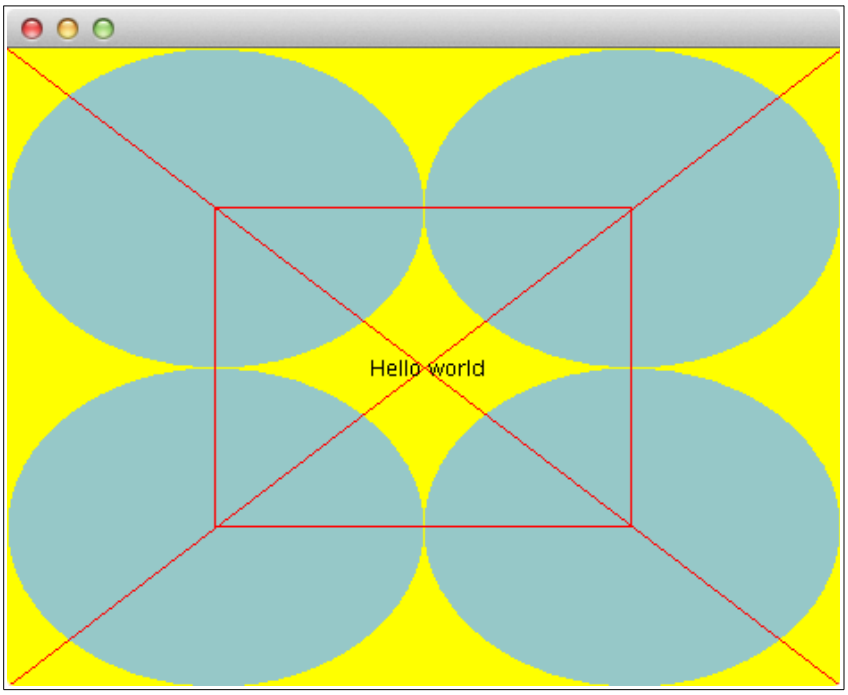
Série D : Dessin et graphisme

Table des matières

Remarque importante : MVC

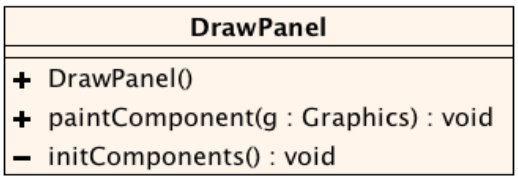
Les dessins des premiers exercices se feront directement sur un panneau (la vue), sans créer une classe modèle. Ces exercices serviront surtout à vous familiariser avec le canevas ou à préparer les exercices qui suivront.

Exercice D.1: Dessin de figures colorées (sans MVC)



Pour vous familiariser avec le canevas et ses méthodes, créez un nouveau projet en NetBeans et réalisez point par point l'exercice suivant :

- Ajoutez une fiche **`MainFrame`** au projet.
- Ajoutez aussi une nouvelle classe du type « JPanel Form » au projet selon le schéma UML suivant :

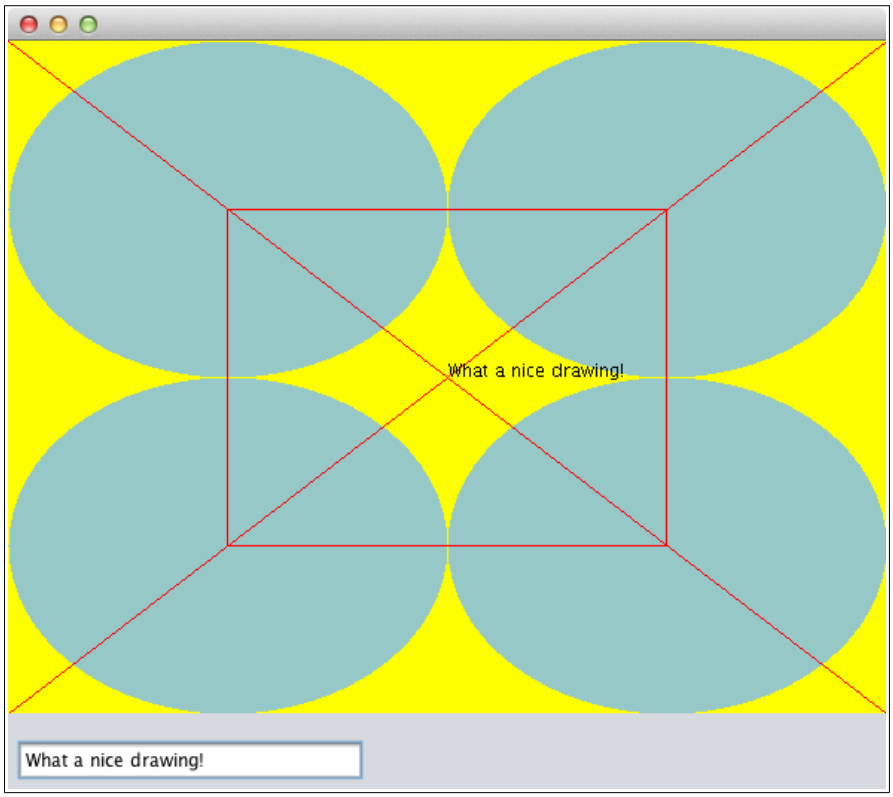


- Compilez le projet et glissez la classe **`DrawPanel`** sur la fiche. Ce panneau doit occuper presque toute la fiche même quand celle-ci est redimensionnée lors de l'exécution du programme. Nommez ce panneau **`drawPanel`**.
- Développez la méthode **`paintComponent`** qui dessine les figures suivantes sur le canevas (ces figures doivent se redimensionner automatiquement en cas de remdimensionnement de la fiche) :
  - L'arrière-fonds du panneau est colorié en jaune.
  - Quatre ovales adjacents sont dessinés comme indiqué sur la capture d'écran ci-dessus. Choisissez pour ces ovales une couleur non prédéfinie dans la classe **`Color`** mais créez votre propre couleur selon le schéma RGB.
  - Une croix rouge diagonale.

- Un rectangle rouge reliant les centres des quatre ovales.
- Le texte « Hello world » imprimé en noir. Le texte doit se trouver (approximativement) au centre du panneau.

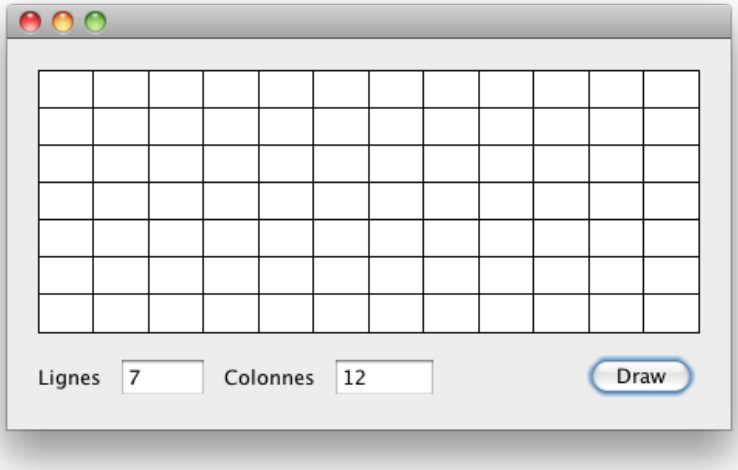
Extension :

Modifiez le projet de manière à pouvoir saisir dans un composant **`TextField`**, placé sur la fiche **`MainFrame`**, le texte affiché sur le panneau **`drawPanel`**. Quelles modifications devez-vous apporter à la classe **`DrawPanel`**?



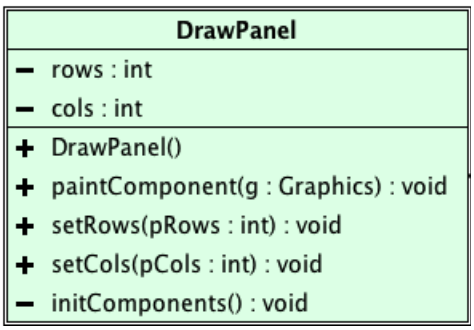
Notions requises:	Dessiner sur le canevas
Composants requis:	JPanel, TextField
Autres objets:	Graphics, Color

Exercice D.2: Grille (sans MVC)



Écrivez une application permettant de dessiner une grille de **`N`** lignes fois **`M`** colonnes. Les valeurs par défaut sont : **`N=7`** et **`M=12`**

Afin d'y arriver, ajoutez votre fiche principale **`MainFrame`** puis ajoutez aussi une nouvelle classe du type « JPanel Form » à votre projet et programmez la de la façon suivante :



Explications

- **`rows`** et **`cols`** indiquent le nombre de lignes et de colonnes à dessiner.
- **`paintComponent(Graphics g)`** est la méthode responsable de réaliser le dessin.
- **`setRows`** et **`setCols`** sont les modificateurs des deux attributs.
- **`initComponents`** est une méthode créée par NetBeans.

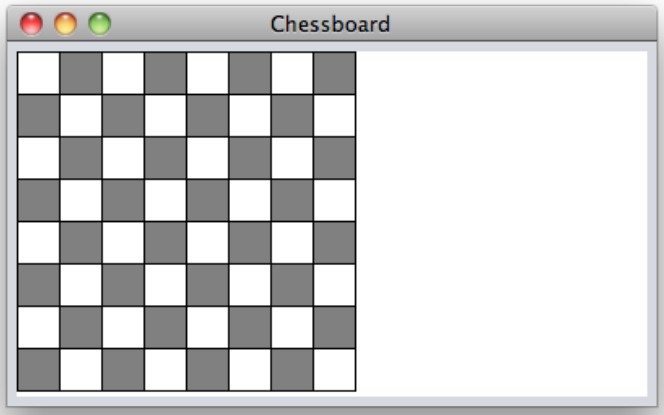
Compilez votre projet, puis ajoutez une instance de la classe **`DrawPanel`** sur votre fiche principale.

Notions requises:	boucles, coordonnées
Composants requis:	JFrame, JButton, JLabel, JTextField, JPanel
Autres objets:	Graphics, Color

Exercice D.3: Echiquier (sans MVC)

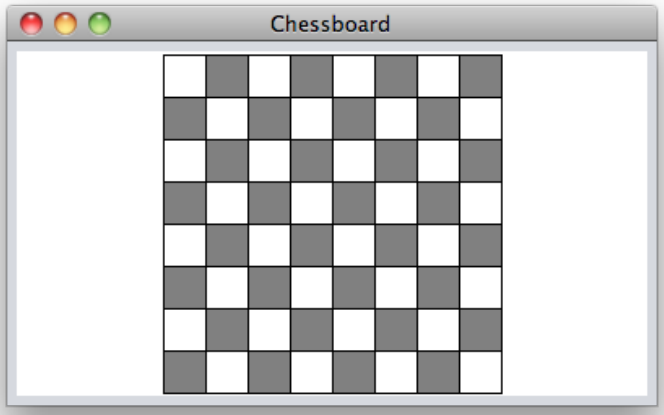
a) Écrivez une application permettant de dessiner un échiquier (DE : Schachbrett). L'échiquier consiste en une grille de 8 x 8 carrés alternativement blancs et gris. L'échiquier dessiné doit être un carré, même si le canevas ne l'est pas. L'échiquier a la taille carrée maximale possible sur le canevas, même si la fiche et le panneau sont redimensionnés. Le dessin est lancé par un bouton.

Commencez à programmer une version dans laquelle l'échiquier colle toujours contre le côté gauche :



Veillez à ce que chaque carré ait aussi une bordure noire.

b) Modifier dans une deuxième phase votre code de manière à ce que l'échiquier soit centré exactement au milieu dans le panneau.



Notions requises:	boucles, coordonnées
Composants requis:	JFrame, JButton, JLabel, JTextField, JPanel
Autres objets:	Graphics, Color

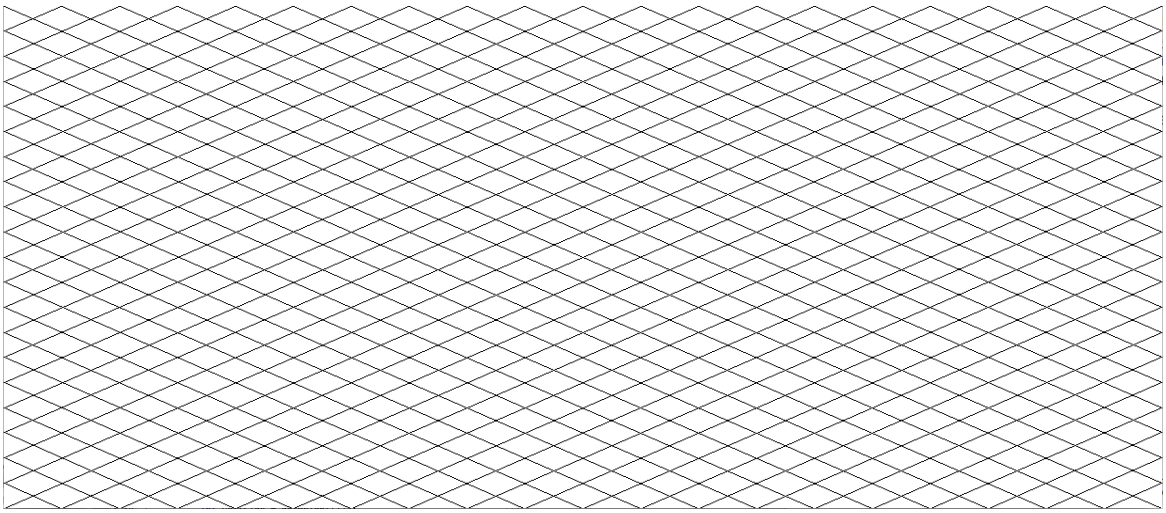
Exercice D.4: Color Mixer (sans MVC)

Il s'agit de réaliser un petit programme pour montrer l'effet du mixage des couleurs élémentaires RGB (*Red, Green, Blue*) ainsi que leur transparence (*Alpha channel*). Une grille est dessinée au fond des rectangles pour voir l'effet de la transparence.

- Créez d'abord la fiche **MainFrame** avec les 4 glisseurs qui sont regroupés dans un panneau.
- Créez dans **DrawPanel** les attributs nécessaires ainsi que leurs manipulateurs (*setters*).

Tous les dessins sont à réaliser dans la méthode **paintComponent** de **DrawPanel** :

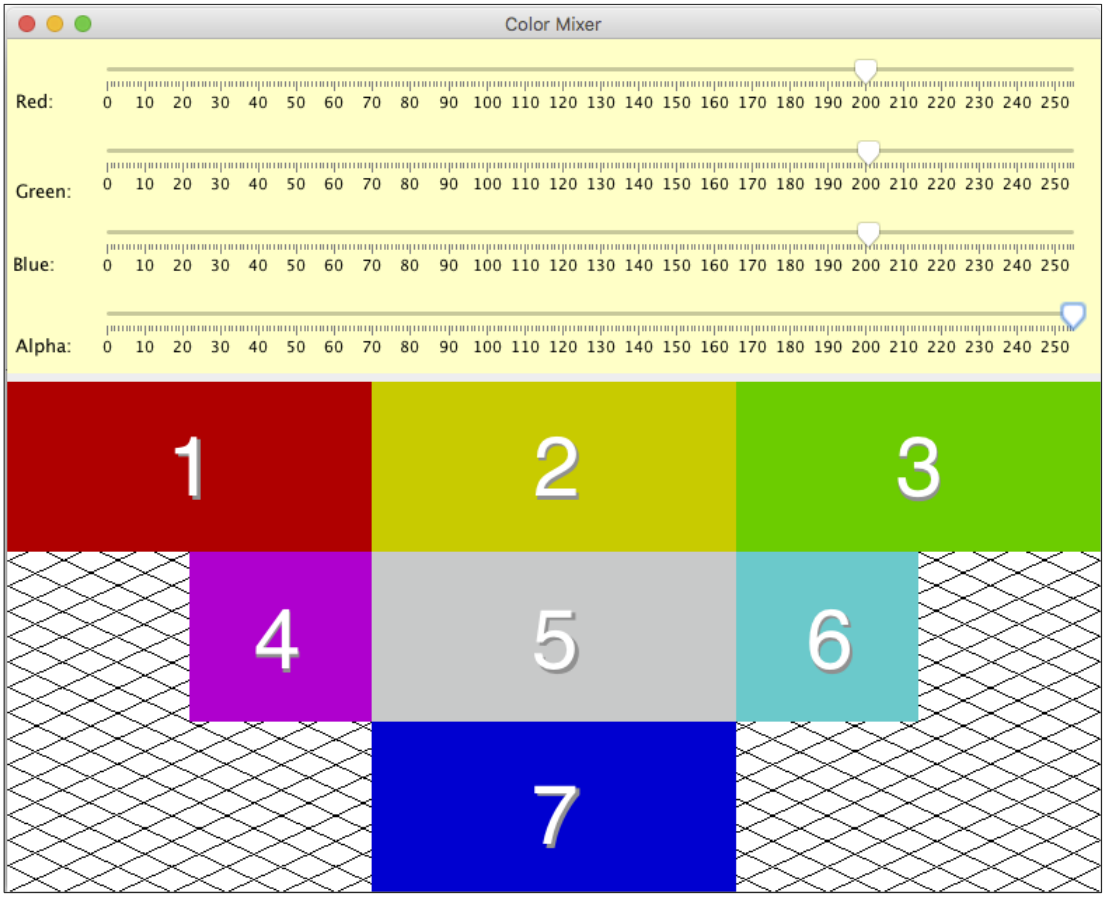
- Le fond de la surface de dessin est blanc.
- Tracez les diagonales en noir comme suit : En hauteur ainsi qu'en largeur, le panneau est divisé en 20 parties égales. Reliez les points limitant ces parties comme montré dans le programme modèle. Le dessin s'adapte en hauteur et en largeur, mais le nombre de diagonales reste toujours le même.
  Conseil : Utilisez deux variables pour la distance entre les diagonales en horizontale et en verticale. Effectuez un croquis sur une feuille pour déterminer les coordonnées des points qui sont reliés par les diagonales.



- Dessinez les 7 rectangles montrant les différentes combinaisons des couleurs fondamentales. (Les numéros dans l'illustration ci-dessous servent seulement à mieux comprendre l'énoncé. Vous n'avez pas besoin d'afficher ces numéros).

  En principe, la surface de dessin est divisée en 9 régions de même dimension (3 divisions en hauteur et 3 divisions en largeur). Cependant les rectangles 4 et 6 ne possèdent que la moitié de la largeur des autres rectangles.

  - La couleur du rectangle 1 est définie par la composante rouge uniquement.
  - La couleur du rectangle 2 est définie par les composantes rouge et verte uniquement.
  - La couleur du rectangle 3 est définie par la composante verte uniquement.
  - La couleur du rectangle 4 est définie par les composantes rouge et bleue uniquement.
  - La couleur du rectangle 5 est définie par les composantes rouge, verte et bleue.
  - La couleur du rectangle 6 est définie par les composantes verte et bleue uniquement.
  - La couleur du rectangle 7 est définie par la composante bleue uniquement.
  - La transparence (*alpha*) agit sur tous les rectangles.



Notions requises:	boucles, coordonnées
Composants requis:	JFrame, JPanel, JSlider
Autres objets:	Graphics, Color avec transparence

Remarque importante : MVC

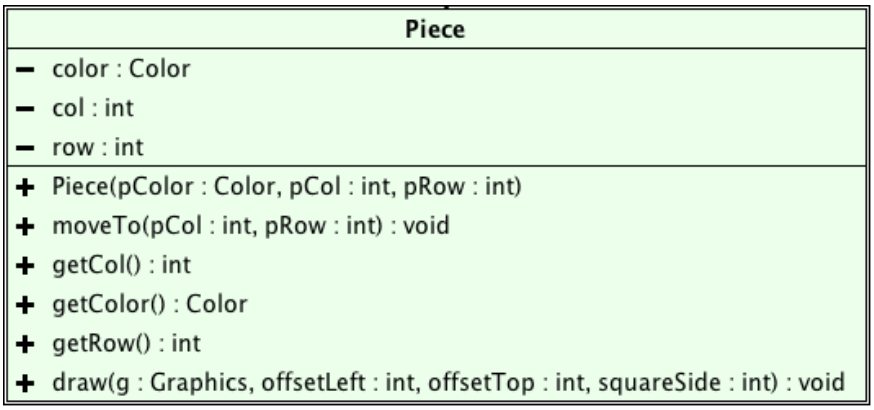
Les exercices suivants se feront en respectant le schéma UML expliqué dans le cours (→ chapitre *"Les dessins et le modèle MVC"*).

Principe : Les classes du modèle savent se dessiner soi-même par la méthode **draw.** La classe **drawPanel** fait appel aux méthodes **draw** à chaque fois que c'est nécessaire. Si nécessaire, les méthodes **draw** possèdent des paramètres pour le calcul des positions.

Exercice D.5: Damier

Dans la suite, vous allez développer une classe qui sait gérer et dessiner un damier.

- Un damier peut posséder un certain nombre de pions de couleurs différentes (en général noir et blanc). Créez d'abord une classe **`Piece`** (pion) qui représente les pions. Chaque pion est défini par sa couleur et sa position (col,row, 8 x 8) sur la table de jeu. Une méthode **`moveTo`** place le pion à une nouvelle position.
- La méthode **`draw`** dessine un pion. Les paramètres **`offsetTop`** et **`offsetLeft`** représentent la marge gauche, respectivement la marge supérieure (Ces marges sont utilisées pour centrer le damier sur le canevas). **`squareSide`** indique la taille de la cellule.



Développez ensuite la classe **`Checkers`** qui représente la table de jeu dans la mémoire et qui sait manipuler les pions :

- Une liste appelée **`alPieces`** contient tous les pions qui se trouvent sur la table de jeu. Au début, l'échiquier est vide.
- Une méthode **`init`** place les pions des deux joueurs (p.ex. rouge et bleu) comme on les retrouve normalement au départ sur un damier (voir Wikipedia).
- Comme le seul endroit où nous avons mémorisé les positions des pions est la liste **`pieces`**, il sera pratique d'écrire une méthode **`getPieceAt(col,row)`** qui traverse la liste des pions et qui retourne le pion qui se trouve à une position donnée ou **`null`** si cette position est libre.
- A l'aide de la méthode **`move(startCol,startRow, destCol,destRow)`** on peut déplacer ou enlever un pion. Un pion est enlevé, si la position d'arrivée se trouve à l'extérieur du damier. Il ne faut pas contrôler si un mouvement est valide selon les règles du jeu, il suffit de vérifier que la position de départ contient un pion et la position d'arrivée est une position vide sur le damier ou se trouve à l'extérieur du damier. La méthode retourne **`true`**, si l'action a été exécutée avec succès.
- La méthode **`draw`** dessine la grille et commande aux différents pions de la liste de se tracer sur le canevas. Référez-vous à l'exercice '*Echiquier*' pour tracer la grille!

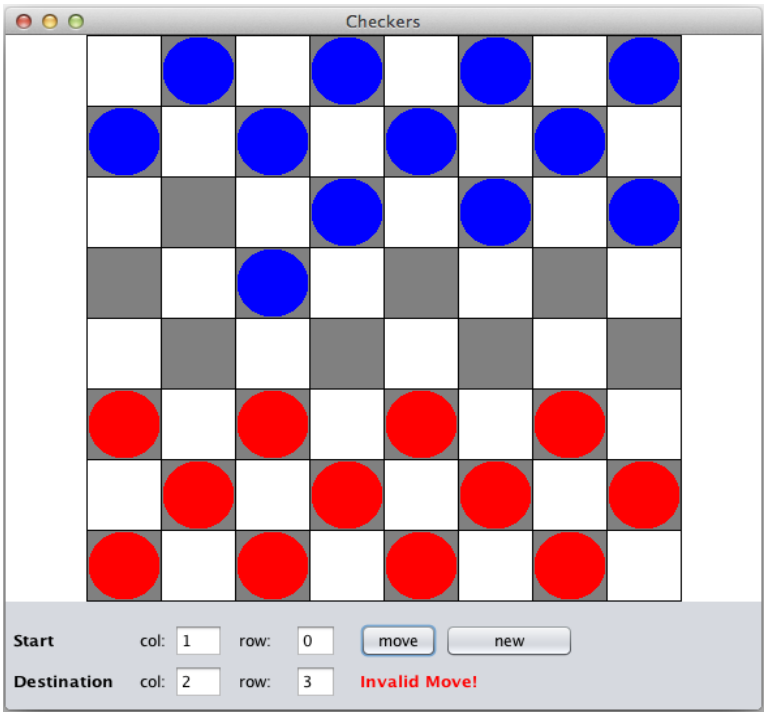
Checkers
− alPieces : ArrayList<Piece>
+ init() : void + getPieceAt(col : int, row : int) : Piece + move(startCol : int, startRow : int, destCol : int, destRow : int) : boolean + draw(g : Graphics, width : int, height : int) : void

La classe **`DrawPanel`** n'a qu'à commander maintenant son instance de **`Checkers`**, de se tracer sur le canevas, mais uniquement, si celle-ci n'est pas nulle. Il faut donc pouvoir lui passer une instance de la classe **`Checkers`** (par **`setCheckers`** → voir cours).

DrawPanel
− checkers : Checkers
+ DrawPanel() + setCheckers(pCheckers : Checkers) : void + paintComponent(g : Graphics) : void − initComponents() : void

Ajoutez une nouvelle fiche avec une instance de la classe **`DrawPanel`**, 4 champs textes et 2 boutons :

- Le premier bouton sert à lancer une nouvelle partie et demande au panneau de se redessiner.
- Les champs texte servent à indiquer les coordonnées de départ et d'arrivée des déplacements. Le deuxième bouton sert à effectuer le déplacement.



Etablir un diagramme montrant les objects et leurs relations dans ce projet.

Notions requises:	MVC, new, typecast, ArrayList, boucles, coordonnées
Composants requis:	JFrame, JButton, JLabel, JTextField, JPanel
Autres objets:	Graphics, Color

Recette : Analyse et conception selon le principe MVC

Analysez d'abord le problème et développez le **schéma UML** de la classe **modèle**, en vous posant l'une après l'autre les questions suivantes (par exemple en discutant en groupes de deux) :

1. Quelles parties du programme seront réalisées dans le modèle (et quelles parties dans la vue ou le contrôleur) ?
2. Quels attributs faudra-t-il pour mémoriser les données centrales du modèle ? Faudra-t-il éventuellement définir une/des classes supplémentaires ?
3. Quelles méthodes faudra-t-il définir dans le modèle pour pouvoir l'utiliser convenablement ?
4. Quels noms seraient les plus appropriés pour : la/les classes, les attributs, les méthodes ?
5. Quels seraient les paramètres les plus appropriés pour les méthodes ? Quels seraient leurs noms et leurs types ? Est-ce que les méthodes auront besoin de retourner une donnée et si oui de quel type ?
6. Quelles parties du modèle doivent être publiques ? privées ?

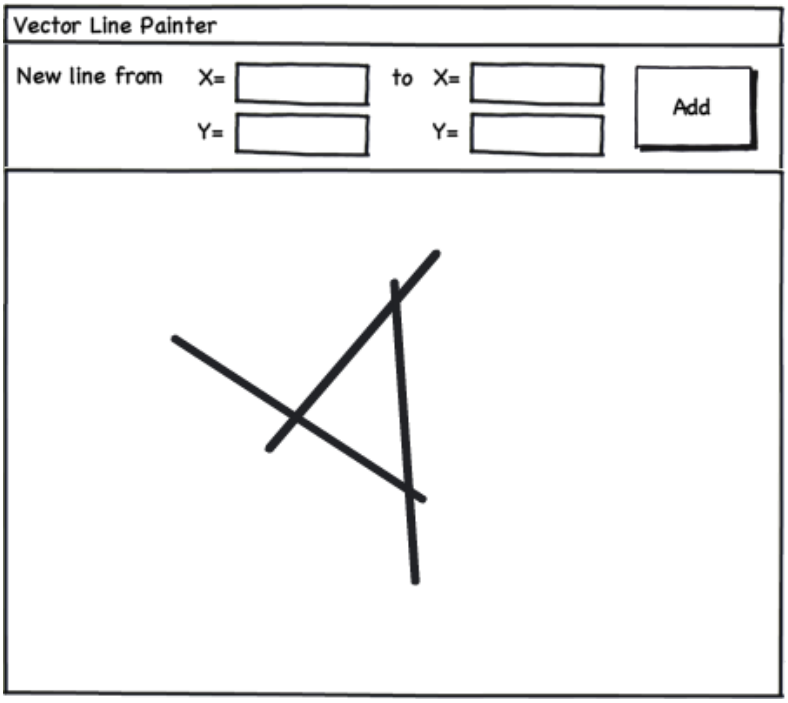
Après avoir défini le schéma UML, réalisez individuellement le modèle en Unimozer ou en NetBeans.

Réalisez ensuite l'interface (vue et contrôleur) en NetBeans et testez l'ensemble.

Exercice D.6: Vector Line Painter

Développez un programme permettant de tracer des dessins vectoriels contenant uniquement des lignes.

Développez d'abord un schéma UML approprié en vous basant sur la structure décrite dans le cours.



Etablir ensuite un diagramme montrant les objets et leurs relations dans ce projet.

Améliorations possibles :

Modifiez votre modèle ainsi que votre vue et votre contrôleur de façon à ce que l'utilisateur puisse choisir une couleur pour chaque ligne. Profitez du composant **`JcolorChooser`**.

Notions requises:	MVC, UML, encapsulation, new, liste, boucles, coordonnées
Composants requis:	JFrame, JButton, JLabel, JTextField, JPanel
Autres objets:	Graphics, Point

Exercice D.7: Histogramme

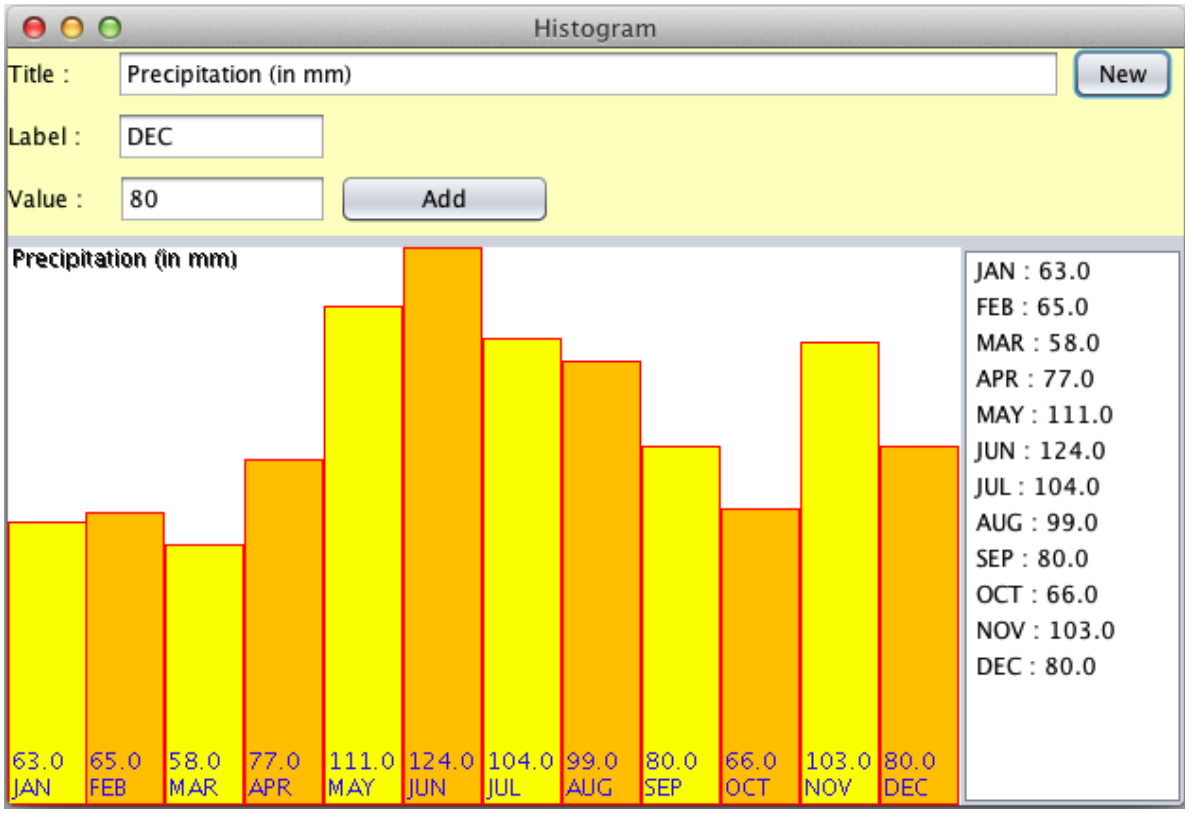
Développez un programme, qui sait représenter sous forme d'histogrammes (DE : *Balkendiagramm*) des séries de nombres réels (positifs).

- Développez d'abord la classe **`NumberInfo`** qui sert à mémoriser un nombre réel positif (**`value`**) avec une brève description textuelle (**`label`**).
- Développez ensuite la classe **`NumberInfos`** qui sait mémoriser et gérer une liste de nombres avec leurs descriptions. **`NumberInfos`** possède aussi un titre pour la série.

  Exemples d'utilisation :

  Averses en mm : 'juin' : 34,5 ; 'juillet' : 52,4 ; 'août' : 112,3 ; ...

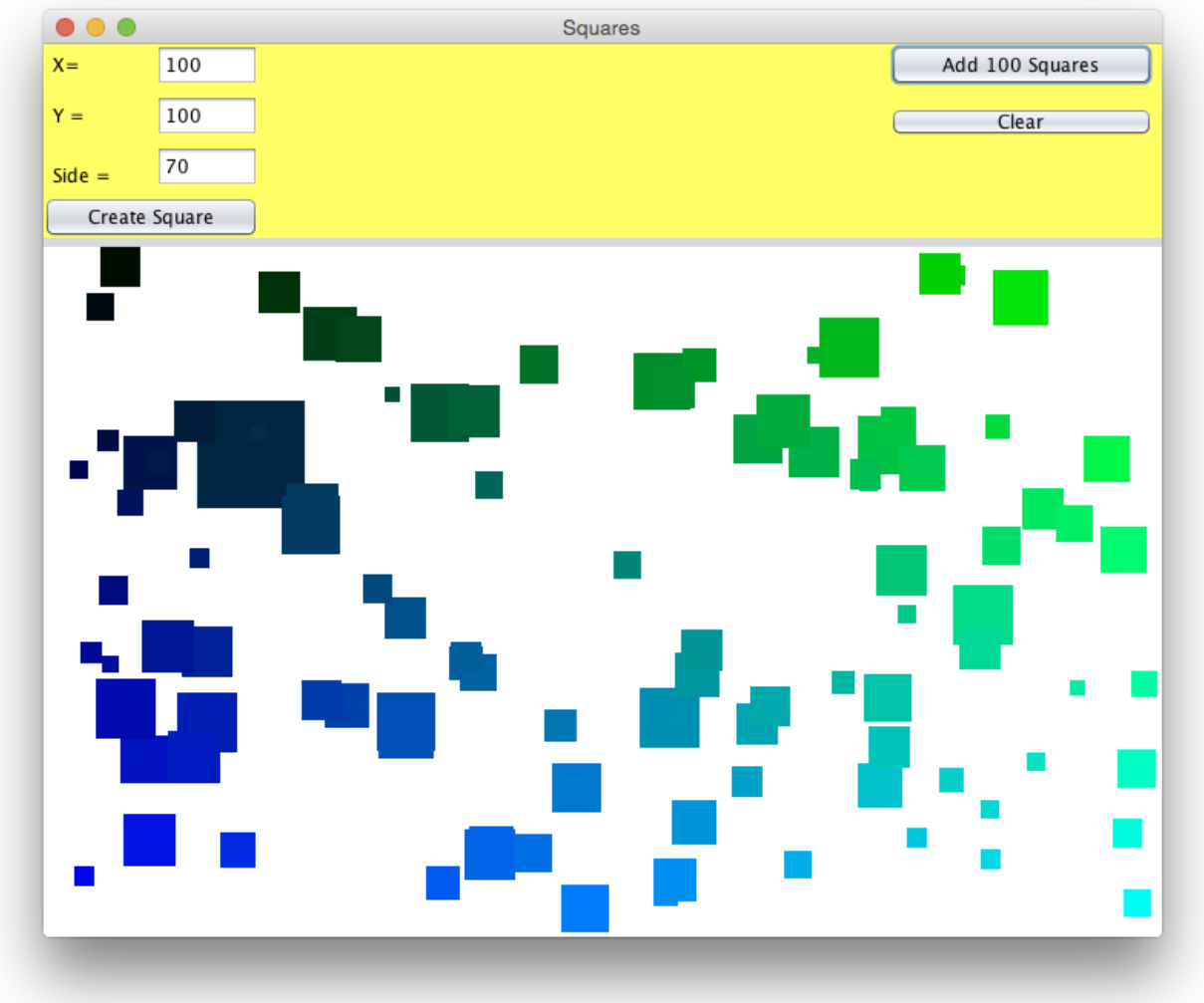
  Élections 2014 en % : 'dei Rosa' : 23,4 ; 'VCS' : 34,4 ; 'PLAS' : 31,6 ; ...
- **`NumberInfos`** sait retourner le total (la somme) et le maximum des nombres mémorisés.
- Il est possible de représenter les informations de **`NumberInfos`** dans une **`JList`**.
- Les méthodes **`get`** et **`size`** correspondent à celle de la liste **`alNumberInfos`**.
- La méthode **`draw`** de **`NumberInfos`** sait représenter les informations de **`alNumberInfos`** dans un histogramme. En largeur et en hauteur, l'espace disponible sur le canevas est occupé de façon optimale par le graphique. Les rectangles qui représentent les valeurs sont alternativement jaune et orange. ils possèdent une bordure rouge. Les valeurs et les leurs libellés sont affichés en bleu en bas du graphique. Le titre est affiché en haut.
- Afin de pouvoir représenter les nombres sur un canevas sous forme d'un histogramme, il faut ajouter le panneau **`DrawPanel`**.
- Bien entendu il faut que la classe **`DrawPanel`** possède une référence vers une liste de nombres.



Exercice D.8: Squares

En considérant uniquement la copie d'écran ci-dessous (et si nécessaire le modèle de programme fourni par votre professeur), construisez une réplique du programme **Squares** !

Déterminez d'abord les classes à définir, leurs rôles et les éléments qu'elles doivent contenir.



Exercice D.9: Traceur de polynômes

Dans la suite, vous allez développer un programme, qui sert à représenter graphiquement des courbes de polynômes.

Effectuez d'abord une copie du projet **`Polynomial`** tel qu'il a été réalisé dans le chapitre traitant les listes. Sauvegardez le projet sous le nom **`PolyPlotter`** et ouvrez-le en NetBeans.

La classe **`Polynomial`** sait calculer les données à représenter. Nous allons y ajouter la possibilité de se dessiner sur un canevas.

Version 1 : grapheur de base

Pour dessiner la courbe, créez un panneau **`DrawPanel`** qui possède comme attribut un polynôme **`poly`** (type **`Polynomial`**). Un manipulateur **`setPoly(...)`** permet d'affecter un polynôme à **`poly`**. Elle possède une méthode **`paintComponent`** qui efface le canevas en le remplissant par sa couleur de fond, puis fait appel à la méthode **`draw`** de **`Polynomial`** (voir plus bas).

Ajoutez à la classe **`Polynomial`** les attributs suivants :

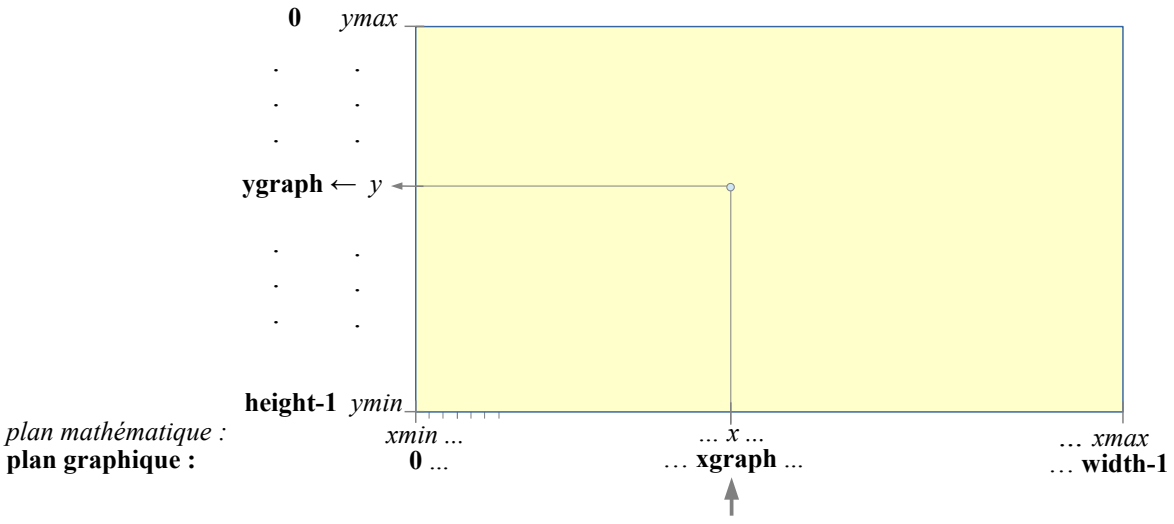
`xmin`	l'abscisse mathématique minimale à représenter (nombre réel)
`xmax`	l'abscisse mathématique maximale à représenter (nombre réel)
`ymin`	l'ordonnée mathématique minimale à représenter (nombre réel)
`ymax`	l'ordonnée mathématique maximale à représenter (nombre réel)

La méthode **`setLimits(...)`** permet d'affecter des valeurs aux attributs **`xmin`**, **`xmax`**, **`ymin`**, **`ymax`**.

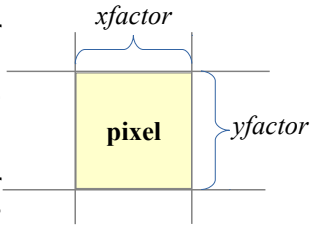
Réalisation de la méthode de dessin draw(...) :

La méthode **`draw`** obtient comme paramètres un canevas **`g`**, les dimensions du canevas (**`pWidth`**, **`pHeight`**) ainsi que la couleur de la courbe.

Pour pouvoir représenter différentes régions du plan mathématique sur le canevas (↔ plan graphique), il est nécessaire de réaliser une transformation entre les coordonnées ***mathématiques*** (**`xmin`**...**`xmax`**; **`ymin`**...**`ymax`**) et les coordonnées ***graphiques*** en pixels (**`0`**...**`width-1`**; **`0`**...**`height-1`**) :



Méthode :



Calculez d'abord la largeur mathématique d'un pixel. Ce facteur de proportionnalité que nous appelons **xfactor** nous aidera à transformer une position horizontale **xgraph** d'un pixel en une valeur **x** mathématique.

Calculez ensuite la hauteur mathématique d'un pixel. Ce facteur **yfactor** nous aidera à la fin à transformer une valeur mathématique **y** en une position verticale **ygraph** d'un pixel sur le canevas.

Parcourez tous les pixels du canevas de gauche à droite avec une variable **xgraph** :

- calculez la valeur **x** mathématique correspondant à la valeur de **xgraph** (Indication : la valeur mathématique **x** correspondant à **xgraph=0** sera **xmin**, ensuite...),
- obtenez la valeur mathématique **y=evalPolynomial(x)** en évaluant le polynôme avec la valeur mathématique de **x**,
- calculez la position graphique **ygraph** correspondant à la valeur mathématique **y**,
- tracez un point aux coordonnées (**xgraph**, **ygraph**).

Comme toujours, la fiche principale (**MainFrame**) joue le rôle du contrôleur. Elle est responsable pour la création et la gestion du polynôme et elle passe les informations nécessaires aux instance de **Polynomial** et **DrawPanel** pour qu'ils puissent dessiner le polynôme correctement à chaque fois que c'est nécessaire. Ajoutez donc une instance de **DrawPanel** à la fiche principale.

Enlevez de la fiche les composants et les lignes de code qui ne jouent plus de rôle pour le dessin (p.ex. le champ texte pour **x** et l'évaluation d'une valeur de **x**).

Ajoutez à l'interface des champs texte pour entrer **xmin**, **xmax**, **ymin** et **ymax.** Faites redessiner la courbe à chaque modification.

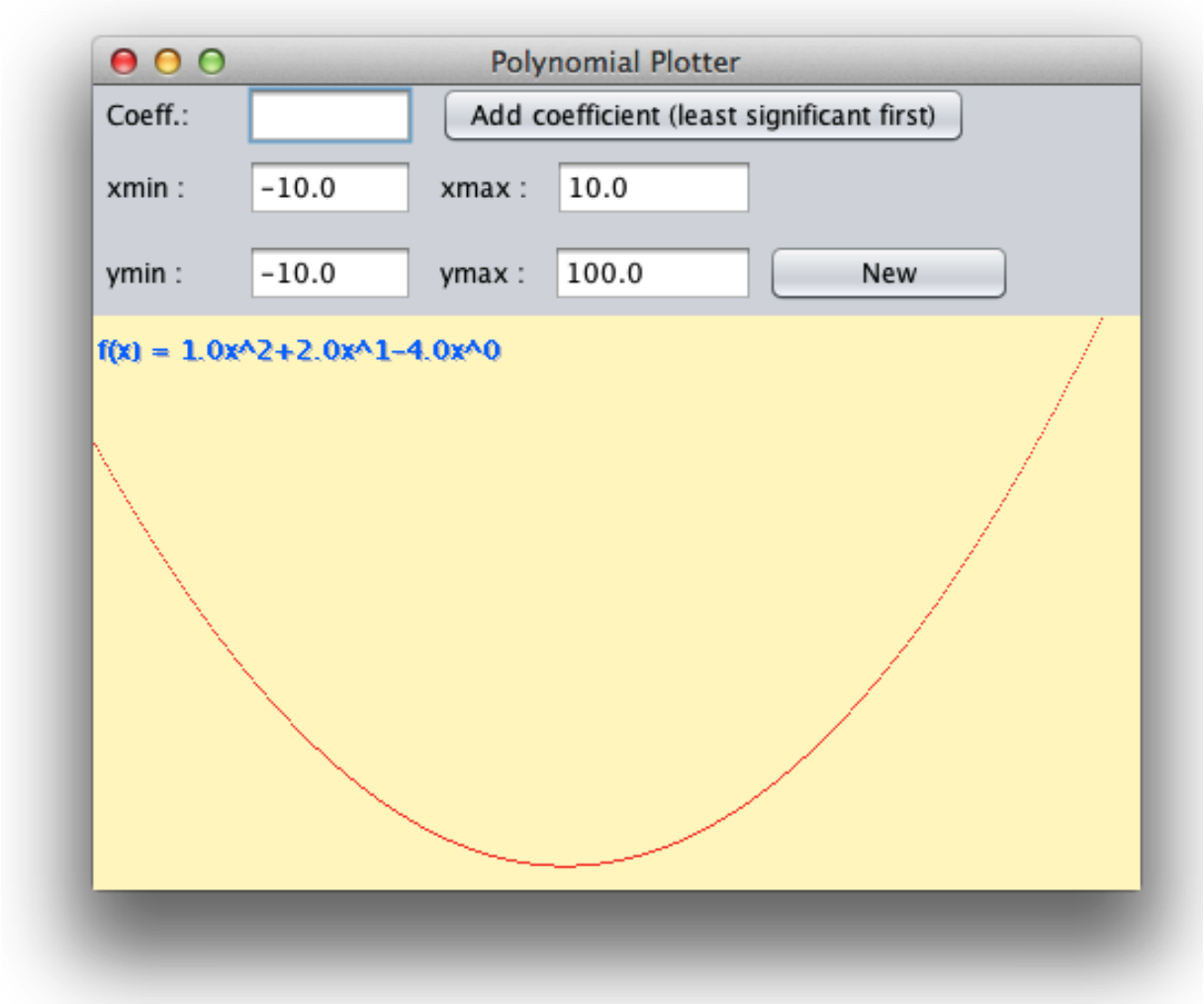
Exemple de la présentation :

Coefficients entrés :

-4,00
2,00
1,00

polynôme : $1 \cdot x^2 + 2 \cdot x^1 + (-4) \cdot x^0$

$= x^2 + 2 \cdot x - 4$


Polynomial Plotter
Coeff.:
Add coefficient (least significant first)
xmin :
-10.0
xmax :
10.0
ymin :
-10.0
ymax :
100.0
New
f(x) = 1.0x^2+2.0x^1-4.0x^0

Traceur de polynômes - Version 2 : Améliorations

1. Affichez l'équation de la courbe (p.ex. en couleur bleue) en haut à gauche sur le panneau (Affichez "f(x) = " devant l'équation de la courbe).
2. Vous voyez bien que la courbe ne remplit pas la surface de dessin de façon optimale. Souvent la courbe dépasse le canevas et parfois aucun point de la courbe n'est représenté. Pour des valeurs **`xmin`** et **`xmax`** données, il faut donc bien choisir **`ymin`** et **`ymax`**. Essayons d'optimiser la représentation en calculant **`ymin`** et **`ymax`** de façon automatique au lieu de les fournir comme paramètres. Comment faut-il choisir **`ymin`** et **`ymax`** pour que nous voyions toujours toute la courbe entre **`xmin`** et **`xmax`** ?
   Enlevez les paramètres **`ymin`** et **`ymax`** de la classe **`Polynomial`** et changez la méthode **`draw`** en conséquence.
3. Employez **`xmin`** et **`ymin`** pour calculer l'endroit à l'écran où se trouvent les axes. Affichez les axes (p.ex en gris clair) avant de dessiner la courbe. Il se peut évidemment que les axes se trouvent à l'extérieur de la surface de dessin. Dans ce cas, il ne faut pas les dessiner.
4. Tracez des lignes au lieu de points pour éviter que la représentation soit discontinue.
5. Si vous ne l'avez pas encore fait, améliorez la méthode **`toString`** de **`Polynomial`** de façon à ce qu'elle n'affiche pas de termes ou de symboles superflus. P.ex. :
   - ne pas afficher de symbole "+" devant des termes négatifs,
   - ne pas afficher de symbole "+" devant le premier terme,
   - ne pas afficher de termes nuls,
   - ne pas afficher le coefficient "1.0*" (si ce n'est pas un terme constant),
   - ne pas afficher "x^0" pour le terme constant
   - ne pas afficher de "^1" pour le terme de degré 1.

Notions requises:	MVC, new, liste, typecast, boucles, coordonnées
Composants requis:	JFrame, JButton, JLabel, JTextField, JPanel
Autres objets:	Graphics

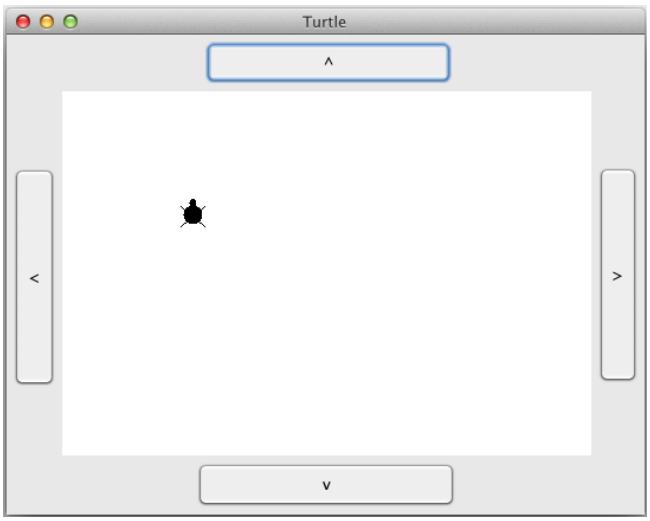
Exercice D.10: Tortue

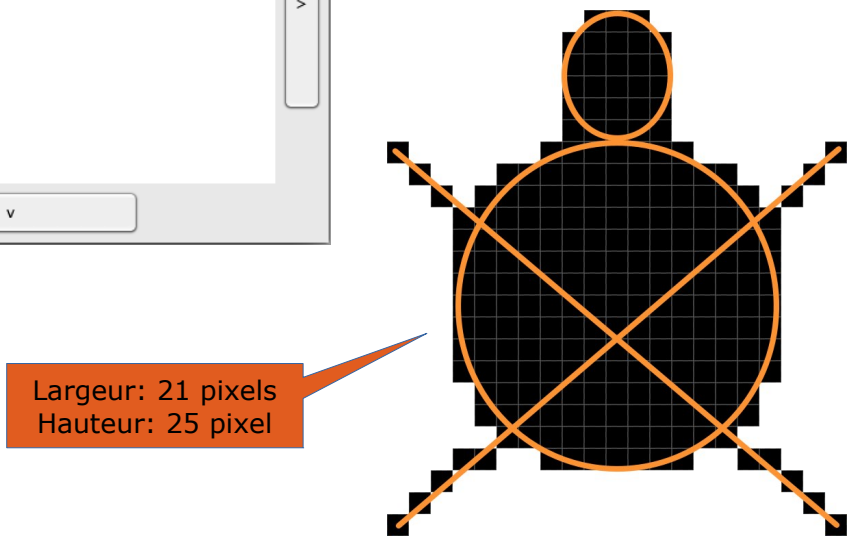
Développez une application permettant de dessiner et de déplacer une tortue à l'écran.

- Le déplacement se fait à l'aide de quatre boutons situés à l'extérieur de la surface de dessin.
- La vitesse de déplacement de la tortue est de 10 pixel par clic.
- La position initiale de la tortue est (100,100).
- Le dessin de la tortue se fait approximativement comme sur le dessin ci-dessous.
- La position de la tortue est celle du centre de la carapace. Elle se dessine elle-même !
- La tortue 'regarde' toujours vers le haut, même si elle se déplace dans une autre direction. La tortue peut se balader aussi en-dehors de la surface visible.

Développer le programme en respectant notre schéma MVC et en employant 3 classes. Déterminez vous-même le rôle des différentes classes. Déterminez ensuite leurs attributs et leurs méthodes ainsi que les interactions entre les objets.

Faites un dessin de préparation (schéma UML) **sur papier** avant de commencer!





Notions requises:	MVC, new, coordonnées
Composants requis:	JFrame, JButton, JPanel
Autres objets:	Graphics

Exercice D.11: RandomStatistics

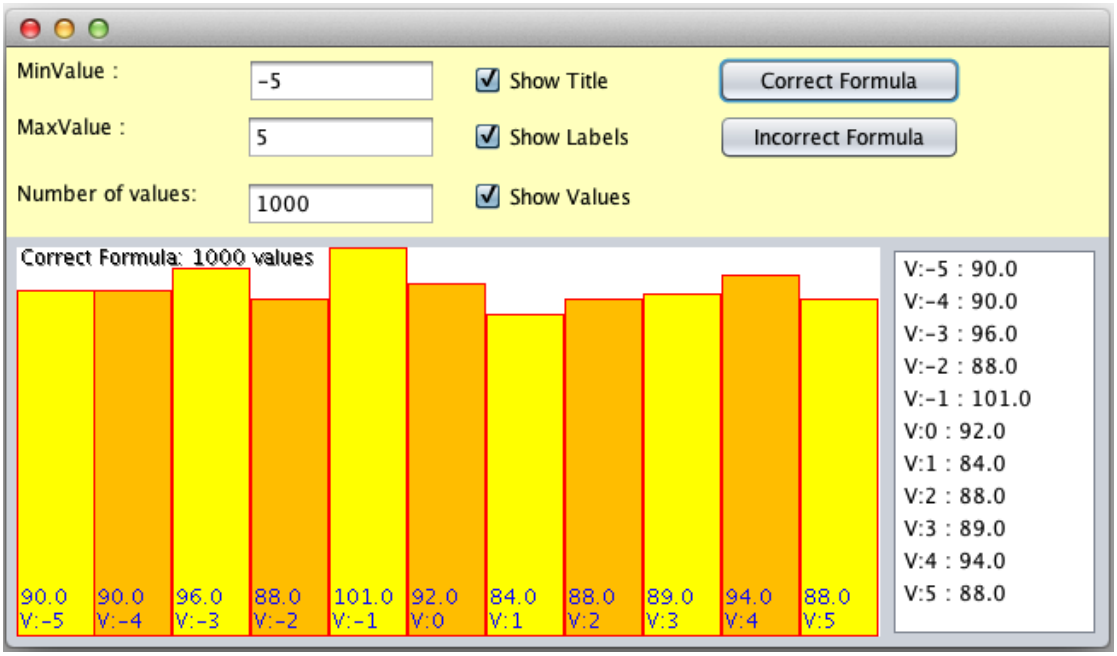
Développez d'abord les classes **`NumberInfo`** et **`NumberInfos`** (Ex. D.7) et les classes **`RandomNumbers`** et **`Randomizer`** (Ex. C.03) pour démontrer à l'aide d'une statistique (liste et histogramme) la différence entre les deux formules :

```
(long)(Math.random()*(max-min+1) + min)          //version incorrecte
```

et

```
(long)(Math.random()*(max-min+1)) + min          //version correcte
```

Représentez le nombre de fois que les différentes valeurs ont été produites dans un histogramme et dans une *JList*.



Méthode :

Ajoutez à **`Randomizer`** une méthode **`getNextIncorrect()`** qui correspond à **`getNext()`**, mais qui emploie la formule incorrecte.

Ajoutez à la classe **`RandomNumbers`** :

- une méthode **`addSeriesIncorrect(...)`** qui correspond à **`addSeries()`**, mais qui fait appel à **`getNextIncorrect()`** au lieu de **`getNext()`**,
- une méthode **`int count(long n)`** qui retourne le nombre de fois que la valeur **`n`** se trouve dans la liste.

Dans **`MainFrame`**, utilisez deux boutons différents pour générer une liste de nombres aléatoires avec la bonne formule et avec la formule incorrecte.

Comptez le nombre de fois que chaque valeur possible a été produite et employez **`NumberInfos`** pour mémoriser le résultat de ce comptage (**`label`** : la valeur produite ; **`value`** : le nombre de fois que la valeur a été produite.) Représenter le contenu de **`numberInfos`** dans la **`JList`** et dans **`drawPanel`**.

Notions requises:	MVC, new, réutilisation de classes définies
Composants requis:	JFrame, JButton, JPanel
Autres objets:	NumberInfo, RandomNumbers, Randomizer

Exercice D.12: Trigonometric Circle

Dans la suite, vous allez réaliser un programme qui sait dessiner un cercle trigonométrique pour un angle donné.

Créez d'abord une classe **Angle** qui mémorise l'angle à représenter en degrés (type double). Les méthodes supplémentaires de la classe **Angle** sont :

Angle constructeur mémorisant un angle (en degrés).

getRadians, **setRadians**	accéder et changer l'angle en radians
getDegrees, **setDegrees**	accéder et changer l'angle en degrés
getSin, **getCos**, **getTan**	retournent le sinus, le cosinus et la tangente de l'angle
draw	dessine le cercle trigonométrique sur un canevas (→ voir détails à la page suivante)

Remarques :

- Les méthodes prédéfinies **Math.sin(double a)**, **Math.cos(double a)**, **Math.tan(double a)** présument que les angles **a** sont exprimés en radians.
- Ajoutez à la fiche principale (**MainFrame**) un glisseur **angleSlider. angleSlider** peut être réglé à des valeurs entières entre -360 et 360. Sa valeur est affichée automatiquement dans le libellé **angleLabel**.
- Une classe **DrawPanel** dérivée de **JPanel** qui possède un attribut **angle**, instance de **Angle**. Ajoutez aussi un manipulateur **setAngle** qui change l'attribut **angle.**
- Placez un panneau **drawPanel** sur la fiche principale. Modifiez la fiche principale pour que chaque modification du glisseur modifie aussi l'angle**.** A chaque modification de l'angle, le panneau est redessiné.

La méthode **draw** de **Angle** réalise les actions suivantes :

- Calculez d'abord les valeurs des variables suivantes et utilisez-les lors du dessin :

centerX, centerY	(double) les coordonnées du centre du panneau
radius	(double) le rayon du cercle trigonométrique. Le rayon correspond au quart (DE : *Viertel*) de la dimension (hauteur ou largeur) la plus petite du panneau.
cos, sin, tan	(double) le sinus, le cosinus et la tangente de l'angle

- Dessinez en blanc le fond du panneau.
- Dessinez en gris clair le cercle et les axes. Le centre du cercle se trouve exactement au milieu du panneau. L'origine de l'axe se trouve exactement au milieu du cercle.
- Le rayon du cercle possède mathématiquement la valeur 1. Le rayon correspond à un quart de la hauteur (ou de la largeur, si celle-ci est plus petite que la hauteur).
- A l'extrémité droite du cercle se trouve une droite parallèle à l'axe des ordonnées (qui servira à montrer la valeur de la tangente.)
- Dessinez l'angle en rouge.
- Dessinez le cosinus en bleu et la ligne d'aide pour le cosinus en gris clair.

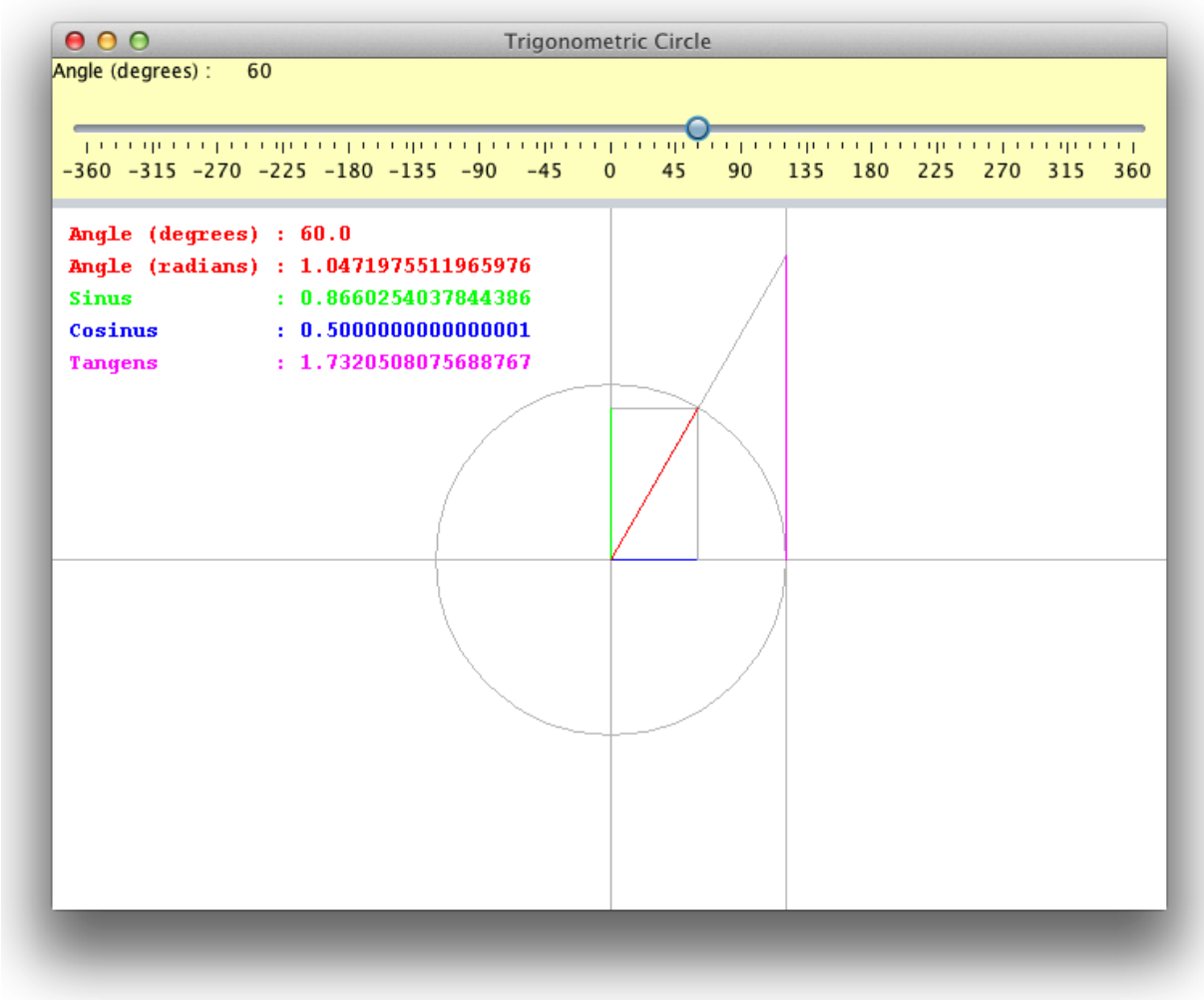
- Dessinez le sinus en vert et la ligne d'aide pour le sinus en gris clair.
- Dessinez la tangente en magenta et la ligne d'aide pour le cosinus en gris clair.

  Si nécessaire, modifiez maintenant la suite des instructions pour que l'angle soit dessiné après la tangente (sinon, l'angle en rouge est couvert par la ligne d'aide de la tangente).
- Affichez sous forme de texte les valeurs suivantes sur le canevas du panneau. Utilisez les couleurs correspondantes au dessin.
- Utilisez l'instruction suivante pour définir une police Courier New lors de l'affichage :

```
g.setFont(new Font("Courier New",Font.BOLD,14));
```

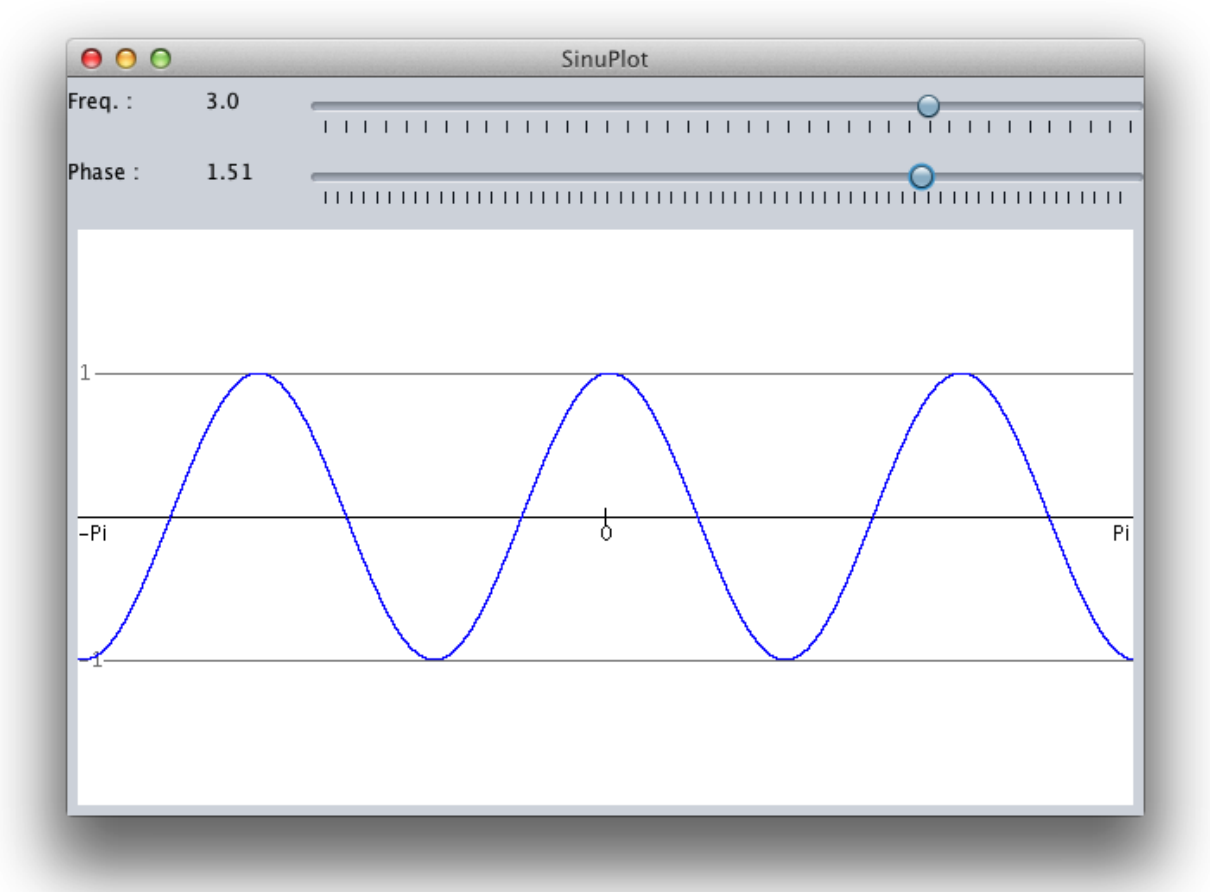
Remarques :

- Pour la représentation graphique il n'y a pas de cas spéciaux à considérer (pas de structures 'if' nécessaires). Les coordonnées découlent directement de quelques calculs simples sur les variables décrites ci-dessus.
- Essayez d'obtenir la meilleure précision possible dans votre dessin.



Exercice D.13: SinuPlot

Remarque : Ce programme est très similaire à l'exercice **D.9 - Traceur de polynômes**. Evidemment la classe modèle (**Polynomial**) est remplacée ici par une classe **Sinusoidal** et l'interface graphique est différent puisqu'on n'a pas besoin de plusieurs coefficients, mais uniquement de la fréquence et du déphasage.



Le programme ***SinuPlot*** représente une sinusoïde sur un graphique. La sinusoïde est donnée par sa fréquence et son déphasage. Le déphasage peut varier entre -Π et Π. La fréquence peut varier entre 0.0 et 4.0.

Les dimensions du plan mathématique sont **fixées** aux valeurs suivantes :

xmin = -Π, xmax = Π, ymin = -2, ymax = 2.

a) Sinusoidal

Créez une classe **`Sinusoidal`** qui possède les attributs suivants (nombres réels) : **`freq`, `phase`.** Les attributs sont initialisés lors de la création, mais la classe possède aussi des manipulateurs (*setter*) pour ces attributs.

La classe **`Sinusoidal`** possède une méthode **`eval(double x)`** qui calcule le sinus (ordonnée **y**) pour un angle (abscisse **x**) donné en radians.

Rappel: Le calcul se fait selon la formule:

$y = \sin(x * f + \varphi)$

x étant un angle (en radians)
f étant la fréquence
φ étant le déphasage (en radians)

La méthode **`draw`** de **`Sinusoidal`** est décrite plus bas.

b) DrawPanel

`DrawPanel` possède comme attributs une sinusoïde **`sinusoidal`** (type **`Sinusoidal`**). Evitez une exception (*NullPointerException)*, si **`sinusoidal`** est **`null`**.

Un manipulateur **`setSinusoidal(...)`** permet d'affecter une sinusoïde à **`sinusoidal`**.

c) méthode draw – dessin de la courbe

Comme pour la représentation d'un polynôme, il est nécessaire de réaliser une transformation entre les coordonnées ***mathématiques*** (**xmin**...**xmax**; **ymin**...**ymax**) et les coordonnées ***graphiques*** en pixels (**0**...**width-1**; **0**...**height-1**).

La méthode **`draw`** de **`Sinusoidal`** trace la sinusoïde sur le canevas **`g`** donné (courbe rouge, fond blanc). La représentation est aussi précise que possible sur le canevas et les limites du plan mathématique (données ci-dessus) correspondent toujours à la totalité du panneau.

Les courbes sont dessinées avec des lignes qui relient les points dessinés (au lieu de point isolés.

d) méthode draw – dessin des axes

Ajoutez une méthode **`drawAxis`** qui est appelée dans **`draw`**. A chaque fois que les sinusoïdes sont retracées, l'axe des abscisses (en noir) avec marquage de l'origine, et les lignes d'aide (en gris) sont retracées. N'oubliez pas les inscriptions '-Pi', '0', 'Pi' pour l'axe des abscisses et '1', '-1' pour les lignes d'aide!

e) MainFrame - Modification des paramètres → `updateView()`

Créez une fiche principale **`MainFrame`** avec un attribut **`sinusoidal`** du type **`Sinusoidal`**.

Placez deux barres de défilement (*JSlider*) sur un panneau. Chaque barre de défilement est précédée par un libellé. Les barres de défilement ***`freqSlider`*** et ***`phaseSlider`*** servent à modifier les paramètres de la sinusoïde de façon simple et pratique. Les limites (***`minimum`*** et ***`maximum`***) de la barre ***`freqSlider`*** sont fixées à des valeurs entre 0 et 400. Les limites de ***`phaseSlider`*** sont fixées à des valeurs entre –314 et 314.

Comme les barres peuvent seulement retourner des nombres entiers (propriété ***`value`***) la valeur retournée est divisée pour obtenir un nombre réel:

- Pour les fréquences ***`freqLabel`*** on a besoin d'une valeur entre 0.0 et 4.0 (au démarrage: Fréquence = 1.0).
- Pour le déphasage ***`phaseLabel`*** (-π ≤ Déphasage ≤ π) de la sinusoïde, on a besoin d'une valeur entre –3.14 et 3.14 (au démarrage: Déphasage = 0.0)

A chaque modification du curseur sur les barres de défilement, les valeurs à utiliser pour les paramètres sont calculées et affichées dans les libellés ***`freqLabel`*** *et* ***`phaseLabel`***. L'attribut **`sinusoidal`** est actualisé et le panneau **`drawPanel`** est redessiné à chaque modification.

Exercice D.14: La formule de Gielis - pour avancés -

La superformule que le hollandais Johan Gielis a publiée en 2003, permet de décrire mathématiquement des formes naturelles ou abstraites très différentes (des cercles, des carrés, des étoiles, des spirales et encore une infinité d'autres figures). On estime que cette formule jouera dorénavant un rôle important dans les logiciels graphiques, car elle permet de décrire des objets complexes à l'aide de quelques paramètres.

Etape 1 - classe *Gielis*

Créez d'abord la classe **Gielis** avec les attributs réels suivants : ***m, n1, n2, n3*** *et* ***maxPhi***. Ces attributs seront initialisés lors de la création et ils possèdent tous un manipulateur (*setter*).

Gielis
− m : double − n1 : double − n2 : double − n3 : double − maxPhi : double − NP : int
+ setMaxPhi(maxPhi : double) : void + setM(m : double) : void + setN1(n1 : double) : void + setN2(n2 : double) : void + setN3(n3 : double) : void − polar2CartX(angle : double, radius : double) : double − polar2CartY(angle : double, radius : double) : double + Gielis(m : double, n1 : double, n2 : double, n3 : double, maxPhi : double) + evaluate(phi : double, a : double, b : double) : double + draw(g : Graphics, width : int, height : int) : void

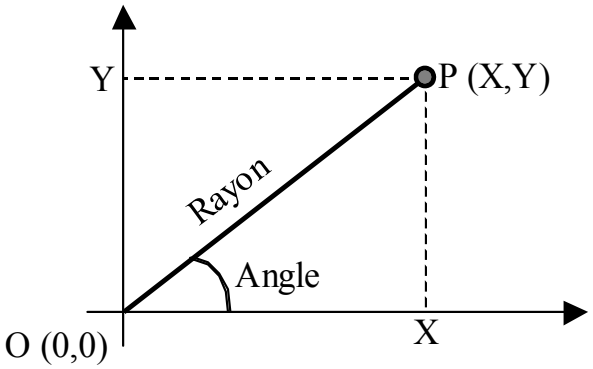
Etape 2 - Gielis : méthode *evaluate*

A l'intérieur de la classe **Gielis** définir la méthode **evaluate** qui calcule et retourne comme résultat un rayon R selon la superformule de Johan Gielis. Le rayon R est calculé à partir des attributs ***m, n1, n2, n3,*** d'un angle **phi** et de deux réels **a** et **b** qui fixent la taille du dessin en horizontale et en verticale. **phi**, **a** et **b** sont passés comme paramètres à la méthode. Le calcul se fait d'après la formule suivante:

$$R = \frac{1}{\sqrt[N1]{\left|\frac{1}{A}\cdot\cos\left(\frac{M}{4}\cdot PHI\right)\right|^{N2} + \left|\frac{1}{B}\cdot\sin\left(\frac{M}{4}\cdot PHI\right)\right|^{N3}}}$$

Etape 3 - Gielis : méthode *polar2Cart*

A l'intérieur de la classe **Gielis** définissez les méthodes privées ***polar2CartX*** et ***polar2CartY*** qui convertissent des coordonnées polaires (***angle, radius***) en des coordonnées cartésiennes. C.-à-d. pour un angle et un rayon donnés, les méthodes ***polar2CartX*** et ***polar2CartY*** calculent les coordonnées x et y correspondantes.



Etape 4 - Gielis : méthode *draw*

Pour chaque figure, nous allons considérer une suite 20000 angles ***phi*** consécutifs entre 0 et ***maxPhi***. Pour chacun des ces angles **phi**, on calcule le rayon ***R*** en utilisant la formule de Gielis. Ainsi on obtient les coordonnées polaires (angle et rayon) de 20000 points qui constituent la figure.

Il suffit de transformer ces coordonnées polaires en coordonnées cartésiennes ***x*** et ***y*** pour tracer la figure.

Indications pratiques:

- Lors du dessin des figures, le point d'origine O(0,0) est supposé se trouver exactement au milieu de la surface de dessin.
- Les paramètres **a** et **b** de la méthode **evaluate** définissent en gros les dimensions de la figure à dessiner. Nous allons prendre la même valeur pour **a** et **b**, notamment la moitié de la plus petite dimension (longueur ou largeur) du panneau.
- Les figures sont dessinées en bleu.

Etape 5 - classe DrawPanel

Créez à partir d'un panneau la classe **DrawPanel** sur laquelle seront représentées les figures.

GielisPanel possède un attribut **gielis** avec un manipulateur.

Sa méthode **paintComponent** efface le contenu du panneau en le remplissant par la couleur de fond du canevas, puis dessine l'objet **gielis** s'il est déjà défini.

Etape 6 - MainFrame : Modification des paramètres

Créez une interface graphique avec les éléments suivants (→ voir aussi le programme

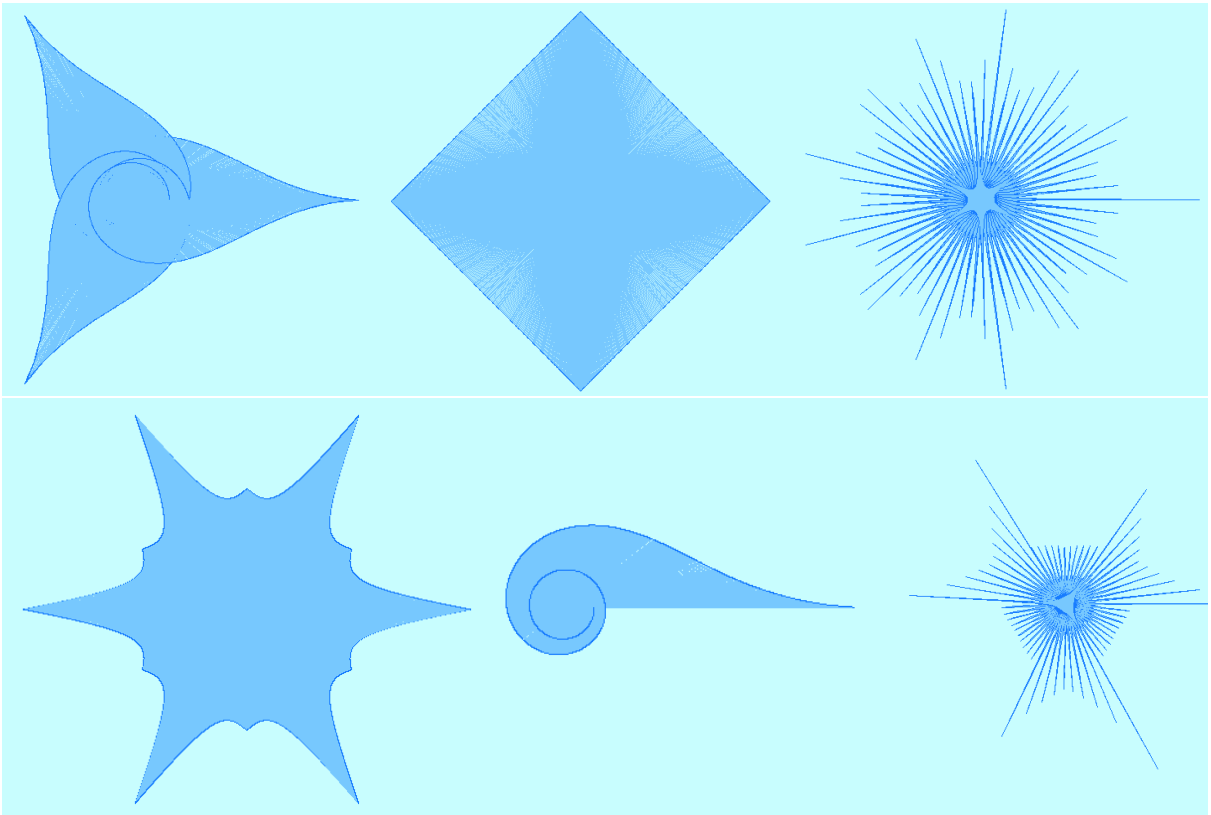
modèle) : la classe possède une instance **gielis** de la classe **Gielis**. Toute la partie inférieure de la fiche est occupée par un panneau **drawPanel** du type ***DrawPanel***.

La méthode **update** synchronise **MainFrame** et les autres classes. Elle a trois charges :

1. passer les valeurs actuelles de l'interface graphique à l'instance **gielis** (détails → voir ci-dessous),
2. actualiser le contenu des libellés : c.-à-d. afficher les valeurs actuelles de ***m, n1, n2, n3, maxPhi*** dans les libellés ***mLabel, n1Label, n2Label, n3Label, maxPhiLabel***,
3. réafficher de dessin.

Les barres de défilement (*JSliders*) ***mSlider, n1Slider, n2Slider, n3Slider*** servent à modifier les paramètres de la formule de Gielis de façon simple et pratique. Les limites (***minimum*** et ***maximum***) de toutes les barres sont fixées à des valeurs entre 0 et 600. Comme les barres peuvent seulement retourner des nombres entiers (propriété ***value***) la valeur retournée est divisée pour obtenir un nombre réel:

- Pour l'attribut **m** (← ***mSlider***) on a besoin d'une valeur entre 0.0 et 60.0
- Pour les attributs **n1, n2, n3** (← ***n1Slider, n2Slider, n3Slider***) on a besoin d'une valeur entre 0.0 et 6.00
- ***maxPhiLabel*** contient l'angle maximal (en radians) pour lequel on va calculer la formule de Gielis. Pour faciliter la saisie, l'angle est entré en tours complets dans ***toursSlider***. L'angle total (2*Pi*tours) est calculée à chaque modification de ***toursSlider*** et mémorisée dans ***maxPhiLabel***.



A chaque modification du curseur sur les barres, il faut réactualiser tous les objets**.**