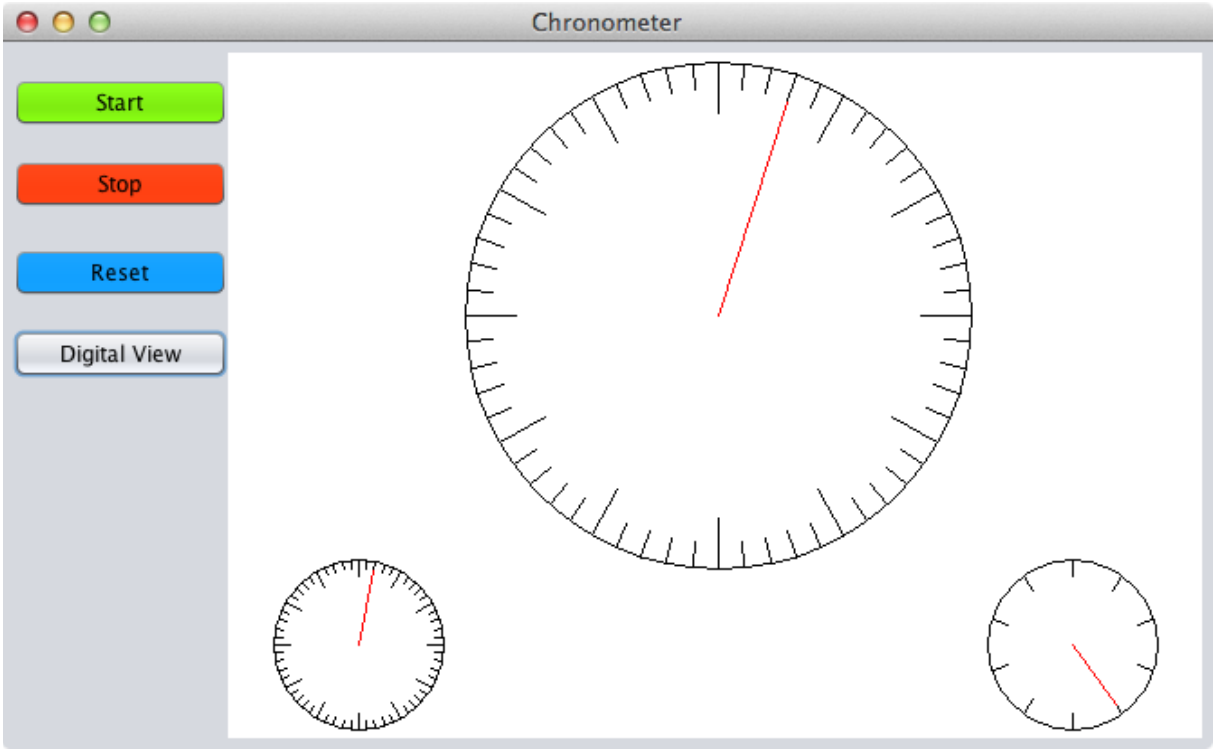
Notions requises : héritage, surcharge de méthodes
Classes requises : ArrayList

Exercice G.5: Chronomètre (suite de l'exercice F.6)

Faites un copie de l'exercice F.6, puis achevez les tâches suivantes :

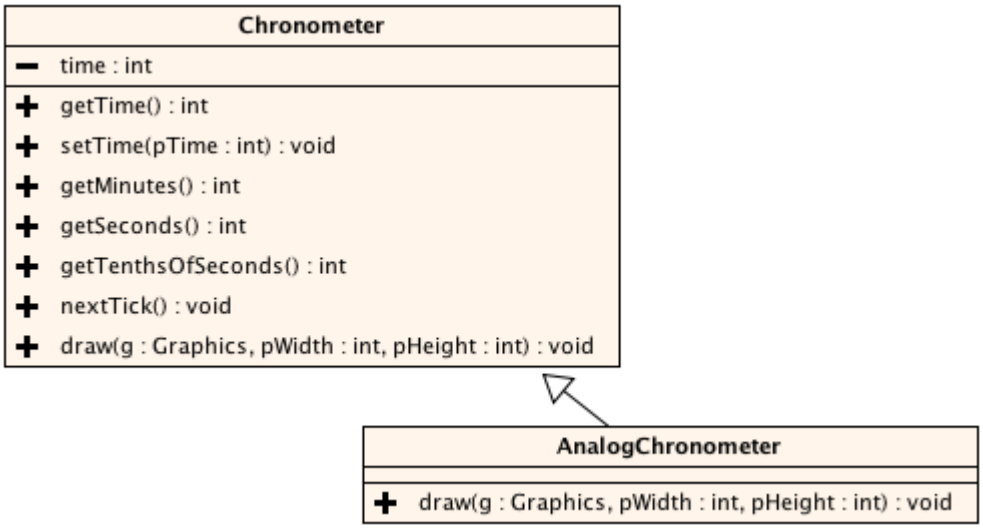
Vous aller maintenant dériver la classe **`AnalogChronometer`** de la classe **`Chronometer`** afin de pouvoir afficher le chronomètre sous forme analogique.

Voici l'interface utilisateur :



Un bouton supplémentaire permet de basculer entre un affichage analogique (« Analog View ») et un affichage sous forme textuelle (« Digital View »). A chaque clic l'intitulé du bouton change.

Voici le schéma UML de cette nouvelle classe :

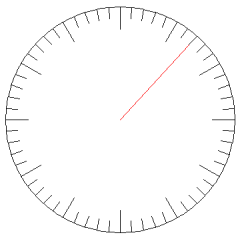


La classe **`AnalogChronometer`** surcharge la méthode **`draw`** pour afficher le chronomètre sous forme analogique. Le chronomètre est affiché à l'aide de trois cadrans: un grand cadran (de taille ¾ du canevas de dessin) pour les secondes et deux petits (de taille ¼ du canevas de dessin) pour les minutes et les dixièmes de secondes.

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

- Les attributs **`x`**, **`y`**, **`width`** et **`height`** définissent l'emplacement et la taille 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. Comme l'emplacement du cadran peut être de forme rectangulaire, le cadran est toujours dessiné au milieu. 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 d'utiliser la classe **`PolarPoint`**.

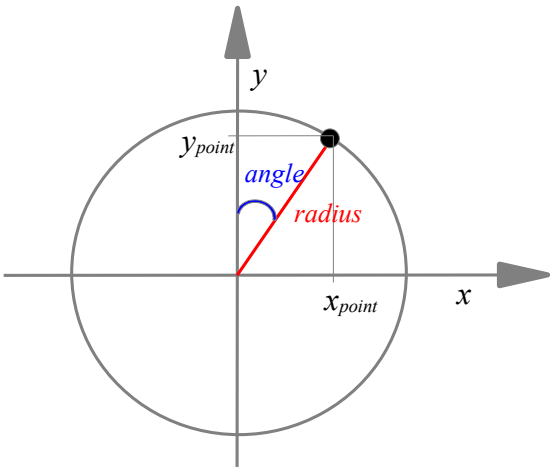
La classe **`PolarPoint`** est derivée de la classe **`Point`** et représente un point sur un cercle trigonométrique. Le constructeur de la classe **`PolarPoint`** surcharge celui de la classe **`Point`**.

Dans le constructeur on spécifie comme paramètres les coordonnées polaires, à savoir l'angle et le rayon (*radius*) du point.

Les coordonnées cartésiennes x_{point}, y_{point} se calculent alors de la manière suivante:

$x_{point} = radius \cdot sin(angle)$

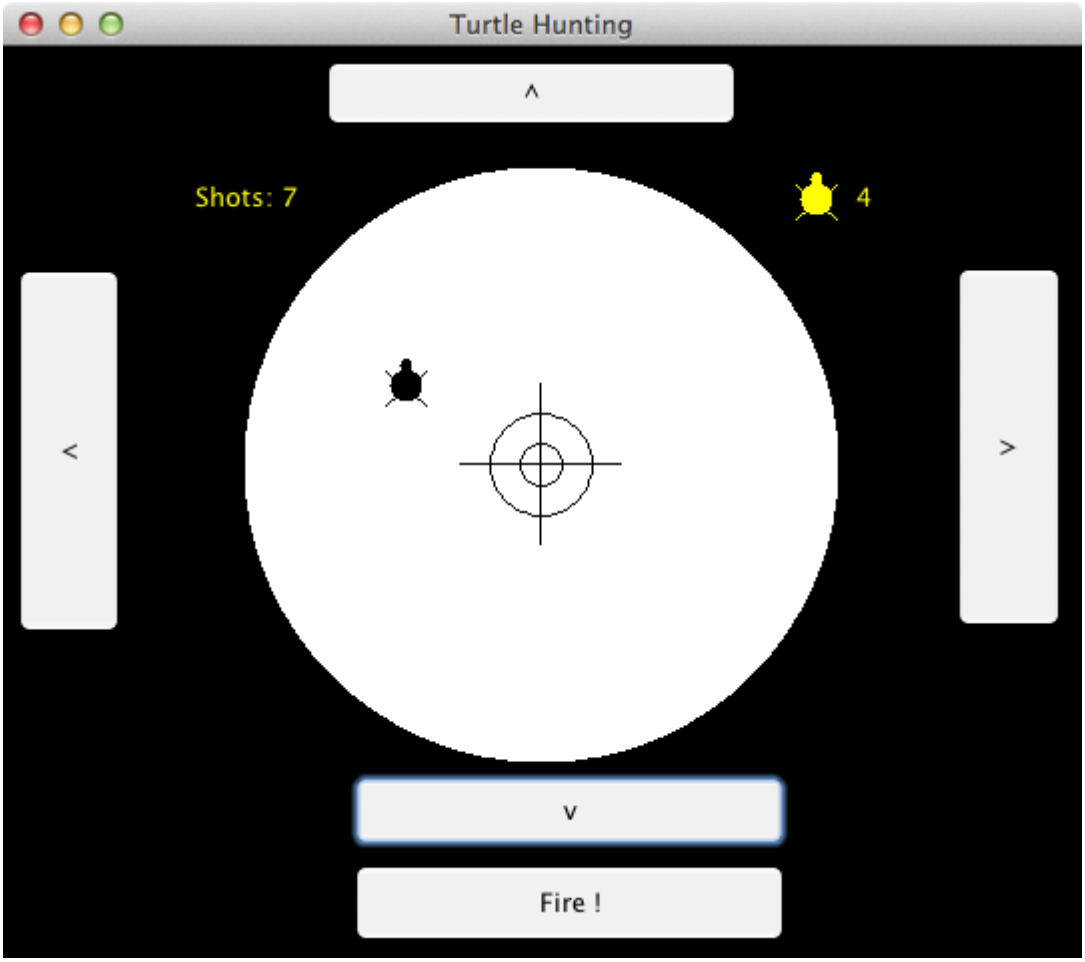
$y_{point} = radius \cdot cos(angle)$



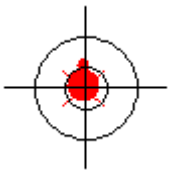
Notions requises : héritage, surcharge d'une méthode, encapsulation

Exercice G.6: Turtle hunting

Vous allez développer un petit jeu qui permet d'abattre des tortues en visant à l'aide d'un réticule (*allemand: Fadenkreuz*, *anglais: crosshair*).



Pour implémenter la tortue et le réticule vous allez développer les classes **`Turtle`** et **`Crosshair`** selon le schéma UML ci-après.



Mais avant de commencer ces développements il faut résoudre le problème suivant :

Comment vérifier que la tortue est touchée lors d'un tir ?

Pour cela vous pouvez utiliser la classe **`Rectangle`** qui fait partie de la bibliothèque de classes standards de Java.

La classe **`Rectangle`** (contenue dans le paquet **`java.awt`**) représente un espace rectangulaire et dispose des attributs suivants :

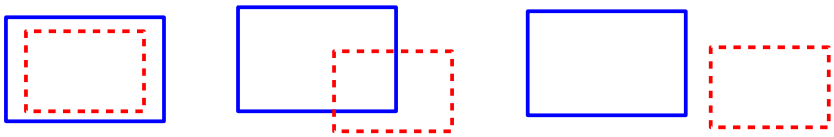
- **`x`** coordonnées x du coin supérieur-gauche.
- **`y`** coordonnées y du coin supérieur-gauche.
- **`width`** largeur du rectangle
- **`height`** hauteur du rectangle

A noter que ces attributs sont de visibilité publique.

Comme pour les composants visuels de Java, les attributs d'un objet de la classe **`Rectangle`** peuvent être accédés et modifiés par les méthodes: **`getX`**, **`getY`**, **`getWidth`**, **`getHeight`** et **`setLocation`**.

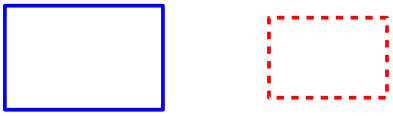
Les méthodes **`contains(int x, int y)`** et **`contains(Point p)`** permettent de vérifier si le point spécifié comme paramètre est contenu dans ce rectangle. Cette méthode peut, par exemple, être utile pour vérifier qu'on a cliqué avec la souris sur un objet dérivé de la classe **`Rectangle`**.

Les méthodes **`contains(Rectangle r)`** permettent de vérifier si le rectangle spécifié comme paramètre est contenu dans ce rectangle (résultat **`true`**) ou non (résultat **`false`**).



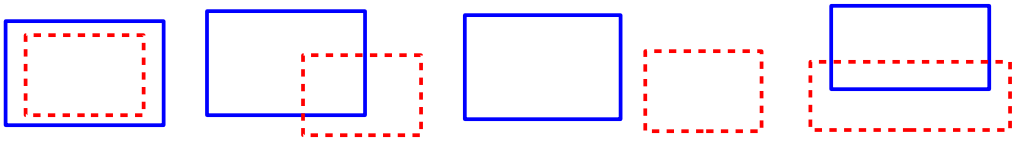
Résultat: **`true`** **`false`** **`false`**

avec



Ce rectangle Rectangle spécifié comme paramètre

La méthode i**`ntersects(Rectangle r)`** permet de vérifier si l'intersection entre ce rectangle et le rectangle spécifié comme paramètre est non vide (résultat **`true`**) ou vide (résultat **`false`**).



Résultat: **`true`** **`true`** **`false`** **`true`**

Le problème posé peut se résoudre alors en **dérivant les classes `Turtle` et `Crosshair` de la classe `Rectangle`** (non représenté sur le schéma UML à la page suivante).

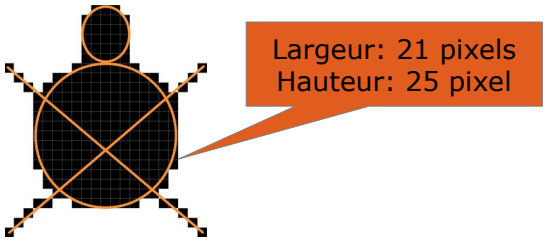
En procédant de cette manière, quelle méthode héritée pouvez-vous alors utiliser pour vérifier que la tortue a été touchée ? Comment procédez-vous ?

Voici quelques précisions concernant les classes à développer :

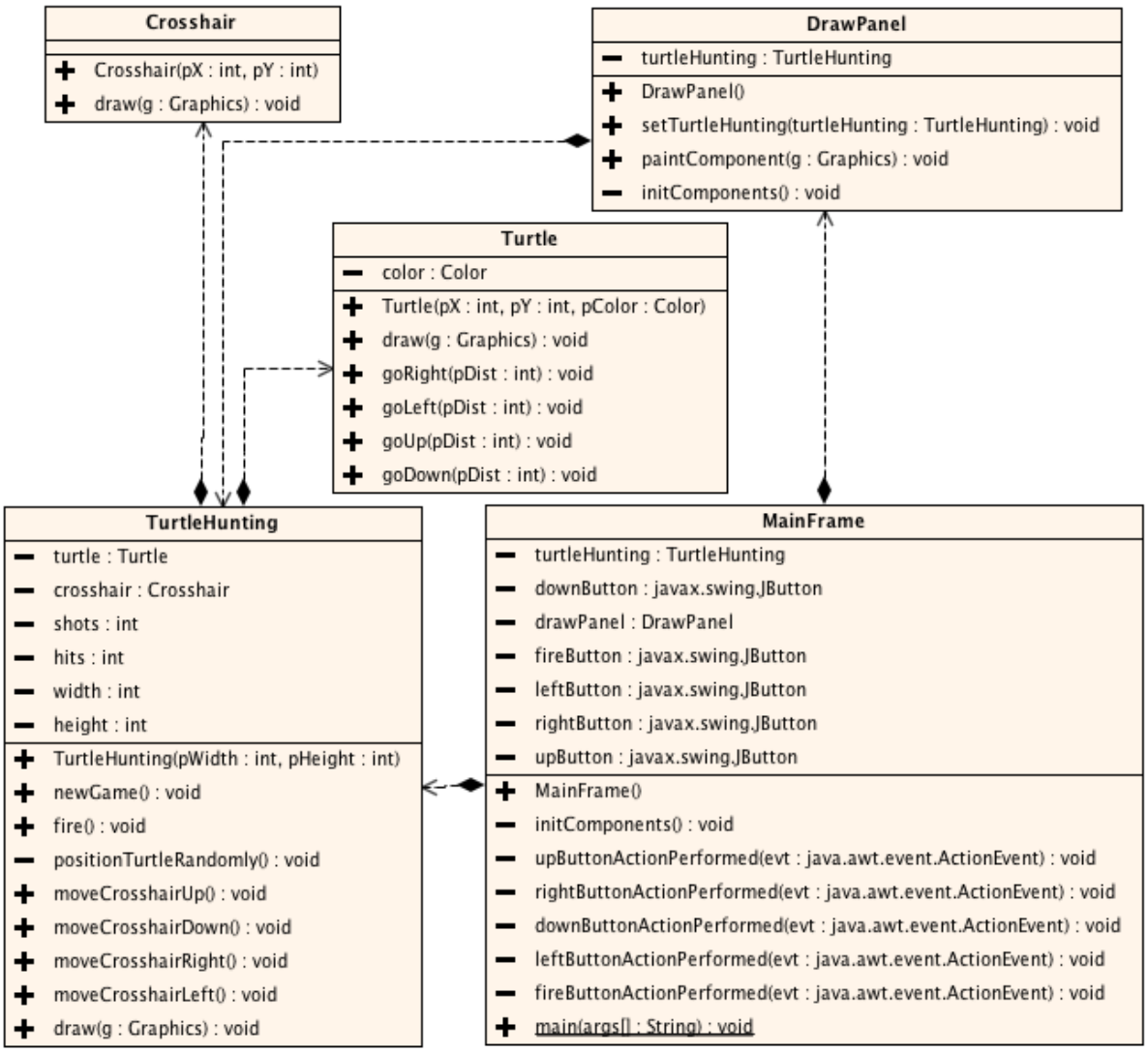
La classe **`Turtle`** représente une tortue.

Les méthodes **`goRight`**, **`goLeft`**, **`goUp`**, **`goDown`** permettent de déplacer la tortue sur le canevas de dessin respectivement vers la droite, la gauche, le haut et le bas. Le paramètre **`pDist`** spécifie la distance de déplacement en pixels.

La méthode **`draw`** dessine sur le canevas de dessin la tortue dans la couleur spécifiée par l'attribut **`color`**.



La classe **`Crosshair`** dispose également d'une méthode **`draw`** qui permet de dessiner le réticule sur le canevas de dessin.



La classe **`TurtleHunting`** implémente le jeu tandis que la classe **`DrawPanel`** permet de représenter sur un panneau la vue à travers le viseur du fusil.

Attributs :

- **`turtle`** la tortue
- **`crosshair`** le réticule
- **`shots`** le nombre de tirs effectués
- **`hits`** le nombre de tortues abattues
- **`width / height`** les dimensions de la surface de dessin

Méthodes:

- **newGame** commencer un nouveau jeu et procéder aux initialisations nécessaires
- **moveCrosshairUp**, **moveCrosshairDown**, **moveCrosshairRight**, **moveCrosshairLeft** déplacer le fusil dans la direction indiquée. Le déplacement du fusil dans une direction (par exemple, vers la droite) peut être simulé en déplaçant la tortue dans la direction opposée (par exemple, vers la gauche).
- **fire** vérifier si la tortue a été touchée et actualiser les attributs **shots** et **hits**. Si la tortue a été touchée alors elle est repositionnée aléatoirement sur la panneau.
- **positionTurtleRandomly** placer la tortue aléatoirement sur le panneau.
- **draw** afficher les différents composants du jeu :
  le réticule, la tortue, le nombre de tirs, …
  Pour rendre la vue à travers le viseur plus réaliste, vous pouvez dessiner le panneau en noir et dessiner un cercle blanc au milieu.

Il reste à développer l'interface graphique dans la classe **MainFrame** avec les boutons de déplacement du fusil et le bouton de tir.

Question:

Au lieu de dériver la classe **Crosshair** de la classe **Rectangle**, quelle autre conception auriez-vous pu imaginer ?

Notions requises : héritage, surcharge de méthodes
Classes requises : JPanel, Graphics, Color

Exercice G.7: Turtle hunting – homemade Rectangle

Faites une copie du projet **Turtle hunting** et programmez vous-même la classe **Rectangle** qui fonctionne comme la la classe prédéfinie **java.awt.Rectangle** que vous avez utilisée.

Supprimez d'abord les instructions d'importation de **java.awt.Rectangle**, puis ajoutez une nouvelle classe **Rectangle** au projet. Ajoutez-y les attributs (**x, y, width, height**) et méthodes (constructeur, accesseurs, **setLocation**, **contains**, **intersects**) dont vous avez besoin dans votre projet. Définissez des méthodes supplémentaires si cela vous paraît convenable.

Etablissez d'abord des croquis de tous les cas possibles

a) où **contains** retourne **true**,

b) où **intersects** retourne **true**.

Il peut même être utile de développer une petite application pour tester les résultats des deux méthodes.

Notions requises : calculs sur les coordonnées graphiques
Classes requises : -

Exercice G.8: Turtle hunting – améliorations

Pour rendre le jeu plus intéressant, ajoutez les fonctionnalités suivantes :

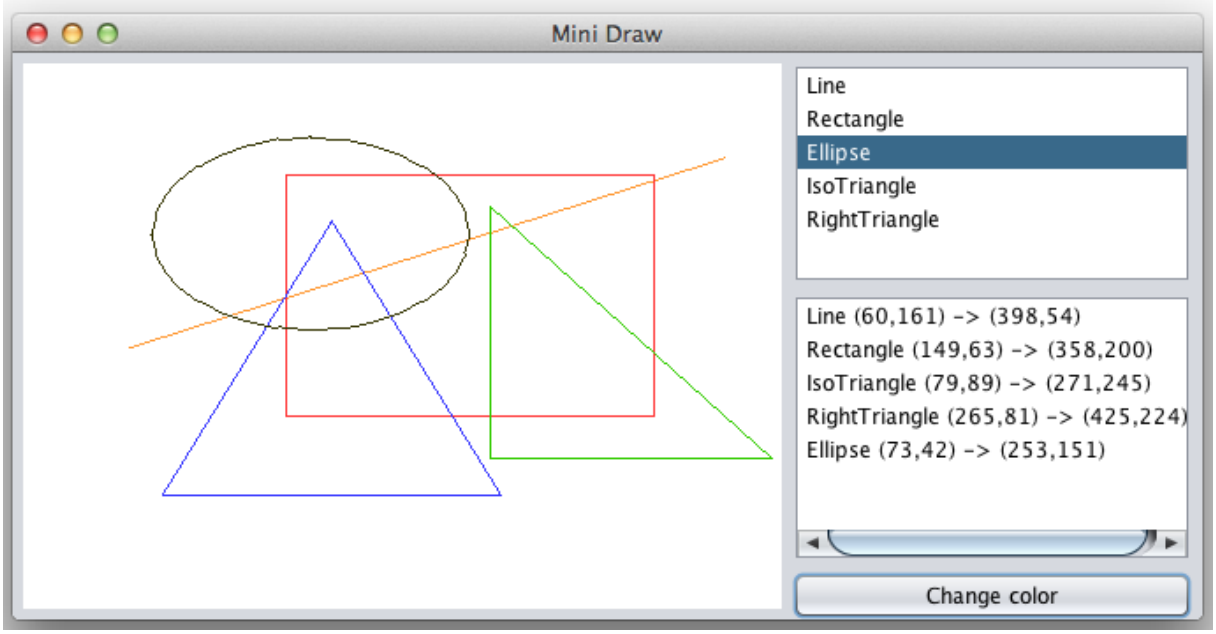
- La tortue effectue des mouvements aléatoires (p.ex. de -10 à 10 pixels) dans une direction aléatoire à des intervalles réguliers (p.ex. 5 fois par seconde).
- Il est possible de bouger le fusil par la souris. P.ex. lorsqu'on enfonce le bouton gauche de la souris et on bouge la souris dans une direction, le fusil est déplacé dans la direction de la souris (c.-à-d. la tortue est déplacée dans la direction opposée).
- On peut tirer avec le bouton droit de la souris.
- Inversez le comptage, c.-à-d. au début, on a 10 balles dans le fusil et 5 tortues. Si on réussit à abattre les 5 tortues avec les 10 balles, on a gagné, sinon on a perdu. Renommez pour cela l'attribut **`shots`** en **`bulletsLeft`** et **`hits`** en **`turtlesLeft`**. Ajoutez un attribut booléen **`gameOver`** à **`turtleHunting`** qui est **`true`** dés que le jeu est fini. La méthode **`draw`** affiche "You won" ou "You lose" à la fin du jeu.
- Enfin, modifiez le programme de façon à ce que la zone de tir dans **Crosshair** ait 11 pixels de hauteur et de largeur. Montrez cette zone en rouge au centre du réticule.

Notions requises : Timer, événements de la souris et du clavier, random

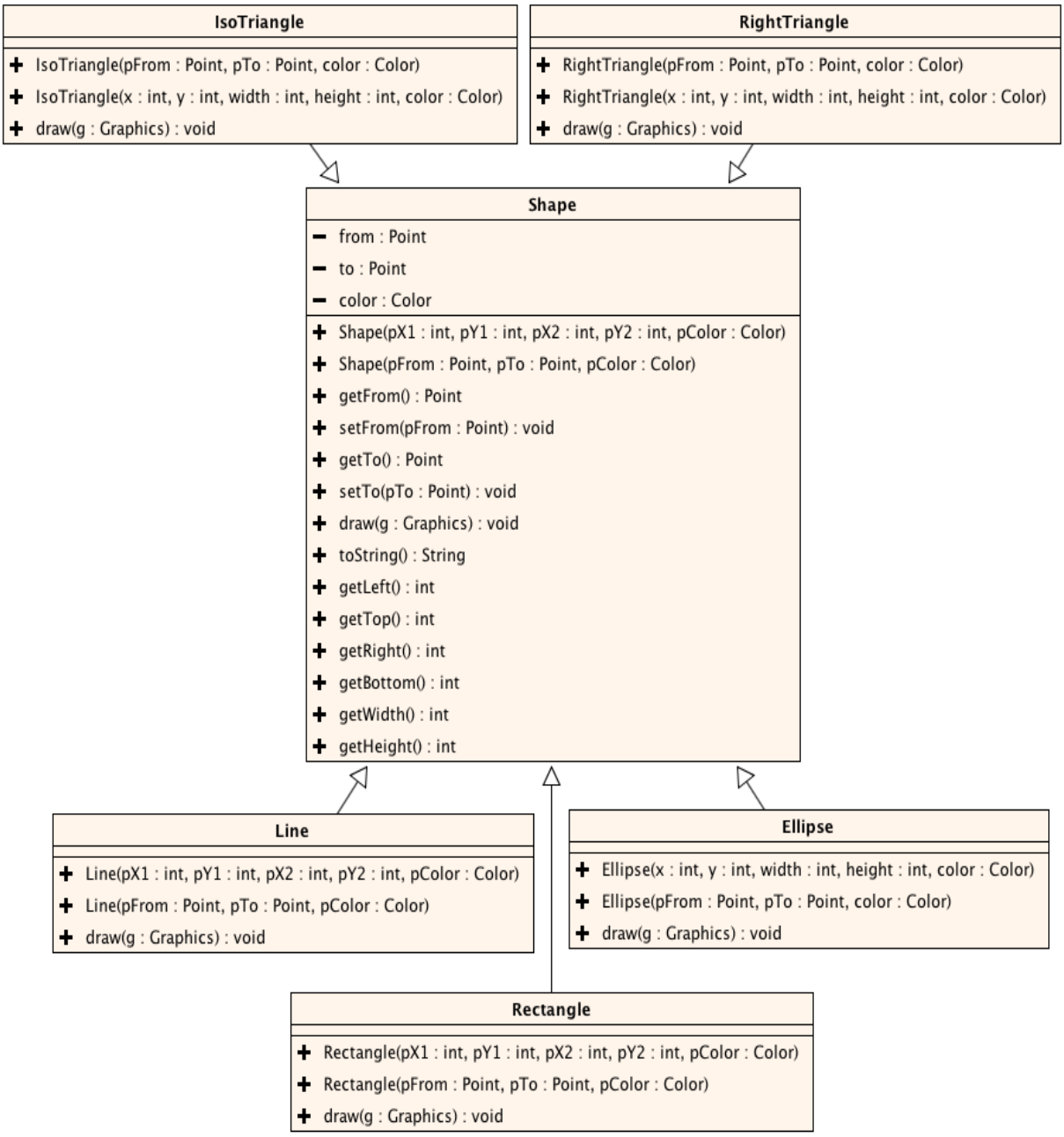
Exercice G.9: Traceur de figures géométriques « Mini Draw »

Reprenez l'exercice **E.3 Traceur de lignes en couleur** et transformez-le pour qu'on puisse dessiner des figures géométriques (angl. *Shape*) de différents types (lignes, rectangles, ellipses, triangles isocèles, triangles à angle droit, …).

1. Remplacez la classe **Line** par la classe **Shape**. Vous pouvez renommer la classe **Line** à l'aide de l'utilitaire ***Refactor*** et appliquer ensuite les modifications suivantes :
   - Enlevez le dessin des lignes dans la méthode **draw()**.
   - Ajoutez les méthodes **getWidth()** et **getHeight()** qui retournent la largeur et la hauteur de la figure géométrique.
   - Ajoutez les méthodes **getLeft()**, **getRight()**, **getTop()** et **getBottom()** qui retournent les abscisses et les ordonnées des limites de la figure géométrique.
   - Modifiez la méthode **toString()** pour ajouter la classe de la figure géométrique.
2. *Dans le cadre du MVC, quel avantage nous donne la définition des méthodes **getLeft()**, **getTop()**, **getWidth()** et **getHeight()** par rapport à l'exercice **G.3** ?*
3. Remplacez la classe **Lines** par la classe **Shapes** permettant de sauvegarder toutes les figures géométriques dessinées. Il suffit de renommer la classe **Lines** et l'attribut **allLines** à l'aide de l'utilitaire **Refactor**.
4. Rajoutez ensuite les classes **Rectangle**, **Ellipse**, **IsoTriangle** et **RightTriangle**.
5. Dans la classe **MainFrame** ajoutez la liste **typeList** qui permet de sélectionner le type de la figure géométrique à dessiner. Vous pouvez initialiser la liste à l'aide de la propriété **model**. Lors du lancement du programme, la figure **Line** doit être sélectionnée. Rajoutez ensuite le code Java pour que les figures géométriques puissent être dessinées.



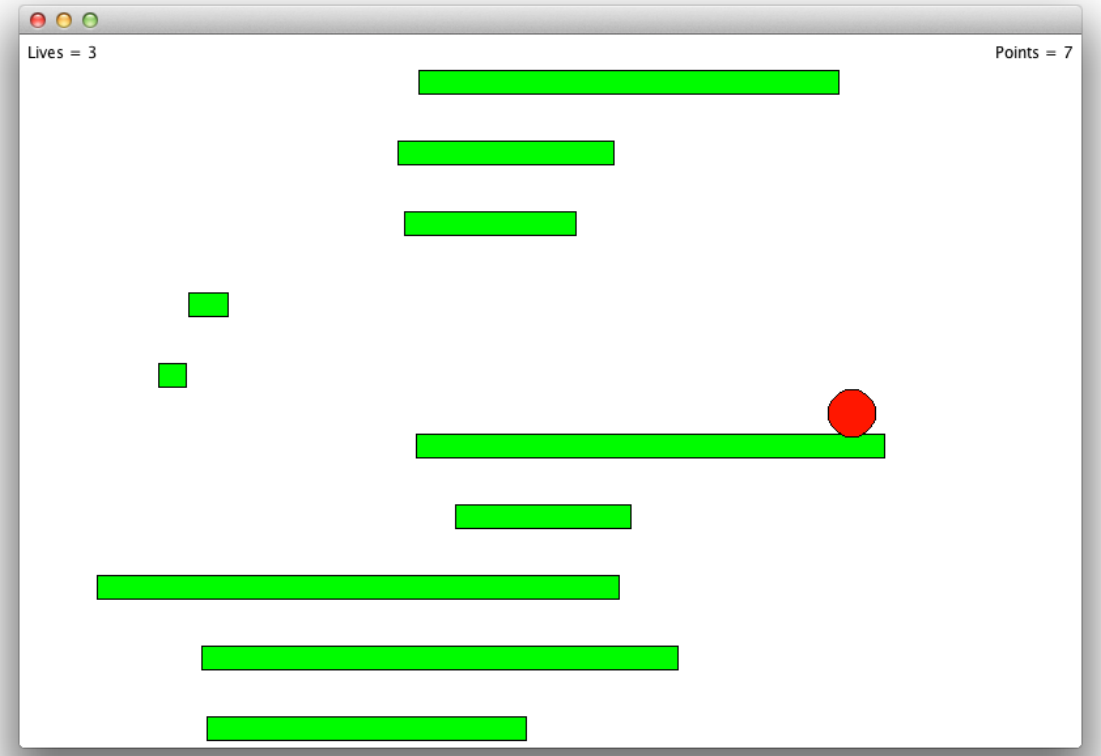
Voici le digramme UML de la partie « modèle ». Les classes **DrawPanel**, **MainFrame** et **Shapes** sont à déduire de la capture d'écran précédente.



Notions requises : dessin, événements souris, héritage, surcharge de méthodes, encapsulation

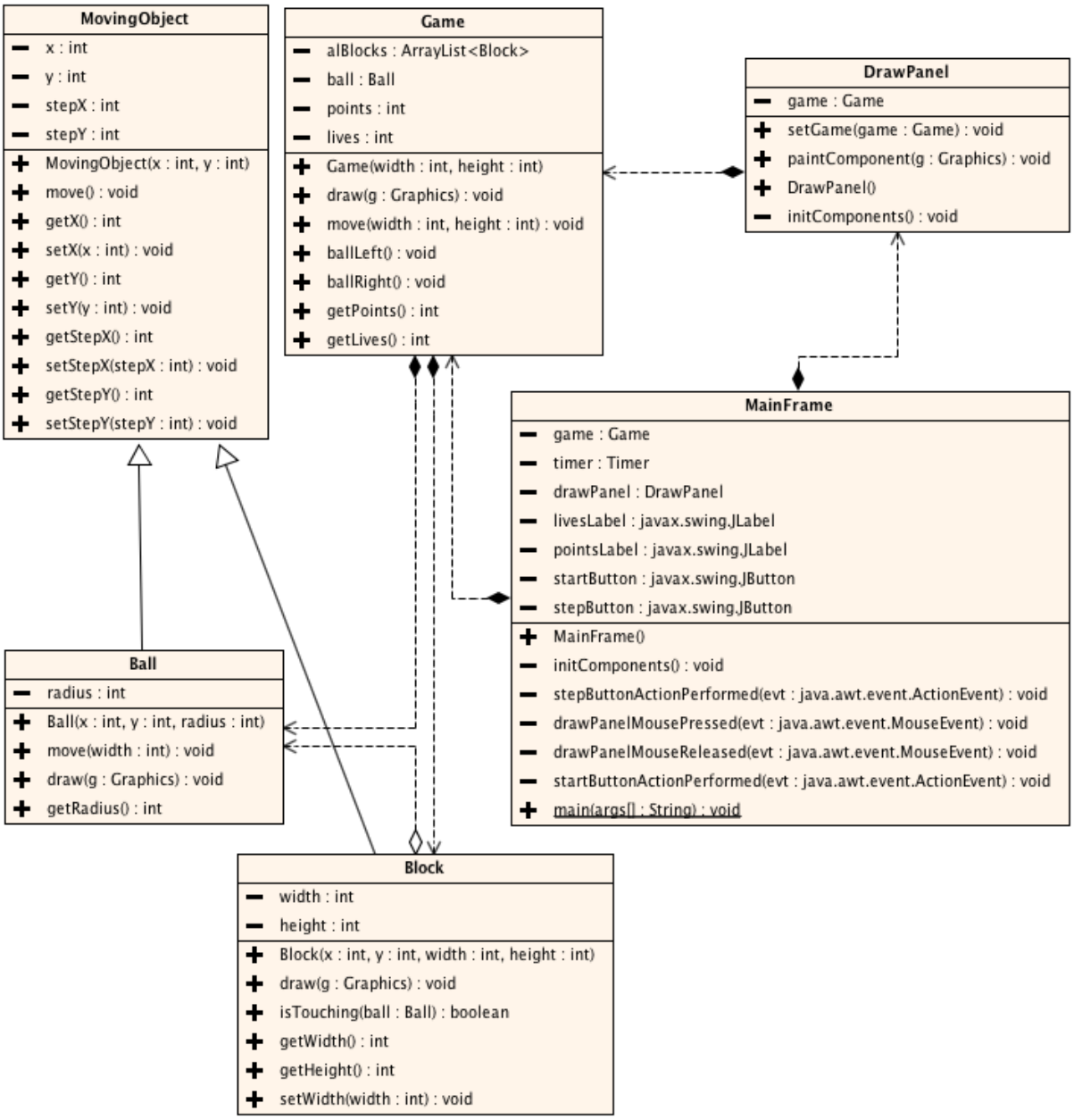
Exercice G.10: Fall Ball

Dans le présent exercice il s'agit de programmer un petit jeu qui fonctionne de la manière suivante :



- Sur la scène de jeu il y a 10 barres qui migrent du bas vers le haut. A chaque fois qu'une barre sort en haut de la fenêtre, elle réapparaît en bas de l'écran à une position aléatoire et avec une largeur aléatoire.
- Le joueur possède une balle qui apparaît après le démarrage du jeu au milieu de la scène. La balle tombe vers le bas jusqu'à ce qu'elle touche l'une des barres. Elle est portée par les barres.
- Lorsque le bouton gauche de la souris est enfoncé, la balle se déplace vers la gauche sur la barre où elle se trouve actuellement. Avec le bouton droit de la souris elle est déplacée vers la droite.
- Une fois dépassé le bord de la barre, la balle tombe vers le bas et n'a plus de mouvement horizontal.
- A chaque fois qu'une barre sort du haut de la scène de jeu, le joueur obtient un point. A chaque nouvelle centaine de points, le joueur gagne une vie.
- A chaque fois que la balle sort en haut ou en bas, le joueur perd une vie. La balle réapparait au milieu de l'aire de jeu.

Voici le schéma UML complet de l'application:



1. Commencez par implémenter la classe **`MovingObject`** qui représente un objet en mouvement.
   - La méthode **`move()`** déplace l'objet selon les valeurs des attributs **`stepX`** et **`stepY`**.
2. Continuez ensuite avec la classe **`Ball`**.
   - Son déplacement vertical dès sa création est fixé à 5 pixels.
   - La balle est rouge et entourée d'une ligne noire.
   - Elle ne peut se déplacer en dehors de l'aire de jeu. La valeur de la largeur de l'aire de jeu est passée en tant que paramètre à la méthode **`move()`**.

3. La prochaine classe à implémenter est la classe **`Block`**.
   - La vitesse de déplacement vertical d'une barre est fixée à sa création à -3 pixels. Les barres sont dessinées en vert avec une bordure noire.
   - La méthode i**`sTouching(Ball)`** teste si le point inférieur de la balle passée en tant que paramètre se trouve à l'intérieur de la barre. **Faites un dessin!!!**
4. Créez la classe **`Game`** qui règle le bon déroulement du jeu :
   - Au début, l'attribut **`ball`** est **`null`**, le joueur a trois vies et ne possède pas encore de points.
   - Lors de la création du jeu:
     - 10 barres sont créées. La distance entre les points (x,y) des barres est un dixième de la hauteur totale de l'aire de jeu.
       - La hauteur d'une barre est un trentième de la hauteur totale de l'aire de jeu.
       - La position horizontale d'une barre est aléatoire sans pour autant dépasser la moitié de la largeur disponible.
       - La largeur d'une barre est aléatoire sans pour autant dépasser 2/3 de la distance entre sa position horizontale et la limite droite.
     - Une nouvelle balle « joueur » est créée et placée au milieu de la scène de jeu.
   - La méthode **`draw`** dessine les barres et, si possible, la balle.
   - Les méthodes **`ballLeft()`** et **`ballRight()`** donnent un mouvement de 10 pixels dans leur direction respective.
   - La méthode **`move`** …
     - … déplace d'abord toutes les barres.
       - Si une barre sort en haut elle est placée tout en bas. Sa position horizontale ainsi que sa largeur sont recalculées de manière aléatoire (voir constructeur) et le joueur gagne un point.
       - Si en plus le nombre de points est un multiple de 100, le joueur gagne une vie.
       - Si la balle touche une barre, le mouvement vertical est mis à la même valeur que celui de la barre.
     - Si la balle ne touche aucune barre, son mouvement horizontal est annulé et le mouvement vertical est mis à 5 pixels.
     - Ensuite la balle est déplacée. Si elle sort en haut ou en bas, elle est repositionnée au centre et le joueur perd une vie.
5. Le jeu est créé et commence dès que l'utilisateur appuie sur le bouton « Start ». La classe **`MainFrame`** possède aussi un chronomètre qui fait bouger les éléments du jeu et met à jour l'interface graphique toutes les 20 millisecondes. Si le joueur ne possède plus de vie, le jeu est terminé et le bouton « Start » réapparait.
   - Le joueur contrôle le mouvement horizontal de la balle à l'aide des deux boutons de la souris.

Notions requises : dessin, timer, héritage, surcharge d'une méthode, encapsulation