IC2

TP 2 : Dessins `Swing` et intersections de rectangles

1 Une première fenêtre `Swing`

Exercice 1.1 Construire une première fenêtre Swing avec les propriétés décrites ci-dessous (voir les diapos de la Section 3 du cours)

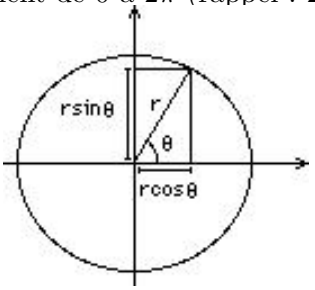
- largeur : 250 pixels
- hauteur : 200 pixels
- position du coin en haut à droite : 100,300 ; utiliser `setLocation(...)`
- la fermeture de la fenêtre (clique sur ×) déclenche l'arrêt immédiat du programme `Ruby`
- la fenêtre contient juste un bouton (classe `javax.swing.Button`) étiqueté `premier bouton`

Exercice 1.2

Continuer le programme principal (l'extérieur de la classe) afin de déplacer la fenêtre sur *une trajectoire circulaire* : utiliser `setLocation(x,y)`. La fenêtre doit suivre une trajectoire décrite par les coordonnées :

$$x = centerX + r \cdot sin(\theta) \text{ et } y = centerY + r \cdot cos(\theta),$$

où θ représente un angle en radians. En fait, θ est incrémenté graduellement de 0 à 2π (rappel : 2π représente un angle de 360 degrés), voir aussi la figure ci-contre.



Par exemple, la trajectoire pourrait être :
$\big(300 + 100 \cdot sin(0 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(0 \cdot \frac{\pi}{12})\big)$,
$\big(300 + 100 \cdot sin(1 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(1 \cdot \frac{\pi}{12})\big)$,
$\big(300 + 100 \cdot sin(2 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(2 \cdot \frac{\pi}{12})\big)$,
...
$\big(300 + 100 \cdot sin(12 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(12 \cdot \frac{\pi}{12})\big)$.

La classe `Graphics`

Nous observons que la méthode `paint(g)` utilise un paramètre `g` pour toute instruction graphique ; `g` représente en fait un objet de classe `Graphics`. Grâce à cet objet, on dispose d'une large gamme de méthodes graphiques qui permettent de tracer des rectangles, des polygones, des cercles, ellipses, etc.

Par exemple, pour écrire un texte, il est possible d'utiliser un code comme :

```
g.setFont(java.awt.Font.new("Arial",java.awt.Font::ITALIC,20));
g.setXORMode (java.awt.Color::RED);
g.drawString("Salut",20,20);
```

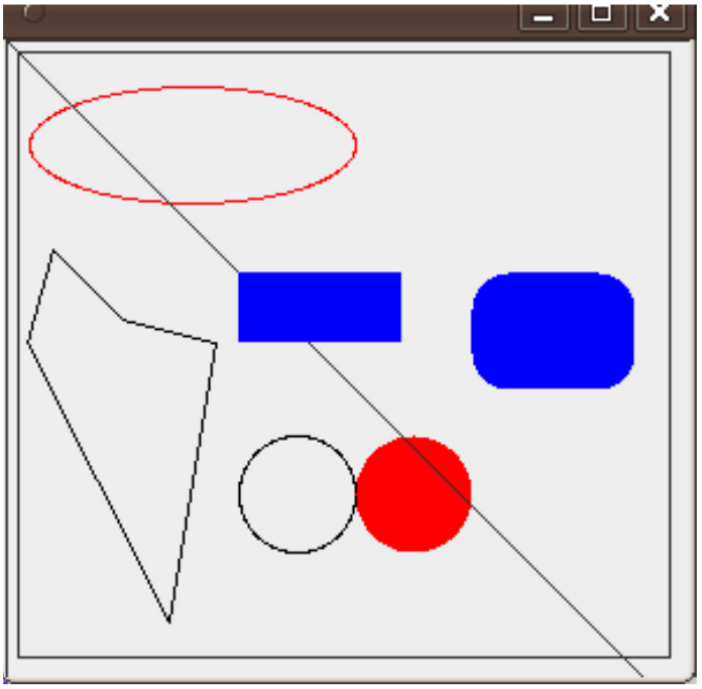
Quel est l'objectif de l'appel `g.setFont(...)` ? Trouvez la documentation de la méthode *setFont* sur Internet. La même question pour `g.setXORMode (...)`. Tester le fonctionnement si vous écrivez ces instructions *avant ou après monDessin.drawString*(...) ; que observez vous ? Pourquoi le dessin n'a pas la couleur rouge ?

Exercice 1.3 Rechercher sur Internet la documentation de la classe Graphics (e.g. taper "java class Graphics" dans votre moteur de recherche préféré) et étudier ses méthodes. Ensuite, trouver les bonnes méthodes à utiliser pour générer un dessin qui contient :

- une ligne
- un rectangle

– un rectangle remplis
– un rectangle avec les coins arrondis
– un cercle (indication : un cercle est une ellipse avec hauteur=largeur)
– un cercle remplis
– une ellipse

Vous pourriez obtenir un dessin similaire à l'image ci-dessous, mais il n'y pas besoin de faire le polytope (cela nécessiterait de transformer les tableaux Ruby en tableaux Java avec `to_java(Java::int)`, ce qui n'est pas essentiel dans ce module).



Faites valider ce programme par l'enseignant.

2 Déplacer un rectangle sur une carte avec des obstacles

Définir une classe `Rectangle` qui fournit une méthode `depl(x,y)` permettant de déplacer le rectangle de x cases/pixels à droite et y à gauches. Ce déplacement peut être effectué uniquement si la nouvelle position du rectangle ne dépasse les limites de taille et n'est pas en conflit (superposition) avec un obstacle.

L'utilisation d'une telle classe est exemplifiée par le programme ci-dessous ; observer les déplacements dans la sortie : les coordonnées sont affichées trois fois.

```
posX = 3
posY = 3
dimX = 4
dimY = 4
monRectangle = Rectangle.new(posX,posY,dimX,dimY)
monRectangle.defTailleCarte(50,10)
monRectangle.ajoutObstacle(48,1)
monRectangle.afficher()
for i in 1..100
    monRectangle.depl(1,0)
end
monRectangle.afficher()
```

```
13 for i in 1..100
14     monRectangle.depl(0,-1)
15 end
16 monRectangle.afficher
```

```
Taille Carte: 50,10. Coord Rectangle:3,3Taille Rectangle:4,4
Taille Carte: 50,10. Coord Rectangle:45,3Taille Rectangle:4,4
Taille Carte: 50,10. Coord Rectangle:45,2Taille Rectangle:4,4
```

3 Déplacer fenêtre en utilisant la classe `Rectangle`

Exercice 3.1 Reprendre le code de la classe `Rectangle` ci-dessus. Ajouter une fenêtre `monCadre` (reprendre le code de l'exercice 1.1 ci-dessus). Le rectangle représente désormais votre fenêtre Swing.

Ajouter dans la classe `Rectangle` une méthode `deplSiPossible(monCadre, x,y)` qui permet de déplacer la fenêtre Swing avec `monCadre.setLocation(...)`. Cette fenêtre se déplace (x pixels à droite et y en bas) *uniquement si* elle peut *rester toute entière dans un écran* de taille 1024× 768 pixels (si la résolution de votre écran est différente, changer cela avec les valeurs disponibles via `Système→Écrans`). La méthode `deplSiPossible(monCadre, x,y)` devrait appeler la méthode `depl(x,y)` qui permet de déplacer le rectangle en évitant de sortir de la carte et en évitant les obstacles.

Indications : Construire un objet `monRectangle` de type `Rectangle`. On observe les correspondances :

- l'objet `Rectangle` est ses coordonnées représentent la fenêtre. Ainsi, l'objet `monRectangle` est construit via `Rectangle.new(fenX, fenY, tailleX, tailleY)` :
  - `(fenX,fenY)` représente la position de la fenêtre définie via `monCadre.setLocation(fenX,fenY)` ;
  - `(tailleX,tailleY)` représente la taille de la fenêtre définie via `monCadre.setSize(tailleX,tailleY)`
- la taille de la carte dans laquelle se trouve le rectangle (indiquée par `defTailleCarte(long,haut)`) représente la résolution de l'écran.

4 Animations 2D

Ajouter dans la classe `Rectangle` une méthode `dessiner(g)` qui dessine (sur l'objet `Graphics g`) le rectangle et les obstacles ajoutés via `ajoutObstacle(positionX,positionY)`. La taille de la toile est passée dans les paramètres de la méthode `defTailleCarte(..., ...)`.

Indications :

- Utiliser l'objet `monRectangle` comme une variable globale, i.e, le nom de la variable devrait commencer par `$`. La méthode `paint(g)` appelle `$monRectangle.dessiner(g)`.

Pour déplacer le rectangle dans la fenêtre sur la même trajectoire que celle du dernier programme du TP 1, vous pouvez utiliser un code comme, par exemple :

```
#Section 1: la classe Rectangle
class Rectangle
    @xObst;                 #liste d'obstacles (les position x)
    @yObst;                 #liste d'obstacles (les position y)
    @larg;@haut             #limites de la carte en largeur/hauteur
    @x1;@dimX; @y1;@dimY;#coordonnées du rectangle principal
    .....
    les autres methodes du TP1 (utiliser votre implémentation du TP 1), voir aussi le TD
    .....
    def dessiner(g)
        g.fillRect(@x1,@y1,@dimX,@dimY)
        .....
        dessiner les obstacles
        ....
```

```
    end
end

#Section 2. les objects Swing
require "java"
import javax.swing.JFrame
class Toile<java.awt.Canvas
    def paint(g)
        $monRectangle.dessiner(g)
    end
end
dimToileX = 500                                  #ou bien votre taille de toile
dimToileY = 500
maToile = Toile.new()
maToile.setPreferredSize(java.awt.Dimension.new(dimToileX,dimToileY))
frame = JFrame.new("Titre Frame");
frame.getContentPane().add(maToile)
......
appeler les autres methodes (ex. rendre le ``frame'' visible, voir le cours)
......
#Section 3:  manipulation du rectangle dans la fenetre
$monRectangle = Rectangle.new(150,100,50,50) #coordonnées rectangle
$monRectangle.defTailleCarte(dimToileX,dimToileY)
$monRectangle.ajoutObstacle(380,40)
for i in 1..100
    $monRectangle.depl(10,0)
    maToile.repaint()                #repaint() appelle la méthode paint(g) de maToile,
                                     #qui appelle ensuite $monRectangle.dessiner(g)
    sleep(0.2)
end
for i in 1..100
    $monRectangle.depl(0,-10)
    maToile.repaint()
    sleep(0.2)
end
```