|  |
| --- |
| Université de Thiès |
| Projet01 |
| Disponible sur Github |

|  |
| --- |
| Mohamed Fily DIARRA  20/03/2019 |

# **RAPPORT**

# **PROJET 1 DU COURS DE DESIGN PATTERN 2019**

[RAPPORT 1](#_Toc3936456)

[PROJET 1 DU COURS DE DESIGN PATTERN 2019 1](#_Toc3936457)

[Introduction : 1](#_Toc3936458)

[Diagramme de classe : 1](#_Toc3936459)

[Capture 1 : 2](#_Toc3936460)

[Capture 2 : 3](#_Toc3936461)

[Capture 3 : 4](#_Toc3936462)

[Capture 4 : 6](#_Toc3936463)

[Implémentation de méthodes : 7](#_Toc3936464)

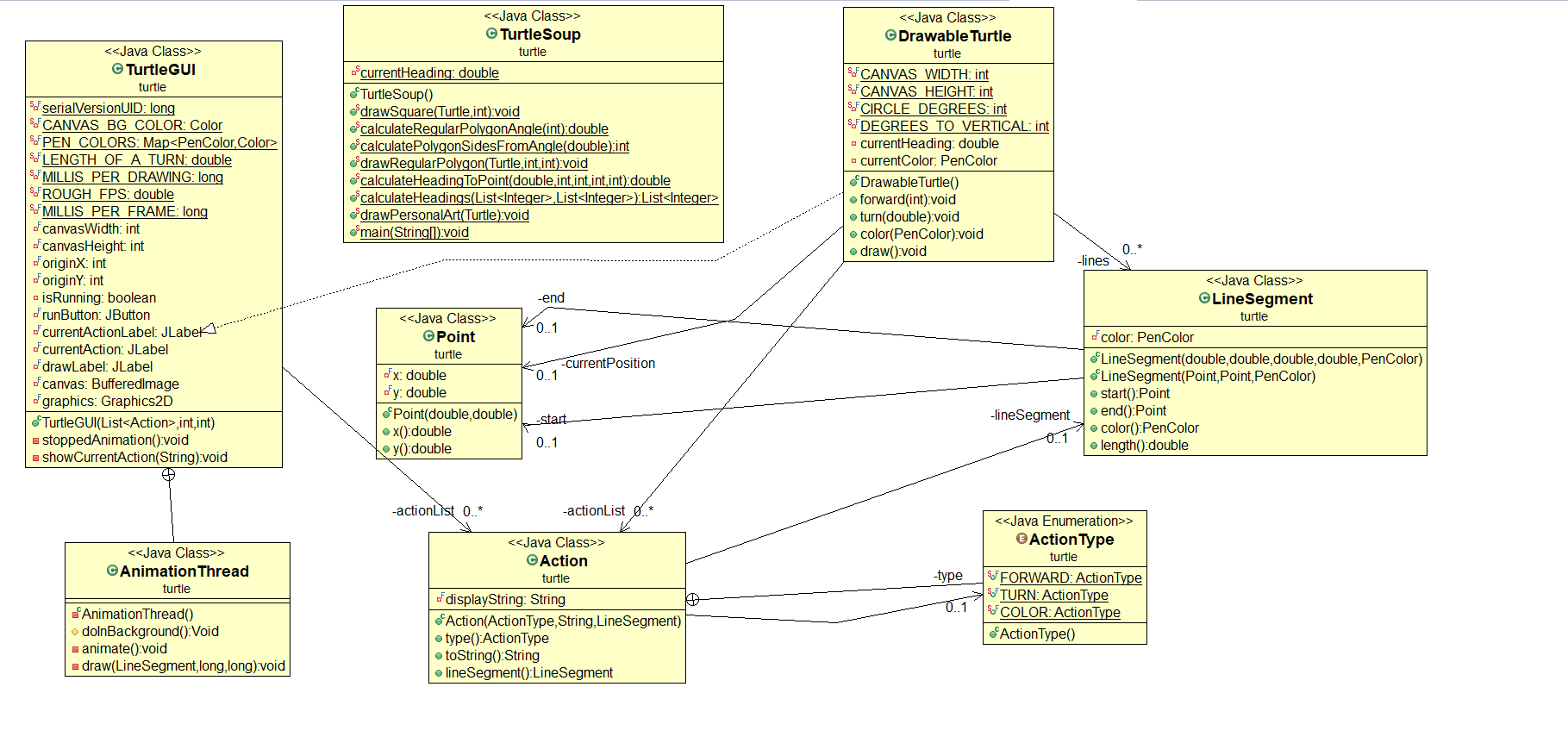
[Conclusion: 8](#_Toc3936465)

# Introduction :

Ce présent document représente le rapport de mon premier projet du cours de design pattern de M. Bousso. Il met l’accès sur l’explication et la description des différentes captures d’écran sur le rendu des polygones à travers du code java.

L’ensemble des tests sont effectués dans la classe TurtleSoup.java.

# Diagramme de classe :



# Capture 1 :

Une image contenant capture d’écran

Description générée automatiquement

Cette capture contient un carré avec des couleurs différentes sur les côtés. Elle est générée par la méthode statique suivante :

**public** **static** **void** drawSquare(Turtle turtle, **int** sideLength) {

//throw new RuntimeException("implement me!");

**for** (**int** i = 0; i < 4; i++) {

turtle.forward(sideLength);

turtle.turn(90);

**if**(i % 2 == 0)

turtle.color(PenColor.***RED***);

**else**

turtle.color(PenColor.***GREEN***);

}

}

# Capture 2 :

Une image contenant capture d’écran

Description générée automatiquement

La deuxième capture représente, en fait, un polygone régulier ayant onze côtés.

Pour calculer un angle interne du polygone régulier, on établit la relation : 360/ns où n étant le nombre de côtés.

Voici le code source qui permet d’afficher la figure :

**public** **static** **double** calculateRegularPolygonAngle(**int** sides) {

//throw new RuntimeException("implement me!");

**double** angle = *CIRCLE\_DEGRES* / sides;

**return** angle;

}

**public** **static** **void** drawRegularPolygon(Turtle turtle, **int** sides, **int** sideLength) {

//throw new RuntimeException("implement me!");

**double** angle = *calculateRegularPolygonAngle*(sides);

System.***out***.println("Sides : " + sides);

System.***out***.println("Angle : " + angle);

**for** (**int** i = 0; i < sides; i++) {

turtle.forward(sideLength);

turtle.turn(angle);

**if**(i % 2 == 0)

turtle.color(PenColor.***ORANGE***);

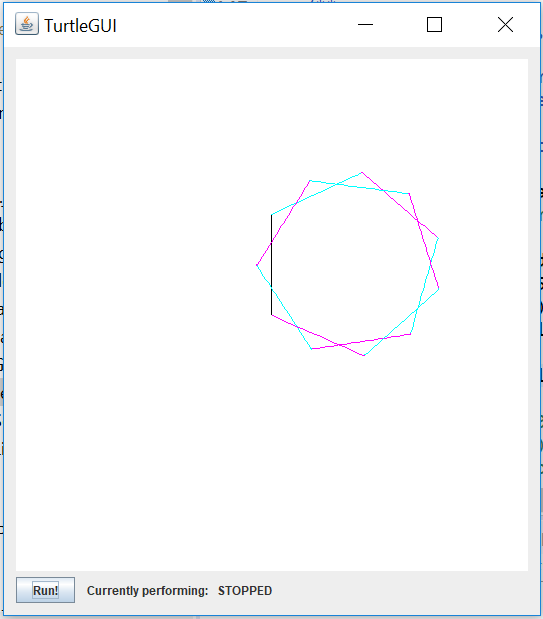
**else**

turtle.color(PenColor.***CYAN***);

}

}

# Capture 3 :



Comme demandé dans l’exercice, la troisième figure est mon œuvre personnel. Elle contient plusieurs polygones qui se coupent.

**public** **static** **void** drawPersonalArt(Turtle turtle) {

//throw new RuntimeException("implement me!");

**for** (**int** i = 0; i < 11; i++) {

turtle.forward(100);

turtle.turn(65.35);

**if**(i % 2 == 0)

turtle.color(PenColor.***CYAN***);

**else**

turtle.color(PenColor.***MAGENTA***);

}

}

# Capture 4 :

Une image contenant capture d’écran

Description générée automatiquement

Concernant la quatrième capture, c’est une combinaison du carré, du polygone et du polygone personnel.

*drawSquare*(turtle, 60);

*drawRegularPolygon*(turtle, 6, 60);

*drawPersonalArt*(turtle);

// draw the window

turtle.draw();

# Implémentation de méthodes :

La fonction ci-dessous retourne une liste d’ensemble de couple d’abscisses et d’ordonnés.

* **public** **static** List<Integer> calculateHeadings(List<Integer> xCoords, List<Integer> yCoords) {

//throw new RuntimeException("implement me!");

/\*\*

\* Création d'une instance de List et combinaison des collections

\* d'abscisses et ordonnés dans results

\*/

List<Integer> results = **new** ArrayList<>();

results.addAll(xCoords);

results.addAll(yCoords);

**return** results;

}

Appel de la méthode calculateHeading dans le main ;

/\*\*

\* **@author** fily

\*/

List<Integer> xpoints = **new** ArrayList<>();

List<Integer> ypoints = **new** ArrayList<>();

xpoints.add(0);

xpoints.add(1);

xpoints.add(1);

xpoints.add(2);

ypoints.add(0);

ypoints.add(1);

ypoints.add(2);

ypoints.add(2);

List<Integer> results = *calculateHeadings*(xpoints, ypoints);

System.***out***.println("Taille list point(x, y) : " + results.size());

Méthode retournant un segment de taille fixe.

* **public** **static** **double** calculateHeadingToPoint(**double** currentHeading1, **int** currentX, **int** currentY,

**int** targetX, **int** targetY) {

//throw new RuntimeException("implement me!");

*currentHeading* = currentHeading1;

LineSegment lineSeg = **new** LineSegment(currentX, currentY, targetX, targetY, PenColor.***GREEN***);

lineSeg.start();

lineSeg.end();

**return** lineSeg.length();

}

Méthode qui retourne un angle en degré.

* **public** **static** **double** calculateRegularPolygonAngle(**int** sides) {

//throw new RuntimeException("implement me!");

**double** angle = *CIRCLE\_DEGRES* / sides;

**return** angle;

}

# Conclusion:

Ce projet nous a permis d’appréhender le design pattern et de mettre à jour nos connaissances sur la programmation orientée objet en particulier Java.