TP - IHM simple

version 2.0

Table des matières

1 Introduction **2**
- 1.1 Objectifs . . . 2
- 1.2 Application . . . 2

2 La classe principale **3**

3 Objectif 1 : création d'une première fenêtre **4**
- 3.1 Votre première fenêtre . . . 4
- 3.2 Interaction avec le bouton . . . 5
- 3.3 Bonus : ajout d'un logo . . . 5

4 Objectif 2 : Allons plus loin **6**
- 4.1 Ajout d'un second bouton . . . 6
- 4.2 Gestion des erreurs de saisie . . . 6
- 4.3 Bonus : affichage d'un texte sur plusieurs lignes . . . 7

5 Objectif 3 : Multi fenêtres et graphisme de base **8**
- 5.1 Création d'une seconde fenêtre . . . 8
- 5.2 Dessinons dans notre nouvelle fenêtre . . . 8
- 5.3 Dessinons maintenant la courbe choisie par l'utilisateur . . . 8
- 5.4 Effaçons la zone de dessin . . . 10

6 Objectif 4 : Gestion du temps et animation **10**
- 6.1 Gérer le temps . . . 10
- 6.2 Animation basique . . . 11
- 6.3 Bonus : animation déclenchée à la souris . . . 11

7 Objectif 5 - Personnalisez votre programme **11**
- 7.1 Interaction avec la souris . . . 11
- 7.2 Interaction avec le clavier . . . 11
- 7.3 Utilisation d'un `JComboBox` . . . 12
- 7.4 Mélangeons les couleurs . . . 12
- 7.5 Récupérons les courbes à partir d'un fichier texte . . . 12
- 7.6 Dessinons dans une zone dédiée de la fenêtre . . . 12
- 7.7 Plaçons de manière automatique les différents éléments de votre fenêtre . . . 12

Avertissement :

— Ce document est en version 2.0. Merci de signaler toutes erreurs rencontrées.

1 Introduction

1.1 Objectifs

L'objectif principal est de découvrir les principes de bases de la construction d'un programme avec une IHM (Interface Homme Machine) sous forme de fenêtre. En effet au lieu d'afficher vos résultats dans un terminal, il est possible de les voir dans une fenêtre. Dans un premier temps, la fenêtre créée ne gérera que l'affichage. Aucune interaction ne sera faite avec celle-ci. Ensuite, il sera possible d'interagir avec et enfin de dessiner des figures et de les animer. La figure 1 présente un exemple de fenêtres que vous serez capables de créer à la fin de ce TP.

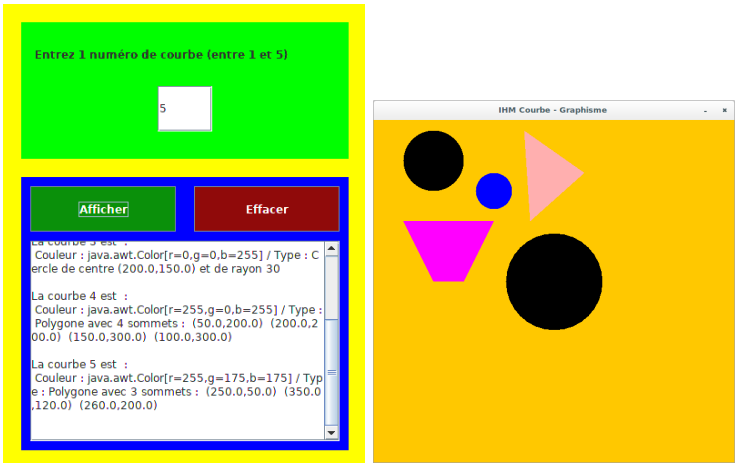

FIGURE 1 – Apparence visuelle de l'application

1.2 Application

Pour simplifier la partie programmation, on s'appuiera sur le code réalisé lors du TP précédent (TP Courbe). Pour la partie POO, vous devez utiliser le code fourni sous Moodle à cette adresse. Il correspond au diagramme UML présent sur la figure 2.

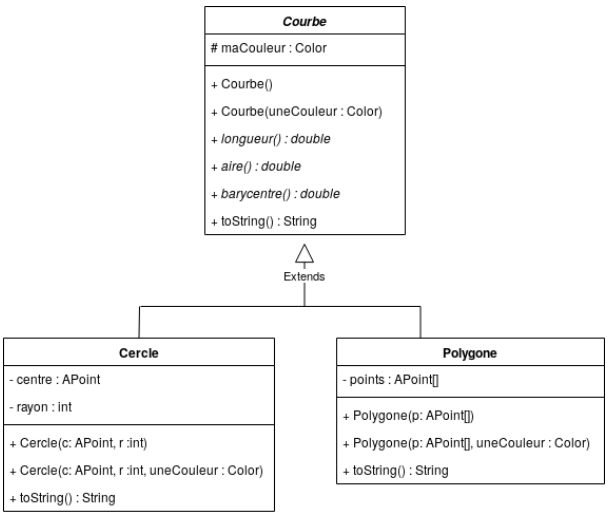

FIGURE 2 – Diagrammes UML des classes `Courbe`, `Cercle` et `Polygone`

2 La classe principale

La classe principale, nommée `GestionCourbe` et contenant la méthode `main`, vous est donnée ci dessous. Cette classe est déjà inclus dans le code fourni sous Moodle. Elle permet de créer 5 courbes (3 cercles et 2 polygones) et de les stocker dans un tableau.

```
/**
 * Classe principale pour gerer differents types de courbe et leur couleur
 *
 * Permet de creer une fenetre pour gerer la liste de courbes
 */

import java.awt.Color;

public class GestionCourbe{

   public static void main(String[] args){

      // Création d'un tableau de courbes
      Courbe[] tabCourbe=new Courbe[5];

      // création de 3 cercles
      Courbe c1 = new Cercle(new APoint(100,100), 50);
      Courbe c2 = new Cercle(new APoint(300,300), 80);
      Courbe c3 = new Cercle(new APoint(200,150), 30, Color.blue);

      // Création de 2 polygones
      APoint[] p1 = new APoint[4];
      p1[0] = new APoint(50, 200);
      p1[1] = new APoint(200, 200);
      p1[2] = new APoint(150, 300);
      p1[3] = new APoint(100, 300);
      Courbe poly1 = new Polygone(p1, Color.magenta);

      APoint[] p2 = new APoint[3];
      p2[0] = new APoint(250, 50);
      p2[1] = new APoint(350, 120);
      p2[2] = new APoint(260, 200);
      Courbe poly2 = new Polygone(p2, Color.pink);

      // Remplissage du tableau par les courbes créées auparavant
      tabCourbe[0]=c1;
      tabCourbe[1]=c2;
      tabCourbe[2]=c3;
      tabCourbe[3]=poly1;
      tabCourbe[4]=poly2;

      // Création de la fenêtre pour l'IHM
      FenetreSelectionCourbe maFrameSelectionCourbe = new FenetreSelectionCourbe(tabCourbe);
   }
}
```

Remarque. **La classe `GestionCourbe` ci-dessus est sous sa forme définitive, vous n'avez pas besoin de la modifier** (sauf pour créer/modifier/supprimer des courbes).

Observez la dernière ligne du code. D'après votre cours, celle-ci permet de créer une fenêtre. Dans un premier temps, cette fenêtre va permettre à l'utilisateur d'afficher des informations liées à une courbe choisie. Ensuite, nous verrons comment dessiner ces figures. Voici un descriptif de ce qui sera fait dans le cadre de ce TP :

1. **Objectif 1 : création d'une première fenêtre**

(a) être capable de placer différents widgets (`JLabel`, `JTextField` et un `JButton`) ;
(b) permettre l'interaction entre l'utilisateur et votre programme suite à l'appui sur le bouton.

2. **Objectif 2 : fonctionnalités (un peu) plus avancées**
   (a) gérer plusieurs boutons ;
   (b) être capable de prévenir des erreurs suite à une erreur de saisie par l'utilisateur.
3. **Objectif 3 : faire du graphisme**
   (a) faire un programme multi fenêtré ;
   (b) dessiner des courbes prédéfinies.
4. **Objectif 4 : gérer le temps**
   (a) être capable de mettre en place un `Timer` ;
   (b) mettre en place une animation.
5. **Objectif 5 : personnalisation de votre programme**
   (a) gérer du texte sur plusieurs lignes ;
   (b) interaction avec la souris et/ou le clavier ;
   (c) faire un menu déroulant ;
   (d) gérer un fichier texte ;
   (e) utiliser un `JPanel` personnalisé.

3 Objectif 1 : création d'une première fenêtre

3.1 Votre première fenêtre

Modifiez la classe `FenetreSelectionCourbe` en ajoutant le code nécessaire afin de respecter le cahier des charges suivant :

1. Avoir une taille de 400x400 ;
2. Être placée aux coordonnées (300,200) sur votre écran ;
3. Utiliser trois conteneurs `JPanel` :
   — le premier contenant 1 `JLabel` et 1 `JTextField` ;
   — le second contenant 1 `JButton` et 1 `JTextField` ;
   — le troisième est le conteneur principal qui contient les deux `JPanel` précédents.

Votre fenêtre devra ressembler à celle présentée sur la figure 3.

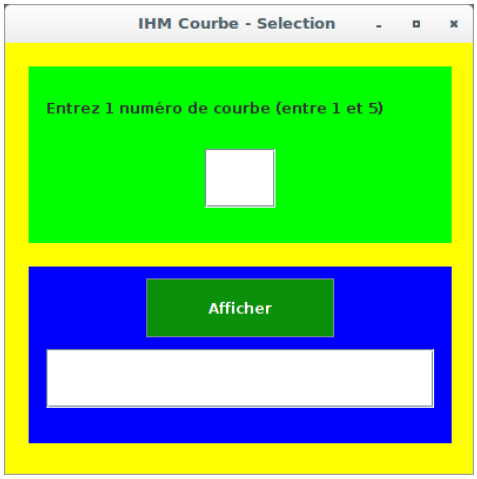

FIGURE 3 – Apparence visuelle de l'application

Remarque. **N'hésitez pas de vous aider d'un schéma sur une feuille de papier pour définir correctement les coordonnées de chaque composant.**

3.2 Interaction avec le bouton

Pour l'instant, lors d'un *click* sur le bouton `Afficher`, rien ne se passe. Il faut permettre à l'utilisateur d'interagir avec notre fenêtre en réagissant à l'événement du *click* sur le bouton.

1. Affichez un message dans le terminal lorsque l'utilisateur appuie sur le bouton `Afficher`.
2. Complétez votre code afin de récupérer le numéro entré par l'utilisateur dans le premier `JTextField`. Affichez ce numéro dans le terminal.
3. Affichez l'information de la courbe correspondante à ce numéro dans le second `JTextField`.

Remarques :

— Dans un premier temps, on admettra que l'utilisateur a rentré un numéro de courbe qui existe.
— N'oubliez pas que chaque courbe possède sa propre méthode `toString()`.
— Pour rappel, pour convertir une chaîne de caractère appelée `maChaineDeCaractere` en entier, vous devez utiliser l'instruction `Integer.parseInt(maChaineDeCaractere)`.
— Pour définir le texte (et réciproquement le récupérer) d'un `JTextField` appelé `monTextField`, on utilise la méthode `monTextField.setText("Mon texte")` (et réciproquement `monTextField.getText()`).
— Dans un tableau, le premier élément est indexé à 0.

3.3 Bonus : ajout d'un logo

Dans la fenêtre de sélection des courbes, nous pouvons améliorer l'affichage en rajoutant le logo de l'INSA.

1. Modifiez la classe `FenetreSelectionCourbe` afin d'avoir une fenêtre équivalente à celle présentée sur la figure 4.
2. Pour afficher le logo, on utilise ici un `JLabel` qui prend en paramètre une icône (appelé `Icon` en java). Vous trouverez plus d'informations dans la javadoc du `JLabel`. On utilisera en particulier la classe `ImageIcon` (javadoc du `ImageIcon`) pour charger un fichier image (ici le logo de l'INSA).

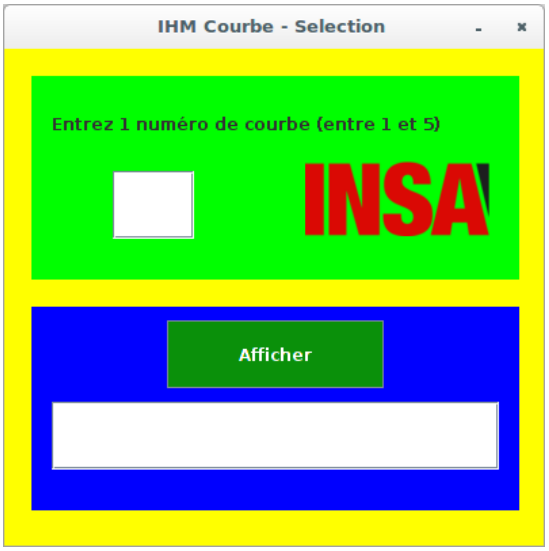

FIGURE 4 – Exemple de l'apparence visuelle de la nouvelle fenêtre

4 Objectif 2 : Allons plus loin

4.1 Ajout d'un second bouton

On souhaite maintenant laisser la possibilité à l'utilisateur d'effacer le contenu du `JTextField` qui affiche le résultat. Pour cela, on ajoute un bouton "Effacer" à côté du bouton nommé "Afficher". La figure 5 présente un exemple d'une fenêtre avec 2 boutons.

FIGURE 5 – Exemple de l'apparence visuelle de la nouvelle fenêtre avec 2 boutons

1. Le click sur un des deux boutons appelle la méthode `actionPerformed`. D'après votre cours, comment distingue t'on les 2 boutons ?
2. Dans un premier temps, affichez dans un terminal un simple texte du type `"Click sur Afficher"` ou `"Click sur Effacer"` selon le bouton sur lequel l'utilisateur a appuyé.
3. Une fois la distinction faite, effacez le contenu du `JTextField` suite à l'appui sur le second bouton.

Remarque. Pour effacer du texte, il suffit de faire appel à `setText("")` (ou `setText(null)` qui est équivalent). Plus de détails sont disponibles sur la javadoc de `JTextField`.

4.2 Gestion des erreurs de saisie

Pour l'instant si l'utilisateur donne un nombre supérieur ou inférieur au nombre de figures disponibles, votre code va générer une erreur de type : `java.lang.IndexOutOfBoundsException` (entre autres).

Pour prévenir cette erreur, il est possible d'afficher une nouvelle fenêtre particulière, appelée `JOptionPane` pour prévenir l'utilisateur. Cette fenêtre propose de nombreuses options qui sont détaillées dans la javadoc correspondante (choix des boutons disponibles, choix de l'icône, ...). Dans un premier temps, nous prendrons la version la plus simple (figure 6). Celle-ci permet d'affecter le texte de notre choix à la fenêtre. Nous utiliserons donc l'instruction `JOptionPane.showMessageDialog(this,"Mon message")` qui permet d'appliquer à votre fenêtre en cours, une boîte de dialogue affichant le texte `Mon message`. Cette boîte de dialogue est de type modale ce qui signifie que, tant qu'elle est ouverte, on ne peut pas accéder aux autres fenêtres du programme.

1. Quelle(s) méthode(s) faut il modifier dans votre code ?
2. Faites les modifications nécessaires afin de prévenir les erreurs de saisies de l'utilisateur.
3. Testez votre programme avec des nombres négatifs ou supérieur au nombre de courbes contenues dans votre tableau.

Remarques :
— Pensez à paramétrer le message en fonction de la taille de votre tableau de courbes.
— Pensez à tester la valeur rentrée par l'utilisateur **avant** de récupérer la courbe concernée.

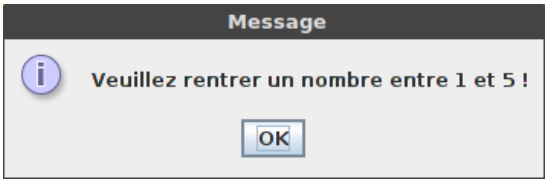

FIGURE 6 – Exemple de message d'erreur qui peut apparaître

4.3 Bonus : affichage d'un texte sur plusieurs lignes

Nous souhaitons maintenant que l'utilisateur puisse voir les données de plusieurs courbes. En effet, actuellement l'affichage dans un `JTextField` ne permet d'afficher les informations que de la dernière courbe choisie. Dans notre cas, il est conseillé d'utiliser un `JTextArea` (lien vers la javadoc) pour afficher du texte sur plusieurs lignes. La figure 7 présente un exemple de ce que vous pouvez obtenir avec un `JTextArea`.

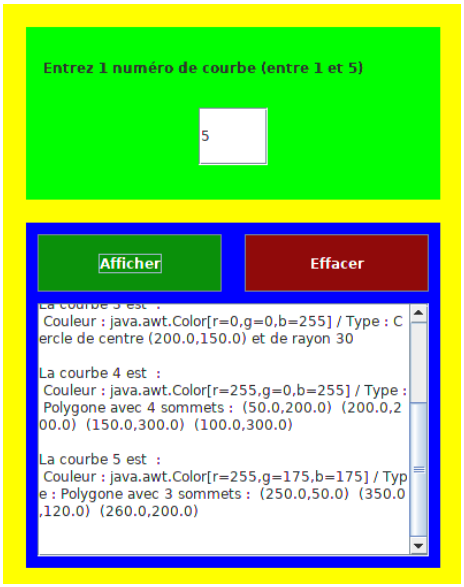

FIGURE 7 – Exemple de l'apparence visuelle de la nouvelle fenêtre avec un `JTextArea`

1. Remplacez le `JTextField` par un `JTextArea` dans lequel s'affichera successivement les informations des courbes sélectionnées.
2. Continuez d'afficher dans le terminal les informations liées aux courbes et au click sur le bouton pour vérifier que tout fonctionne.
3. Modifiez éventuellement la taille de votre fenêtre et de votre `JPanel`.
4. Comme pour le `JTextField`, vous pouvez faire appel aux méthodes `toString()` des courbes contenues dans la liste.
5. N'oubliez pas d'adapter les actions de vos boutons à ce nouveau composant.

Pour information, les méthodes de l'objet `JTextArea` qui peuvent vous être utiles sont :
— la méthode `append` avec comme paramètre la chaîne de caractère à afficher permet de rajouter du texte dans la zone de texte.
— la méthode `setLineWrap` avec comme paramètre le booléen `true` permet de faire un retour à la ligne automatique pour améliorer l'affichage.
— pour un retour à la ligne manuel, il suffit de rajouter `\n` dans la chaîne de caractère concernée.
— pour plus de détails ou d'informations, voici le lien de la javadoc du `JTextArea`

On remarque que si le nombre de lignes à afficher est trop élevé, le `JTextArea` ne permet pas de tout afficher. Il peut être intéressant pour cela d'ajouter un ascenseur qui permet de naviguer verticalement dans le `JTextArea`. Pour cela on peut utiliser un conteneur particulier : le `JScrollPane` (lien vers la javadoc). Il suffit de mettre en paramètre le `JTextArea` concerné lors de l'appel au constructeur du `JScrollPane`. L'ascenseur apparaîtra seulement s'il est nécessaire. On ajoute ensuite ce `JScrollPane` à notre conteneur principal.

Remarque. On n'a plus besoin de définir la taille du `JTextArea` puisqu'il sera intégré au `JScrollPane`. De même, maintenant c'est le `JScrollPane` qui sera ajouté au `JPanel` concerné et non plus le `JTextArea`.

5 Objectif 3 : Multi fenêtres et graphisme de base

5.1 Création d'une seconde fenêtre

Afin d'éviter de surcharger la fenêtre courante, il est possible de créer une seconde fenêtre. Cette dernière sera dédiée à l'aspect graphique de votre programme. Ainsi les courbes contenues dans le tableau de courbes seront dessinées dans cette nouvelle fenêtre.

1. Créez une nouvelle classe appelée `FenetrePlotCourbe` qui permet de créer une fenêtre vide de taille 600*600. Cette fenêtre ne sera pas visible à sa construction.
2. Elle ne sera rendu visible qu'au moment où l'utilisateur appuie sur le bouton `Afficher`. Que faut il modifier pour réaliser cette opération ? En sachant qu'il n'est pas nécessaire de modifier la classe principale `GestionCourbe`, où faut il faire les modifications ?

5.2 Dessinons dans notre nouvelle fenêtre

Il a été vu pendant le cours qu'il est possible de redéfinir la méthode `paint(Graphics g)` qui est appelé à chaque fois que la fenêtre a besoin d'être affichée ou rafraîchie. L'objet graphique `g` possède des méthodes permettant de choisir une couleur, de dessiner une droite, une ellipse, un rectangle, *etc.*

La méthode `setColor(Color c)` de l'objet `Graphics` permet de définir la couleur `c` courante.

La méthode `fillOval(int x,int y,int l,int h)` (ou `drawOval(int x,int y,int l,int h)`) de l'objet `Graphics` permet de tracer une ellipse pleine (ou le périmètre seulement) en spécifiant un rectangle dans laquelle elle s'inscrit (le sommet en haut à gauche est de coordonnées (`x`,`y`), la largeur `l` et la hauteur `h`). Vous pouvez également utiliser :

- `fillRect(int x,int y,int largeur,int hauteur);`
- `drawRect(int x,int y,int largeur,int hauteur);`
- `drawPolygon(int[] tabX, int[] tabY, int nbPoints);`
- `fillPolygon(int[] tabX, int[] tabY, int nbPoints);`
- `drawLine(int x1,int y1,int x2,int y2);`
- `drawString(String texte,int x,int y );`.

Vous trouverez facilement plus d'informations sur ces différentes méthodes dans la javadoc associée.

1. Redéfinissez la méthode `paint(Graphics g)`. Quand et comment cette méthode est elle appelée ?
2. Tracez un rectangle plein de la couleur de votre choix qui prend toute la taille de la fenêtre.
3. Tracez des cercles (ou ellipses) de différents rayons et de différentes couleurs dans différentes positions. A votre avis, que se passe t'il si on trace de nouveau le rectangle précédent ?
4. Dessinez maintenant des polygones quelconques.

La figure 8 montre un exemple de ce que vous pouvez obtenir.

5.3 Dessinons maintenant la courbe choisie par l'utilisateur

5.3.1 Récupérons la courbe choisie par l'utilisateur

Pour rappel, le programme principal permet de créer 5 courbes (3 cercles et 2 polygones). On souhaite maintenant dessiner la courbe choisie par l'utilisateur. Dans un premier temps, il faut donc d'abord récupérer cette courbe dans la classe `FenetrePlotCourbe`. Pour cela, il faut suivre les étapes suivantes :

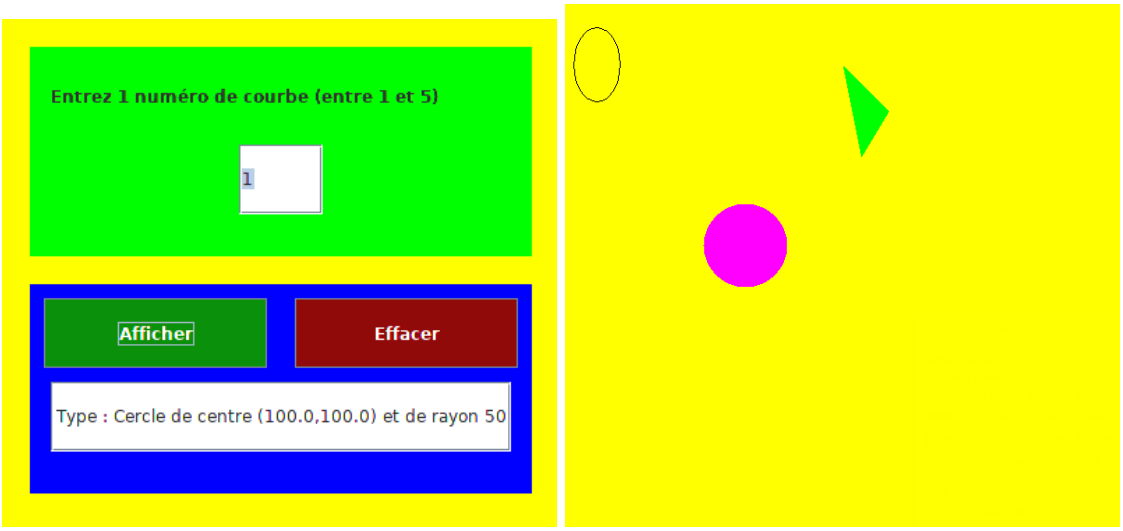

FIGURE 8 – Exemple d'un programme gérant 2 fenêtres. La fenêtre de droite est apparue suite à l'appui sur le bouton `Afficher`.

1. Ajoutez un attribut privé de type `Courbe` dans la classe `FenetrePlotCourbe` ;
2. Dans la classe `FenetrePlotCourbe`, ajoutez la méthode `choixCourbe(Courbe uneCourbe)` qui affecte à cet attribut la courbe passée en paramètre ;
3. Faites appel à cette méthode à partir de la classe `FenetreSelectionCourbe`. Où faut-il faire cet appel ?

5.3.2 Dessinons cette courbe

Votre classe `FenetrePlotCourbe` a désormais accès à la courbe choisie par l'utilisateur. Pour la dessiner, il faut modifier votre méthode `paint(Graphics g)` de votre classe `FenetrePlotCourbe`. Celle-ci s'écrit simplement ainsi :

```
/**
 * Pour dessiner la courbe choisie par l'utilisateur
 * @param l'objet graphique
 */
public void paint(Graphics g){
     g.setColor(Color.orange);
     g.fillRect(0,0,this.getWidth(),this.getHeight());
     if (maCourbe!=null)
         maCourbe.dessine(g);
}
```

Cette méthode permet de dessiner sur votre fenêtre un rectangle orange de la taille de votre fenêtre. Si l'attribut `maCourbe` de la classe n'est pas `null`, elle fait appel à la méthode `dessine(Graphics g)` qui doit être définie dans vos classes `Cercle`, `Polygone` et `Courbe`. Modifiez votre code en tenant compte de ces remarques :

1. N'oubliez pas qu'un cercle et un polygone ne se dessinent pas de la manière. Il est donc nécessaire que votre code s'adapte en fonction de l'objet à dessiner.
2. Attention dans la classe `APoint`, les attributs `x` et `y` sont des `double` et les arguments des méthodes `fillOval` et `fillPolygon` sont des `int`. Il faut donc convertir les valeurs de `x` et `y`. Pour rappel, l'instruction suivante `int b = (int)a` permet d'affecter à l'entier `b` la partie entière du `double a`.
3. Pensez à rajouter `import java.awt.Graphics;` en préambule des classes si nécessaire.

La figure 9 représente toutes les courbes contenues dans le tableau de courbes. Vous ne dessinerez **qu'une** de ces courbes.

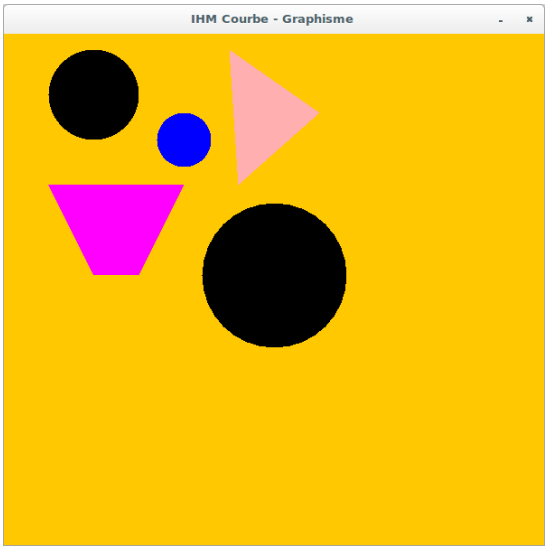

FIGURE 9 – Représentation graphique des courbes contenues dans le tableau de courbe de votre TP. *Pour rappel, vous ne devez en dessiner qu'une à la fois.*

5.3.3 Et pour dessiner une autre courbe ?

Dans l'état actuel des choses, si l'utilisateur choisit une autre courbe et clique de nouveau sur le bouton `Afficher`, rien ne se passe. En effet, la fenêtre d'affichage n'a pas été mise à jour. Comme cela a été vu en cours, il est possible d'utiliser la méthode `repaint()` pour forcer une mise à jour de la fenêtre.

Dans quelle méthode faut il insérer l'appel à la méthode `repaint()` pour obtenir l'affichage de la nouvelle courbe sélectionnée ?

5.4 Effaçons la zone de dessin

Pour l'instant le bouton `Effacer` permet uniquement d'effacer les zones de texte. On souhaiterait maintenant que la zone de dessin soit également effacée.

Pour effacer la zone de dessin, il suffit de dessiner par dessus un rectangle qui recouvre toute la zone. En vous inspirant de ce qui a été fait précédemment, faites les modifications nécessaires.

6 Objectif 4 : Gestion du temps et animation

Cet objectif consiste à réaliser une animation de base. Les figures que vous avez dessinées vont "tomber" petit à petit comme si elle était soumise à la gravité.

6.1 Gérer le temps

Avant de réaliser une animation, il est nécessaire de mettre en place un `Timer`. Pour rappel, celui ci, lors de son réveil, lance la méthode `actionPerformed(ActionEvent e)` à une certaine fréquence.

1. A votre avis, quelle fenêtre (et donc quelle classe) a besoin d'être mise à jour régulièrement ?
2. Créez un `Timer` dans cette classe et affichez le message "Coucou" dans un terminal toutes les secondes. Pour l'instant, votre `Timer` débutera dès que la classe sera créée.
3. Tant que l'utilisateur n'a pas appuyé sur le bouton "Afficher" il ne sert à rien de lancer votre chronomètre. Quelle(s) classe(s) doivent être modifiée(s) ? Faites les modifications nécessaires.
4. A titre informatif, affichez le temps qui s'écoule dans la barre du titre de la fenêtre courante.

6.2 Animation basique

Maintenant que le Timer a été mis en place, il est possible de réaliser une petite animation.

1. Dans un premier temps, on souhaite faire "tomber" la courbe comme si elle était soumise à la gravité. On souhaite donc mettre en place une méthode `deplaceY(int deltaY)` qui permet de modifier les coordonnées de la courbe présente sur votre fenêtre.

   En vous inspirant de la méthode `dessine(Graphics g)` (section 5.3), modifiez votre code en tenant compte de ces remarques :

   (a) À votre avis, dans quelle(s) classe(s) devrai(en)t se trouver la méthode `deplaceY(int deltaY)` ?

   (b) Quelle(s) méthode(s) appelle(nt) cette méthode ?

   (c) Dans un premier temps, on considéra un déplacement de 100 pixels par seconde. Sur quel(s) paramètre(s) peut on jouer pour avoir un déplacement moins saccadé ?
   Remarque. Pour rappel, l'axe Y est vertical descendant.

2. Au bout d'un certain temps, la courbe "sort" de la fenêtre car ses coordonnées sont supérieures à la taille de la fenêtre. Faites en sorte que lorsque la courbe atteint les limites supérieures et inférieures de la fenêtre, elle reparte dans l'autre sens.

   Remarques :
   — Attention l'axe Y est dirigé vers le bas.
   — Pour l'instant nous ne tiendrons pas compte de l'épaisseur des bordures de la fenêtre. Pour ce faire, vous devez chercher des informations sur la méthode `getInsets()`.

6.3 Bonus : animation déclenchée à la souris

Il serait intéressant de déclencher autrement le début du chronomètre pour pouvoir dessiner plusieurs courbes avant qu'elles ne commencent à se déplacer. Il est possible par exemple de déclencher le début du Timer lorsque l'utilisateur clique avec sa souris sur la fenêtre où sont présentes les courbes.

Si l'utilisateur clique à nouveau, le chronomètre (et donc l'animation) se met en pause jusqu'au prochain click et ainsi de suite. Faites les modifications nécessaires.

Remarque. Rappel : la méthode `monTimer.isRunning()` renvoie `true` si un Timer est en cours d'exécution et `false` sinon.

7 Objectif 5 - Personnalisez votre programme

Il existe de nombreuses possibilités pour gérer une interface graphique. Nous ne pouvons pas tout couvrir ici mais vous trouverez ici quelques suggestions afin de tester différentes possibilités. Vous trouverez des exemples minimalistes sous Moodle (PolyJavaExemple.pdf)

A partir de ces exemples, voici quelques propositions pour compléter votre programme actuelle. Libre à vous de les faire dans l'ordre de votre choix ou d'en ajouter d'autres.

7.1 Interaction avec la souris

1. Affichez les coordonnées dans un terminal de la souris lors du click.
2. Affichez les coordonnées dans la barre de titre de la souris lors du click.
3. Cliquez sur une figure et celle-ci doit changer de couleur.
4. Modifiez la position d'une courbe lorsqu'on clique dessus.

7.2 Interaction avec le clavier

1. Faites apparaître ou disparaître une courbe en appuyant sur le numéro de la courbe.
2. Toutes les courbes deviennent bleues (respectivement rouge) si l'utilisateur appuie sur la touche "b" (respectivement "r").

7.3 Utilisation d'un `JComboBox`

Pour faciliter la sélection de la courbe à étudier, il est possible d'utiliser un menu déroulant. Le `JTextField` précédemment utilisé sera remplacé par un `JComboBox` (lien vers la javadoc). Dans ce widget, on affichera la liste des courbes créées afin de permettre à l'utilisateur de choisir celle qu'il souhaite afficher (figure 10). Pour créer un `JComboBox`, il faut simplement envoyer en paramètre de son constructeur un tableau de `String` contenant la liste des items à afficher.

Ainsi il suffira d'écrire `JComboBox courbeListe = new JComboBox(liste)` où `liste` est le tableau de `String` contenant le texte adéquat. Afin de rendre votre code flexible, pensez à paramétrer la création de `liste` en fonction du tableau de courbes passée en paramètre du constructeur de la fenêtre.

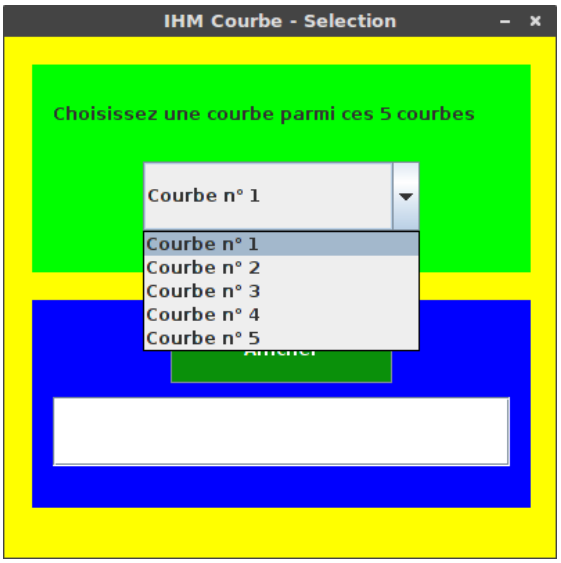

FIGURE 10 – Détails de la liste contenue dans le `JComboBox`

Remarque. Pour récupérer l'information de la ligne choisie dans un `JComboBox`, il faut faire appel à la méthode `getSelectedIndex()` (*cf.* javadoc sur le `JComboBox`).

7.4 Mélangeons les couleurs

1. Si deux courbes se superposent, leur couleur change et devienne un mélange des 2 couleurs initiales
2. Rajoutez des `JSlider` pour définir la proportion de rouge, vert et bleu des courbes.

7.5 Récupérons les courbes à partir d'un fichier texte

1. Créez un fichier texte qui contient les données nécessaires à la création des courbes.
2. Créez une classe qui récupère ces données et crée un tableau de courbes en conséquence.
3. Utilisez votre programme pour dessiner les courbes de ce tableau.

7.6 Dessinons dans une zone dédiée de la fenêtre

1. Créez un `JPanel` dans lesquels les courbes seront dessinées.
2. Amélioriez votre programme pour intégrer proprement ce nouveau `JPanel`.

7.7 Plaçons de manière automatique les différents éléments de votre fenêtre

Le fait de ne pas utiliser de `LayoutManager` ne vous garantit pas une bonne visualisation de votre IHM. Il suffit de redimensionner la fenêtre pour que le résultat affiché ne soit pas celui attendu.

Pour cela, il faut penser à utiliser des `JPanel` et un `LayoutManager`. Les `LayoutManager` les plus communs sont `FlowLayout` et `BorderLayout`. Celui qui permet le plus de liberté est le `GridBagLayout` (cf. ce site par exemple).

Vous trouverez de nombreux exemples sur la JavaDoc ou sur Internet pour vous guider.