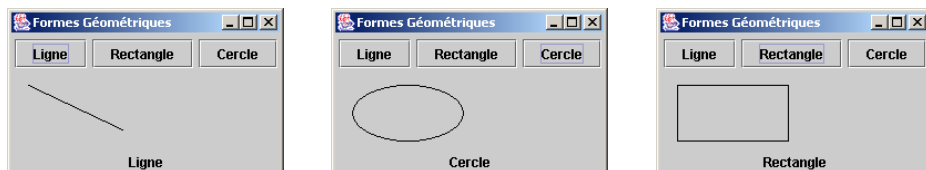


TD JAVA n°8: Interface Graphique - Dessiner en Java

Exercice 1 : Dessiner en Java

A l'aide de composants **Swing**, écrire le code qui permet de visualiser l'application graphique suivante (taille conseillée pour la fenêtre largeur: 500 pixels et hauteur: 300 pixels)



→ Lorsqu'on clique sur le bouton « **Ligne** », le mot « **Ligne** » est affiché dans la zone d'affichage (en bas) et un dessin représentant une ligne est dessiné dans le panneau de dessin.

→ Lorsqu'on clique sur le bouton « **Rectangle** », le mot « **Rectangle** » est affiché dans la zone d'affichage et un dessin représentant un rectangle est dessiné dans le panneau de dessin.

→ Lorsqu'on clique sur le bouton « **Cercle** », le mot « **Cercle** » est affiché dans la zone d'affichage et un dessin représentant un cercle est dessiné dans le panneau de dessin.

Remarque : lors du lancement de l'application, on considère que l'on est dans le premier cas, c'est à dire qu'une ligne est affichée.

| | | | |
|--------------------------|---------------------------|---|---|
| java.awt | Class Graphics | abstract void drawLine(int x1, int y1, int x2, int y2) | Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system. |
| | | abstract void drawOval(int x, int y, int width, int height) | Draws the outline of an oval. |
| java.lang.Object | +--java.awt.Graphics | abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints) | Draws a closed polygon defined by arrays of x and y coordinates. |
| | | void drawPolygon(Polygon p) | Draws the outline of a polygon defined by the specified Polygon object. |
| Direct Known Subclasses: | DebugGraphics, Graphics2D | abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints) | Draws a sequence of connected lines defined by arrays of x and y coordinates. |
| | | void drawRect(int x, int y, int width, int height) | Draws the outline of the specified rectangle. |

Exercice 2 : Utilisation d'une énumération...

Quelques mots à propos des types énumérés : mot clé **enum**

Dans les applications, vous avez parfois besoin de créer une ensemble de constantes pour représenter les seules valeurs valides d'une variable. Cet ensemble de valeurs valides est appelée **énumérations**.

↳ L'ancienne façon de simuler une énumération était de créer une classe composée de constantes (déclarées en `static final`, comme pour les constantes de l'API du genre `EXIT_ON_CLOSE` de la classe `JFrame`.)

```
class Titre {
    public static final int MONSIEUR = 0;
    public static final int MADAME = 1;
    public static final int MADEMOISELLE = 2;
};

public class TestTitreConstante {

    public static void main(String[] args) {
        int titrePatient = Titre.MONSIEUR;

        if (titrePatient == Titre.MONSIEUR)
            System.out.println("Titre " + titrePatient + " = Mr");
        if (titrePatient == Titre.MADAME)
            System.out.println("Titre " + titrePatient + " = Mme");
        if (titrePatient == Titre.MADEMOISELLE)
            System.out.println("Titre " + titrePatient + " = Melle");
    }
}
```

Le principal inconvénient de cette technique est qu'il n'y a pas de contrôle sur la valeur affectée à une donnée surtout si les constantes ne sont pas utilisées c-a-d qu'on peut affecter une valeur hors valeurs prédéfinies, par exemple 6 ou -123

↳ C'est pourquoi le type énuméré (**mot-clé enum**) a été introduit dans Java 5 pour créer "proprement" des énumérations. En effet, grâce au mot clé **enum**, le compilateur peut vérifier la validité des valeurs à la compilation, apportant ainsi une plus grande sécurité dans vos applications

```
public class TestTitreEnumeration {
    enum Titre {MONSIEUR, MADAME, MADEMOISELLE};

    public static void main(String[] args) {
        Titre titrePatient = Titre.MONSIEUR;

        if (titrePatient == Titre.MONSIEUR)
            System.out.println("Titre " + titrePatient + " = Mr");
        if (titrePatient == Titre.MADAME)
            System.out.println("Titre " + titrePatient + " = Mme");
        if (titrePatient == Titre.MADEMOISELLE)
            System.out.println("Titre " + titrePatient + " = Melle");
    }
}
```

Remarques :

→ Les énumérations sont alors des classes spéciales. Notez bien les accolades utilisées pour la déclaration de l'énumération : **enum { ... }**

Quand vous créez un type énuméré, vous créez une nouvelle classe et vous étendez implicitement `java.lang.Enum`. Vous pouvez la déclarer comme isolée dans son propre fichier source, ou comme dans l'exemple précédent comme un membre d'une autre classe.

→ Les différentes valeurs de l'**enum** correspondent à des constantes statiques et publiques et peuvent donc être accédées directement comme les champs **public static** (par exemple avec `Titre.MONSIEUR`). Notez également l'affichage : la méthode `toString()` de chaque type énuméré *affichera par défaut son nom* tel qu'il est inscrit dans le fichier source Java (ici `MONSIEUR`).

Autres exemples de types énumérés:

```
public enum Couleur {COEUR, CARREAU, PIQUE, TREFLE};
public enum Jour {Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi,Dimanche};
public enum Season {spring, summer, autumn, winter};
```

☞ Dans sa forme la plus basique, un **enum** contient simplement la liste des valeurs possibles qu'elle peut prendre. Toutefois, un **type énuméré reste une classe Java**, qui peut accepter des champs, des méthodes et des constructeurs. Par exemple, il est possible de compléter la dernière énumération `Season` avec le nom français de la saison :

```
public enum Season {
    spring("printemps"),summer("été"), autumn("automne"), winter("hiver");

    private String label;

    // Constructeur
    Season(String pLabel) {
        this.label = pLabel;
    }

    public String getLabel() {
        return this.label;
    }
}
```

Remarques :

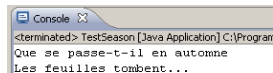
→ Les constructeurs des **enum** sont implicitement **private**, ils n'acceptent pas d'autres modificateurs d'accessibilité (pas de **public**, ni de **protected**) car ils ne sont utilisés que pour initialiser les différentes valeurs de l'énumération.

Le code suivant est ainsi incorrect : ~~`Season s = new Season ("Hiver");`~~

→ Les types **enum** de **Java 5** sont **type-safe**, c'est à dire qu'une **enum** ne peut en aucun cas prendre une autre valeur que celles définies dans sa déclaration, c'est pourquoi on ne peut pas construire de nouvelle instance. On ne permet pas non plus l'héritage d'une **enum** qui serait susceptible de définir un ensemble infini de valeurs et "casserait" donc la logique des enum (à savoir liste finie d'éléments).

☞ Enfin, les types énumérés peuvent directement être utilisés dans un **switch** :

```
public class TestSeason {
    public static void main(String[] args) {
        Season saisonEnCours = Season.autumn;
        System.out.println("Que se passe-t-il en " + saisonEnCours.getLabel());
        switch (saisonEnCours) {
            case spring: System.out.println("Les arbres sont en fleurs !!! ");
            break;
            case summer: System.out.println("Il fait chaud !!! "); break;
            case autumn: System.out.println("Les feuilles tombent... ");break;
            case winter: System.out.println("Il neige !!! "); break;
        }
    }
}
```



Pour en savoir plus sur enum : <http://download.oracle.com/javase/1.5.0/docs/guide/language/enums.html>

☞ Travail à faire ...

L'exercice précédent se prête bien à l'utilisation d'une énumération.

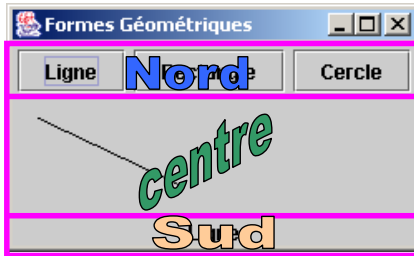
1. Dans un premier temps, écrire une **énumération Forme** qui contient les valeurs suivantes : **ligne**, **rectangle**, **cercle**
2. Modifier le code du panneau de dessin de l'exercice 1 de manière à utiliser cette énumération dans la mise en place de l'application demandée.

Correction TD JAVA n°8: Interface graphique - Dessiner en Java -

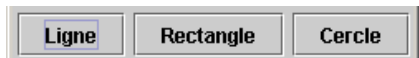
Exercice 1 :

3 fichiers :

- **TestDessinGeometrie** : fichier test qui lance l'application (pas besoin de l'écrire).
- **PanelPrincipal** : mise en place des composants et gestion des événements (clic sur les boutons) (...`extends JPanel implements ActionListener`)
- **PanelDessin** : pour dessiner en Java : ligne, rectangle, cercle ⇒ redéfinition de `paintComponent`



mainPanel de type PanelPrincipal



panelBoutons de type JPanel



MonDessin de type PanelDessin
(qui hérite de JPanel et
redéfinie `paintComponent`)

🔗 **TestDessinGeometrie.java** : fichier test qui lance l'application.

```
import javax.swing.*;

public class TestDessinGeometrie
{
    public static void main(String args[])
    {
        new TestDessinGeometrie();
    }

    public TestDessinGeometrie()
    {
        // Création d'une fenêtre graphique
        JFrame mainFrame = new JFrame("Formes Géométriques");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Ajout d'un panel au contenu de la fenêtre
        JPanel mainPanel = new PanelPrincipal();
        mainFrame.getContentPane().add(mainPanel);

        // Taille et Visibilité de la fenêtre
        mainFrame.setSize(500,300); // hauteur: 300 pixels - largeur: 500 pixels
        mainFrame.setVisible(true);
    }
}
```

🔗 **PanelPrincipal.java** : mise en place des composants et gestion des événements (clic sur les boutons)

```
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.event.*; //événement

public class PanelPrincipal extends JPanel implements ActionListener
{
    JButton bLigne;
    JButton bRectangle;
    JButton bCercle;

    JLabel infoAffichage;
    PanelDessin monDessin; // il faut bien déclarer un PanelDessin et non JPanel
    // pour que les méthodes dessineLigne, dessineRectangle, dessineCerclesoient connues

    PanelPrincipal()
    {
        setLayout (new BorderLayout());

        // Création et mise en place des boutons dans le pannel
        JPanel panelBoutons = new JPanel(); // panel en FlowLayout par défaut
        bLigne = new JButton("Ligne");
        panelBoutons.add(bLigne);
        bRectangle= new JButton("Rectangle");
        panelBoutons.add(bRectangle);
        bCercle= new JButton("Cercle");
        panelBoutons.add(bCercle);
        add(panelBoutons,BorderLayout.NORTH);

        // Création et mise en place de la zone texte
        infoAffichage=new JLabel("Ligne"); // par défaut, on dessine une ligne
        infoAffichage.setHorizontalAlignment(JLabel.CENTER); // pour centrer le texte
        add(infoAffichage,BorderLayout.SOUTH);

        // Création et mise en place du panel de dessin
        monDessin=new PanelDessin();
        add(monDessin,BorderLayout.CENTER);

        // enregistrement des écouteurs
        bLigne.addActionListener(this);
        bRectangle.addActionListener(this);
        bCercle.addActionListener(this);
    }

    // gestionnaire d'événements directement dans la classe (comme ici)
    // ou en classe interne membre
    // mais pas en classe anonyme car les 3 boutons
    // sont branchés sur le même écouteur et on utilise getSource()
    // pour identifier le composant qui a lancé l'événement ActionEvent (clic...)

    public void actionPerformed(ActionEvent ev)
    {
        if (ev.getSource() == bLigne)
        {
            infoAffichage.setText("Ligne");
            monDessin.dessineLigne();
        }
        if (ev.getSource() == bRectangle)
        {
            infoAffichage.setText("Rectangle");
            monDessin.dessineRectangle();
        }
        if (ev.getSource() == bCercle)
        {
            infoAffichage.setText("Cercle");
            monDessin.dessineCercle();
        }
    }
}
```

```

        {
            infoAffichage.setText("Cercle");
            monDessin.dessineCercle();
        }

// Remarque : on ne peut pas écrire un switch(ev.getSource())
// car public Object getSource() et switch ne sait pas traiter les Object ...
    }
}

```

🔗 **PanelDessin.java**: pour dessiner en Java : ligne, rectangle, cercle ⇒ redéfinition de **paintComponent**

```

import java.awt.Graphics;
import javax.swing.JPanel;

public class PanelDessin extends JPanel
{
    private String formeChoisie = "LIGNE";

    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g); // Ne pas oublier !!!

        // l'origine point (0,0) est bien le coin haut gauche du panel de dessin
        if (formeChoisie == "LIGNE") g.drawLine(15,10,100,50);
        if (formeChoisie == "RECTANGLE") g.drawRect(15,10,100,50);
        if (formeChoisie == "CERCLE") g.drawOval(15,10,100,50);

        repaint(); // si on ne met pas le repaint, il faut attendre
                   // que la fenêtre soit redimensionnée pour que la forme change
    }

    public void dessineLigne(){
        formeChoisie = "LIGNE";
    }

    public void dessineRectangle(){
        formeChoisie = "RECTANGLE";
    }

    public void dessineCercle(){
        formeChoisie = "CERCLE";
    }
}

```

Remarque : au lieu de faire un réaffichage continu (comme précédemment et comme dans le cours), on peut appeler **repaint()** uniquement lorsqu'un bouton est cliqué...

repaint() est alors appliqué sur l'objet monDessin de type JPanelPrincipal ...

On ne bloque plus le CPU, puisque l'affichage se fait « ponctuellement »...

Le code précédent devient :

```

public void actionPerformed(ActionEvent ev)
{
    if (ev.getSource() == bLigne)
    {
        infoAffichage.setText("Ligne");
        monDessin.dessineLigne();
        monDessin.repaint();
    }

    if (ev.getSource() == bRectangle)
    {
        infoAffichage.setText("Rectangle");
        monDessin.dessineRectangle();
        monDessin.repaint();
    }

    if (ev.getSource() == bCercle)
    {
        infoAffichage.setText("Cercle");
        monDessin.dessineCercle();
        monDessin.repaint();
    }
}

```

🔗 **PanelDessin.java**: pour dessiner en Java : ligne, rectangle, cercle ⇒ redéfinition de **paintComponent**

```

import javax.swing.*;
import java.awt.*;

public class PanelDessin extends JPanel
{
    private String formeChoisie = "LIGNE";

    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        if (formeChoisie == "LIGNE") g.drawLine(15,10,100,50);
        // l'origine point (0,0) est bien le coin haut gauche du panel de dessin
        if (formeChoisie == "RECTANGLE") g.drawRect(15,10,100,50);
        if (formeChoisie == "CERCLE") g.drawOval(15,10,100,50);
        // pas de repaint, puisqu'il est appelé lors du clic ...
    }

    public void dessineLigne(){
        formeChoisie = "LIGNE";
    }

    public void dessineRectangle(){
        formeChoisie = "RECTANGLE";
    }

    public void dessineCercle(){
        formeChoisie = "CERCLE";
    }
}

```

Exercice 2 : Utilisation d'une énumération dans le panneau de Dessin ...

On peut utiliser l'énumération dans la classe PanelDessin qui devient alors (modification enrouge) ...

```
import java.awt.Graphics;

import javax.swing.JPanel;

public class PanelDessin extends JPanel{

    enum Forme {ligne,rectangle,cercle}; //déclaration de l'énumération...
    private Forme formeChoisie = Forme.ligne; //on déclare une Forme
                                           // et plus un String

    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        switch(formeChoisie)
            //avec une enum, on peut maintenant écrire un switch ...
            {
                case ligne :      g.drawLine(15,10,100,50);
                                break;
                case rectangle : g.drawRect(15,10,100,50);
                                break;
                case cercle :    g.drawOval(15,10,100,50);
                                break;
            }

        repaint();
        // si on ne met pas le repaint,
        // il faut attendre que la fenêtre soit redimensionnée
        // pour que la forme change
    } // fin paintComponent

    public void dessineLigne()
    { formeChoisie = Forme.ligne;}

    public void dessineRectangle()
    {formeChoisie = Forme.rectangle;}

    public void dessineCercle()
    {formeChoisie = Forme.cercle;}
}
```

... AutreExercice 2 : Compteur avec deux boutons

```
import javax.swing.*; // pour les composants
import java.awt.*;    // pour le gestionnaire

import java.awt.event.*; // pour traiter les événements
public class TestDeuxBoutons
{
    private JLabel label;
    private JButton buttonPlus;
    private JButton buttonMoins;
    private int count = 0;

    public static void main(String args[])
    {
        new TestDeuxBoutons();
    }

    public TestDeuxBoutons()
    {
        // Déclaration et Instanciation de la fenêtre
        JFrame mainFrame = new JFrame("Test avec 2 boutons");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Placement des composants dans un panel
        JPanel mainPanel = new JPanel(new BorderLayout());

        JPanel panelButtons = new JPanel(new GridLayout(0, 1));
        buttonPlus = new JButton("plus");
        buttonMoins = new JButton("moins");
        panelButtons.add(buttonPlus);
        panelButtons.add(buttonMoins);

        label = new JLabel("0", SwingConstants.CENTER);
        mainPanel.add(panelButtons, BorderLayout.WEST);
        mainPanel.add(label, BorderLayout.CENTER);

        // Ecouteurs
        buttonPlus.addActionListener(new MyActionListener());
        buttonMoins.addActionListener(new MyActionListener());

        // Ajout du panel à la fenêtre
        mainFrame.getContentPane().add(mainPanel);

        // Affichage de la fenêtre à l'écran
        mainFrame.setSize(500,150);
        mainFrame.setVisible(true);
    }

    private class MyActionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            if(event.getSource() == buttonPlus) count++;
            else count--;
            label.setText(Integer.toString(count));
        }
    }
}
```



But de l'exercice :

Utiliser getSource et choisir la bonne implémentation : pas de classe interne anonyme ... mais ... on regroupe le traitement dans la même méthode actionPerformed grâce à getSource

Remarque : une valeur 0 pour l'initialisation du GridLayout signifie "un nombre indéterminé" de composants.