<Trajectoire>

Principe :

Une interface (comme une classe abstraite mais sans constructeur) que les 3 autres classes implémentent. Ça sert à créer un super type pour les autres classes, c'est-à-dire que <Line> <PolyLine> et <Function> sont aussi des objets Trajectoire, je peux donc faire Trajectoire t = new Line() ;

<Trajectoire> hérite de <Cloneable>, qui est utilisée pour cloner des objets, les classes qui implémentent <Trajectoire> vont aussi hériter de <Cloneable>

Note : on ne peut pas utiliser 2 fois la même <Trajectoire> à la suite, car certains de ses attributs changent lorsqu'elle est parcourue, il faut donc cloner la <Trajectoire> avant de la parcourir si on veut en avoir 2.
Note 2 : les getters & setters ne sont pas décrits
Note 3 : Pourquoi j'utilise beaucoup le mot-clef final ? l'utilisation du mot-clef final quand c'est possible améliore les performances

Méthodes :

- *next(JComponent):void*

Fonction abstraite qui devra être redéfinie par toutes les classes qui implémentent <Trajectoire>
C'est elle qui permet de faire avancer le JComponent

<Line>

Principe :

Objet ligne défini à partir de 4 coordonnées (2 points) rentrées en % (double entre 0 et 100) de la taille du conteneur contenant le JComponent à déplacer.
Ex : Line l = new Line(0,0,100,100) ; crée la diagonale du coin haut gauche au coin bas droit.

Méthodes :

- *previous(JComponent):void*

Permet de reculer (décrémente r et appelle move()), méthode sans doute inutile

- *next(JComponent):void*

Permet d'avancer (incrémente r et appelle move())

- *move(JComponent):void*

On considère la ligne comme un rayon qui varie, on calcule l'angle de la ligne avec l'horizontale grâce à atan2(), on projette le point de rayon r sur les axes x et y, on ajoute à ces coordonnées les coordonnées du point de départ. Ces coordonnées sont en %, il faut donc les diviser par 100 et les multiplier par la longueur de l'axe moins la longueur de l'image.

Finalement on place le JComponent à ces nouvelles coordonnées.

Si la distance parcourue est supérieure à la longueur max de la ligne, le booléen terminus devient vrai pour indiquer que la ligne a été parcourue entièrement.

- *hasEnded()*

Renvoie vrai si la ligne a été parcourue entièrement.

- *getReverse():Line*

Renvoie une nouvelle ligne avec point de départ et d'arrivée inversés

- *reset():void*

Remet l'attribut r à 0 et terminus à false (on peut alors réutiliser la ligne)

- *getShifted(double, double):Line*

renvoie une nouvelle ligne déplacée selon la distance indiquée en x et y en %

- *clone():Line*

Pour que le code reste cohérent on utilise la fonction clone de la classe <Object> pour cette opération. Renvoie une nouvelle ligne identique à this.

<PolyLine>

Un tableau de plusieurs lignes, la méthode next() permet de déplacer un JComponent suivant ce tableau.

Trois constructeurs :

Le … permet de passer un nb indéfini d'arguments dans un constructeur ou une méthode (à mettre en dernier dans la liste d'arguments → on ne peut l'utiliser qu'une fois), ces arguments sont gérés comme un tableau.

un qui prend un tableau de lignes,
un qui prend un tableau de points (n+1 pts → n lignes),
un qui prend un tableau de coordonnées x et un tableau de coordonnées y (n+1 couples de coor → n lignes)

Pour les 2 derniers constructeurs on vérifie qu'il y a assez d'arguments pour tracer au moins une ligne, si non on lance une exception.

Méthodes

- *next(JComponent):void*

Permet d'avancer : appelle la méthode next() de <Line> sur la ligne d'indice <cur> dans le tableau <lines>, si la ligne est finie on passe à la ligne suivante en appelant moveLine()

- *moveLine():void*

On remplace la ligne qui s'est terminée par une nouvelle ligne identique dont le x_start est au x_end de la dernière ligne : permet la répétition périodique du motif créé par le tableau <lines>
On passe à la ligne suivante en incrémentant le curseur de 1

- *getSymmetric():PolyLine*

Renvoie une PolyLine symétrique par rapport à X=50 %
On parcourt le tableau <lines> et pour chaque lignes on construit son symétrique qu'on ajoute à un nouveau tableau de lignes. Ce tableau est utilisé pour construire une nouvelle PolyLine qu'on renvoie : c'est le symétrique de this.
Le symétrique d'un point est construit par rapport à l'axe X=50 de la façon suivante :
x_sym = X+distance_x_à_l'axe_X = 50 + (50-x) = 100-x

- *getShifted(double, double):PolyLine*

Renvoie une nouvelle PolyLine déplacée de x, y en %
On construit une nouvelle PolyLine en appelant getShifted() sur chaque <Line> de <lines> et on la renvoie.

- *randomPolyLine(int, double):PolyLine*

Méthode static à appeler en faisant <NomDeClasse>.<nomMethode>
Renvoie une <PolyLine> aléatoire de <n> <Line> et dont le motif s'étend sur une <period>

*Pour avoir un motif sur une <period> on crée un X1 aléatoire compris entre 0 et period*0.75 (on limite la taille à 75 % de <period> pour ne pas avoir une ligne qui occupe tout le motif) puis un X2 compris entre X1 et X1+(period-X1)*0.75 etc, on s'arrête à X(N-1) et on affecte à XN la valeur period. Ici sum est la somme des X(k).*

Pour Y il faut que la fin d'une ligne soit à la même hauteur que le début de la ligne suivante (sinon on dirait que la cible se téléporte) → si k est pair y(k) = y(k-1) si non on lui affecte une valeur aléatoire entre 0 et 100. Comme le signal est périodique on fait en sorte que y(0)=y(n)

- *randomPolyLine(int):PolyLine*

Méthode static à appeler en faisant <NomDeClasse>.<nomMethode>
Pareil que la méthode ci-dessus mais la <period> est choisie aléatoirement entre 10 et 100

- *clone():PolyLine*

Renvoie une nouvelle <PolyLine> identique à this. La méthode clone de la classe Object clone les objets par référence (c'est-à-dire que l'attribut A du clone et l'attribut A de this pointent vers la même adresse, contrairement aux types primitifs (int, double, cas particulier du String,..) qui sont clonés par valeur). Il faut donc cloner « à la main » nos <Line>. Elles n'ont que des types primitifs pour attributs, on appelle donc la méthode clone() de <Line> sur chaque lignes de <lines>.

<Function>

Principe

Permet de déplacer un JComponent selon une fonction mathématique
La description des fonctions disponibles est dans le commentaire en haut, en voici la liste actuelle :

SIN, COS (vous connaissez)
ABS_SIN, ABS_COS (permet de faire quelque chose qui s'approche d'un demi-cercle)
LN, EXP, POW (plus difficile d'obtenir un truc joli avec ça, mais l'exp est marrante car elle commence lentement et accélère)

Constructeur : coordonnées x, y indiquent l'origine de la <function>, amp (coefficient multiplicateur de la fonction), period : effectivement la période pour les fonction périodiques, String <function> même s'il est possible de taper directement "sin" il vaut mieux utiliser Function.SIN

Par défaut le pas sur x vaut 0.01 % si x_start ≤ 50 % sinon -0.01

? est l'opérateur ternaire :
step = (x_start<=50)? 0.01:-0.01; équivaut à

```
if(x_start<=50)
        step = 0.01 ;
else
        step = -0.01 ;
```

Méthodes

- *next(JComponent)*

incrémente x de <step> puis apelle move(), permet de faire bouger le JComponent

- *reverse()*

inverse le signe de <step>

- *move(JComponent)*

méthode privée appelée par next() elle calcule la valeur de x et y à partir d'un switch case sur <function> et appelle la méthode privée moveMode()

- *moveMode(JComponent, int, int)*

Et si on voulait que JComponent suive non plus la trajectoire (X,Y) mais (Y,X) ? c'est possible grâce à cette méthode qui fait un switch case sur <mode> (XY par défaut, on peut le changer avec setMode(), en passant Function.YX ou Function.YY en param par ex) NB : en fonction du x_start et y_start choisi l'inversion peut faire sortir la cible de l'écran… certains modes avait l'air peu utiles (sortie d'écran hautement probable) donc je les ai mis en commentaire.

- *getSymmetric()*

Renvoie le symétrique de la <function> si elle est périodique et la fait se déplacer dans le sens inverse, si elle n'est pas périodique il ne s'agit plus du symétrique, elle commence juste à un point symétrique à x_start et se déplace dans le sens inverse, par ex exp va rester constant car exp(<nb_négatif>) tend vers 0 on fait donc y_start+0.

- *Clone()*

Renvoie le clone de la <function>, tous ses attributs sont primitifs, on se contente donc d'appeler super.clone()