LES MENUS ET BOÎTES DE DIALOGUE

Java et la programmation événementielle (Durée : 1 semaine)

1. Notre première fenêtre
2. Le fil rouge : une animation
3. Positionner des boutons
4. Interagir avec des boutons
5. TP : une calculatrice
6. Exécuter des tâches simultanément
7. Les champs de formulaire
8. **Les menus et boîtes de dialogue**
9. TP : l'ardoise magique
10. Conteneurs, sliders et barres de progression
11. Les arbres et leur structure
12. Les interfaces de tableaux
13. TP : le pendu
14. Mieux structurer son code : le pattern MVC
15. Le Drag'n Drop
16. Mieux gérer les interactions avec les composants

Quiz

Accéder au forum

Voici deux éléments très utiles à l'élaboration de programmes offrant plusieurs fonctio ns les différentes applications que vous pourrez trouver sur le Net. Vous verrez que la manière d'utiliser familiariser avec l' pour pouvoir faire des choses sympa. Quant à l'utilisation de boîtes de dialogue, c'est

Les boîtes de dialogue

Les boîtes de dialogue, c'est certain, vous connaissez ! Cependant, afin de nous assure e boîte de dialogue s'agit d'une petite fenêtre pouvant servir à plusieurs choses :

- afficher une information (message d'erreur, d'avertissement…) ;
- demander une validation, une réfutation ou une annulation ;
- demander à l'utilisateur de saisir une information dont le système a besoin ;
- etc.

Vous pouvez voir qu'elles peuvent servir à beaucoup de choses. Il faut toutefois les util rre une boîte de dia à chaque notification, car toute boîte ouverte doit être fermée ! Pour ce point je vous la

Les boîtes d'information

L'objet que nous allons utiliser tout au long de ce chapitre est le `JOptionPane`, un o

La figure suivante vous montre à quoi ressemblent des boîtes de dialogues « informati

Exer

Ces boîtes ne sont pas destinées à participer à de quelconques opérations : elles affich r ces boîtes :

```
1 JOptionPane jop1, jop2, jop3;
2
3 //Boîte du message d'information
4 jop1 = new JOptionPane();
5 jop1.showMessageDialog(null, "Message informatif", "Information", JOptionPane.INFORMATION_MESSAGE);
6
7 //Boîte du message préventif
8 jop2 = new JOptionPane();
9 jop2.showMessageDialog(null, "Message préventif", "Attention", JOptionPane.WARNING_MESSAGE);
10
11 //Boîte du message d'erreur
12 jop3 = new JOptionPane();
13 jop3.showMessageDialog(null, "Message d'erreur", "Erreur", JOptionPane.ERROR_MESSAGE);
```

Ces trois boîtes ne s'affichent pas en même temps, tout simplement parce qu'en Java (mais aussi dans les autres langages), les boîtes de dialogue sont dites *modales*. Cela signifie que lorsqu'une boîte fait son apparition, celle-ci bloque toute interaction avec un autre composant, et ceci tant que l'utilisateur n'a pas mis fin au dialogue !

Maintenant, voyons de plus près comment construire un tel objet. Ici, nous avons utilisé la méthode `showMessageDialog(Component parentComponent, String messa String title, int messageType);` où :

- `String title` : permet de donner un titre à l'objet.
- `int messageType` : permet de savoir s'il s'agit d'un message d'information, de prévention ou d'erreur. Vous avez sans doute remarqué que, mis à part le texte et le titre, seul ce variait entre nos trois objets !

Il existe deux autres méthodes `showMessageDialog()` pour cet objet : une qui prend deux paramètres en moins (le titre et le type de message), et une qui prend un paramètre en plus (l'icône à utiliser).

Je pense qu'il est inutile de détailler la méthode avec les paramètres en moins, mais voici des exemples de boîtes avec des icônes définies par nos soins.

```
1 import javax.swing.ImageIcon;
2 import javax.swing.JOptionPane;
3
4 public class Test {
5   public static void main(String[] args) {
6     JOptionPane jop1, jop2, jop3;
7     jop1 = new JOptionPane();
8     ImageIcon img = new ImageIcon("images/information.png");
9     jop1.showMessageDialog(null, "Message informatif", "Information", JOptionPane.INFORMATION_MESSAGE, img);
10    jop2 = new JOptionPane();
11    img = new ImageIcon("images/warning.png");
12    jop2.showMessageDialog(null, "Message préventif", "Attention", JOptionPane.WARNING_MESSAGE, img);
13    jop3 = new JOptionPane();
14    img = new ImageIcon("images/erreur.png");
15    jop3.showMessageDialog(null, "Message d'erreur", "Erreur", JOptionPane.ERROR_MESSAGE, img);
16  }
17 }
```

Ces images ont été trouvées sur Google puis rangées dans un dossier « images » à la racine du projet Eclipse. Je vous invite à télécharger vos propres images et à faire vos tests. Vous rema aussi l'emploi de l'objet `ImageIcon`, qui lit le fichier image à l'emplacement spécifié dans son constructeur. La figure suivante représente le résultat obtenu.

Images personnalisées dans des boîtes de dialogue

Ce type de boîte est très utile pour signaler à l'utilisateur qu'une opération s'est terminée ou qu'une erreur est survenue. L'exemple le plus simple qui me vient en tête est le cas d'une divisi zéro : on peut utiliser une boîte de dialogue dans le bloc `catch`.

Voici les types de boîtes que vous pouvez afficher (ces types restent valables pour tout ce qui suit) :

- `JOptionPane.ERROR_MESSAGE`
- `JOptionPane.INFORMATION_MESSAGE`
- `JOptionPane.PLAIN_MESSAGE`
- `JOptionPane.QUESTION_MESSAGE`
- `JOptionPane.WARNING_MESSAGE`

Je pense que vous voyez désormais l'utilité de telles boîtes de dialogue. Nous allons donc poursuivre avec les boîtes de confirmation.

Les boîtes de confirmation

Comme leur nom l'indique, ces dernières permettent de valider, d'invalider ou d'annuler une décision. Nous utiliserons toujours l'objet `JOptionPane`, mais ce sera cette fois avec la mét `showConfirmDialog()`, une méthode qui retourne un entier correspondant à l'option que vous aurez choisie dans cette boîte :

- Yes ;
- No ;
- Cancel.

Comme exemple, nous pouvons prendre notre animation dans sa version la plus récente. Nous pourrions utiliser une boîte de confirmation lorsque nous cliquons sur l'un des boutons cont l'animation (`Go` ou `Stop`).

Voici les modifications de notre classe `Fenetre` :

```
1 //Les autres imports n'ont pas changé
2 import javax.swing.JOptionPane;
3
4 public class Fenetre extends JFrame{
5   private Panneau pan = new Panneau();
6   private JButton bouton = new JButton("Go");
7   private JButton bouton2 = new JButton("Stop");
8   private JPanel container = new JPanel();
9   private JLabel label = new JLabel("Choix de la forme");
10  private int compteur = 0;
11  private boolean animated = true;
12  private boolean backX, backY;
13  private int x,y ;
14  private Thread t;
```

```
  }

  private void go(){
    //Cette méthode n'a pas changé non plus
  }

  public class BoutonListener implements ActionListener{
    public void actionPerformed(ActionEvent arg0) {
      JOptionPane jop = new JOptionPane();
      int option = jop.showConfirmDialog(null,
        "Voulez-vous lancer l'animation ?",
        "Lancement de l'animation",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);

      if(option == JOptionPane.OK_OPTION){
        animated = true;
        t = new Thread(new PlayAnimation());
        t.start();
        bouton.setEnabled(false);
        bouton2.setEnabled(true);
      }
    }
  }

  class Bouton2Listener  implements ActionListener{
    public void actionPerformed(ActionEvent e) {
      JOptionPane jop = new JOptionPane();
      int option = jop.showConfirmDialog(null,
        "Voulez-vous arrêter l'animation ?",
        "Arrêt de l'animation",
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE);

      if(option != JOptionPane.NO_OPTION &&
      option != JOptionPane.CANCEL_OPTION &&
      option != JOptionPane.CLOSED_OPTION){
        animated = false;
        bouton.setEnabled(true);
        bouton2.setEnabled(false);
      }
    }
  }

  class PlayAnimation implements Runnable{
    public void run() {
      go();
    }
  }

  class FormeListener implements ActionListener{
    //Rien de changé
  }

  class MorphListener implements ActionListener{
    //Rien de changé
  }
}
```

Les instructions intéressantes se trouvent ici :

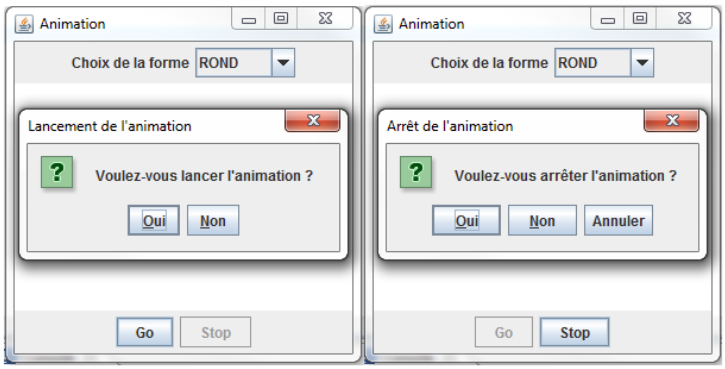
```
//…
JOptionPane jop = new JOptionPane();
int option = jop.showConfirmDialog(null, "Voulez-vous lancer l'animation ?", "Lancement de l'animation", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MES

if(option == JOptionPane.OK_OPTION){
  animated = true;
  t = new Thread(new PlayAnimation());
  t.start();
  bouton.setEnabled(false);
  bouton2.setEnabled(true);
}
//…
//…
JOptionPane jop = new JOptionPane();
int option = jop.showConfirmDialog(null, "Voulez-vous arrêter l'animation ?", "Arrêt de l'animation", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION

if(option != JOptionPane.NO_OPTION &&
   option != JOptionPane.CANCEL_OPTION &&
   option != JOptionPane.CLOSED_OPTION){
     animated = false;
     bouton.setEnabled(true);
     bouton2.setEnabled(false);
}
```

Voyons ce qu'il se passe ici :

- nous initialisons notre objet `JOptionPane` : rien d'étonnant ;
- en revanche, plutôt que d'afficher directement la boîte, nous affectons le résultat que renvoie la méthode `showConfirmDialog()` à une variable de type `int` ;
- nous nous servons de cette variable afin de savoir quel bouton a été cliqué (oui ou non).

En fait, lorsque vous cliquez sur l'un des deux boutons présents dans cette boîte, vous pouvez affecter une valeur de type `int` :

- correspondant à l'entier `JOptionPane.OK_OPTION`, qui vaut 0 (`JOptionPane.YES_OPTION` a la même valeur);
- correspondant à l'entier `JOptionPane.NO_OPTION`, qui vaut 1 ;
- correspondant à l'entier `JOptionPane.CANCEL_OPTION` pour la boîte apparaissant lors du clic sur « **Stop** », qui vaut 2 ;
- correspondant à l'entier `JOptionPane.CLOSED_OPTION` pour la même boîte que ci-dessus et qui vaut -1.



JOptionPane avec notre animation

Les boîtes de saisie

Je suis sûr que vous avez deviné à quoi peuvent servir ces boîtes. Oui, tout à fait, nous allons pouvoir y saisir du texte ! Et mieux encore : nous pourrons même obtenir une boîte de dialogue propose des choix dans une liste déroulante. Vous savez déjà que nous allons utiliser l'objet `JOptionPane`, et les plus curieux d'entre vous ont sûrement dû jeter un œil aux autres méth proposées par cet objet… Ici, nous allons utiliser la méthode `showInputDialog(Component parent, String message, String title, int messageType)`, qu retourne une chaîne de caractères.

Voici un code la mettant en œuvre et la figure suivante représentant son résultat :

```
1 import javax.swing.JOptionPane;
2
3 public class Test {
4   public static void main(String[] args) {
5     JOptionPane jop = new JOptionPane(), jop2 = new JOptionPane();
6     String nom = jop.showInputDialog(null, "Veuillez décliner votre identité !", "Gendarmerie nationale !", JOptionPane.QUESTION_MESSAGE);
7     jop2.showMessageDialog(null, "Votre nom est " + nom, "Identité", JOptionPane.INFORMATION_MESSAGE);
8   }
9 }
```

Exemples de boîtes de saisie

Rien d'extraordinaire… Maintenant, voyons comment on intègre une liste dans une boîte de ce genre. Vous allez voir, c'est simplissime !

```
1 import javax.swing.JOptionPane;
2
3 public class Test {
4   public static void main(String[] args) {
5     String[] sexe = {"masculin", "féminin", "indéterminé"};
6     JOptionPane jop = new JOptionPane(), jop2 = new JOptionPane();
7     String nom = (String)jop.showInputDialog(null,
8       "Veuillez indiquer votre sexe !",
9       "Gendarmerie nationale !",
10      JOptionPane.QUESTION_MESSAGE,
11      null,
12      sexe,
13      sexe[2]);
14    jop2.showMessageDialog(null, "Votre sexe est " + nom, "Etat civil", JOptionPane.INFORMATION_MESSAGE);
15  }
16 }
```

Ce code a pour résultat la figure suivante.

Liste dans une boîte de dialogue

Voici un petit détail des paramètres utilisés dans cette méthode :

- les quatre premiers, vous connaissez ;
- le deuxième `null` correspond à l'icône que vous souhaitez passer ;
- ensuite, vous devez passer un tableau de `String` afin de remplir la combo (l'objet `JComboBox`) de la boîte ;
- le dernier paramètre correspond à la valeur par défaut de la liste déroulante.

qu'elle prend un paramètre supplémentaire et que le type de retour n'est pas un objet mais un entier.

Ce type de boîte propose un choix de boutons correspondant aux éléments passés en paramètres (tableau de `String`) au lieu d'une combo ; elle prend aussi une valeur par défaut, mais retourne l'indice de l'élément dans la liste au lieu de l'élément lui-même.

Je pense que vous vous y connaissez assez pour comprendre le code suivant :

```
1 import javax.swing.JOptionPane;
2
3 public class Test {
4   public static void main(String[] args) {
5     String[] sexe = {"masculin", "féminin", "indéterminé"};
6     JOptionPane jop = new JOptionPane(), jop2 = new JOptionPane();
7     int rang = jop.showOptionDialog(null,
8       "Veuillez indiquer votre sexe !",
9       "Gendarmerie nationale !",
10      JOptionPane.YES_NO_CANCEL_OPTION,
11      JOptionPane.QUESTION_MESSAGE,
12      null,
13      sexe,
14      sexe[2]);
15    jop2.showMessageDialog(null, "Votre sexe est " + sexe[rang], "Etat civil", JOptionPane.INFORMATION_MESSAGE);
16  }
17 }
```

Ce qui nous donne la figure suivante.

Boîte multi-boutons

Voilà, vous en avez terminé avec les boîtes de saisie. Cependant, vous avez dû vous demander s'il n'était pas possible d'ajouter des composants à ces boîtes. C'est vrai : vous pourriez avo besoin de plus de renseignements, sait-on jamais… Je vous propose donc de voir comment créer vos propres boîtes de dialogue !

Des boîtes de dialogue personnalisées

Je me doute que vous êtes impatients de faire vos propres boîtes de dialogue. Comme il est vrai que dans certains cas, vous en aurez besoin, allons-y gaiement ! Je vais vous révéler un se bien gardé : les boîtes de dialogue héritent de la classe `JDialog`. Vous avez donc deviné que nous allons créer une classe dérivée de cette dernière.

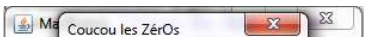
Commençons par créer un nouveau projet. Créez une nouvelle classe dans Eclipse, appelons-la `ZDialog`, faites-la hériter de la classe citée précédemment, et mettez-y le code suivant :

```
1 import javax.swing.JDialog;
2 import javax.swing.JFrame;
3
4 public class ZDialog extends JDialog {
5   public ZDialog(JFrame parent, String title, boolean modal){
6     //On appelle le construteur de JDialog correspondant
7     super(parent, title, modal);
8     //On spécifie une taille
9     this.setSize(200, 80);
10    //La position
11    this.setLocationRelativeTo(null);
12    //La boîte ne devra pas être redimensionnable
13    this.setResizable(false);
14    //Enfin on l'affiche
15    this.setVisible(true);
16    //Tout ceci ressemble à ce que nous faisons depuis le début avec notre JFrame.
17  }
18 }
```

Maintenant, créons une classe qui va tester notre `ZDialog` :

```
1 import java.awt.FlowLayout;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import javax.swing.JButton;
5 import javax.swing.JFrame;
6
7 public class Fenetre extends JFrame {
8   private JButton bouton = new JButton("Appel à la ZDialog");
9
10  public Fenetre(){
11    this.setTitle("Ma JFrame");
12    this.setSize(300, 100);
13    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14    this.setLocationRelativeTo(null);
15    this.getContentPane().setLayout(new FlowLayout());
16    this.getContentPane().add(bouton);
17    bouton.addActionListener(new ActionListener(){
18      public void actionPerformed(ActionEvent arg0) {
19        ZDialog zd = new ZDialog(null, "Coucou les ZérOs", true);
20      }
21    });
22
23    this.setVisible(true);
24  }
25
26  public static void main(String[] main){
27    Fenetre fen = new Fenetre();
28  }
29 }
```

La figure suivante vous présente le résultat ; bon, c'est un début.



Votre première boîte personnalisée

Je pense que vous avez deviné le rôle des paramètres du constructeur, mais je vais tout de même les expliciter :

- `JFrame Parent` correspond à l'objet parent ;
- `String title` correspond au titre de notre boîte ;
- `boolean modal` correspond à la modalité ; `true` : boîte modale, `false` : boîte non modale.

Rien de compliqué… Il est donc temps d'ajouter des composants à notre objet. Par contre, vous conviendrez que si nous prenons la peine de construire un tel composant, nous attendons qu'une simple réponse à une question ouverte (oui/non), une chaîne de caractères ou encore un choix dans une liste… Nous en voulons bien plus ! Plusieurs saisies, avec plusieurs listes er temps !

Vous avez vu que nous devrons récupérer les informations choisies dans certains cas, mais pas dans tous : nous allons donc devoir déterminer ces différents cas, ainsi que les choses à fai

Partons du fait que notre boîte comprendra un bouton OK et un bouton `Annuler` : dans le cas où l'utilisateur clique sur OK, on récupère les informations, si l'utilisateur clique sur `Annul` ne récupère rien. Et il faudra aussi tenir compte de la modalité de notre boîte : la méthode `setVisible(false);` met fin au dialogue ! Ceci signifie également que le dialogue s'entam moment où l'instruction `setVisible(true);` est exécutée. C'est pourquoi nous allons sortir cette instruction du constructeur de l'objet et la mettre dans une méthode à part.

Maintenant, il faut que l'on puisse indiquer à notre boîte de renvoyer les informations ou non. C'est pour cela que nous allons utiliser un booléen - appelons-le `sendData` - initialisé à `fal` mais qui passera à `true` si on clique sur OK.

```
1 //Cas où notre ZDialog renverra le contenu
2 //D'un JTextField nommé jtf
3 public String showZDialog(){
4   this.sendData = false;
5   //Début du dialogue
6   this.setVisible(true);
7   //Le dialogue prend fin
8   //Si on a cliqué sur OK, on envoie, sinon on envoie une chaîne vide !
9  return (this.sendData)? jtf.getText() : "";
10 }
```

Il nous reste un dernier point à gérer.

Comment récupérer les informations saisies dans notre boîte depuis notre fenêtre, vu que nous voulons obtenir plusieurs informations ?

C'est vrai qu'on ne peut retourner qu'une valeur à la fois. Mais il peut y avoir plusieurs solutions à ce problème :

- Dans le cas où nous n'avons qu'un composant, nous pouvons adapter la méthode `showZDialog()` au type de retour du composant utilisé.
- Dans notre cas, nous voulons plusieurs composants, donc plusieurs valeurs. Vous pouvez :
  - retourner une collection de valeurs (`ArrayList`) ;
  - faire des accesseurs dans votre `ZDialog` ;
  - créer un objet dont le rôle est de collecter les informations dans votre boîte et de retourner cet objet ;
  - etc.

Nous allons opter pour un objet qui collectera les informations et que nous retournerons à la fin de la méthode `showZDialog()`. Avant de nous lancer dans sa création, nous devons sa que nous allons mettre dans notre boîte… J'ai choisi de vous faire programmer une boîte permettant de spécifier les caractéristiques d'un personnage de jeu vidéo :

- son nom (un champ de saisie) ;
- son sexe (une combo) ;
- sa taille (un champ de saisie) ;
- sa couleur de cheveux (une combo) ;
- sa tranche d'âge (des boutons radios).

Pour ce qui est du placement des composants, l'objet `JDialog` se comporte exactement comme un objet `JFrame` (`BorderLayout` par défaut, ajout d'un composant au conteneur…).

Nous pouvons donc créer notre objet contenant les informations de notre boîte de dialogue, je l'ai appelé `ZDialogInfo`.

```
1 public class ZDialogInfo {
2   private String nom, sexe, age, cheveux, taille;
3
4   public ZDialogInfo(){}
5   public ZDialogInfo(String nom, String sexe, String age, String cheveux, String taille){
6     this.nom = nom;
7     this.sexe = sexe;
8     this.age = age;
9     this.cheveux = cheveux;
10    this.taille = taille;
11  }
12
13  public String toString(){
14    String str;
15    if(this.nom != null && this.sexe != null && this.taille != null && this.age != null && this.cheveux != null){
16      str = "Description de l'objet InfoZDialog";
```

```
    else{
      str = "Aucune information !";
    }
    return str;
  }
}
```

L'avantage avec cette méthode, c'est que nous n'avons pas à nous soucier d'une éventuelle annulation de la saisie : l'objet d'information renverra toujours quelque chose.

Voici le code source de notre boîte perso :

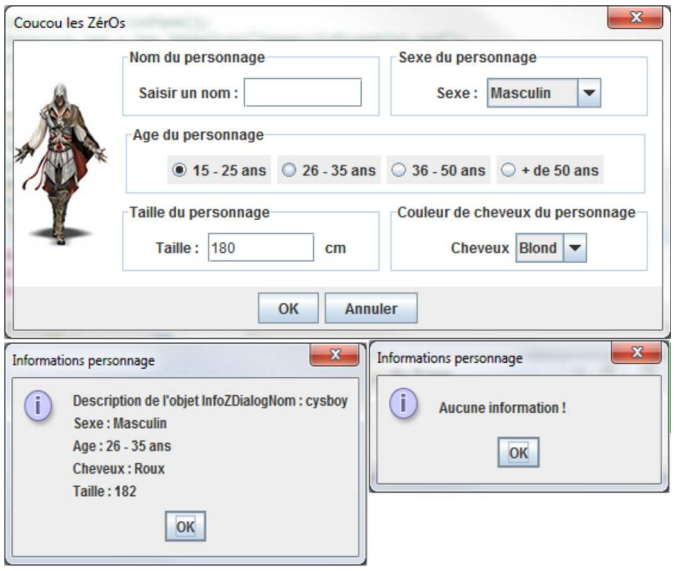
```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JTextField;

public class ZDialog extends JDialog {
  private ZDialogInfo zInfo = new ZDialogInfo();
  private boolean sendData;
  private JLabel nomLabel, sexeLabel, cheveuxLabel, ageLabel, tailleLabel,taille2Label, icon;
  private JRadioButton tranche1, tranche2, tranche3, tranche4;
  private JComboBox sexe, cheveux;
  private JTextField nom, taille;

  public ZDialog(JFrame parent, String title, boolean modal){
    super(parent, title, modal);
    this.setSize(550, 270);
    this.setLocationRelativeTo(null);
    this.setResizable(false);
    this.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
    this.initComponent();
  }

  public ZDialogInfo showZDialog(){
    this.sendData = false;
    this.setVisible(true);
    return this.zInfo;
  }

  private void initComponent(){
    //Icône
    icon = new JLabel(new ImageIcon("images/icone.jpg"));
    JPanel panIcon = new JPanel();
    panIcon.setBackground(Color.white);
    panIcon.setLayout(new BorderLayout());
    panIcon.add(icon);

    //Le nom
    JPanel panNom = new JPanel();
    panNom.setBackground(Color.white);
    panNom.setPreferredSize(new Dimension(220, 60));
    nom = new JTextField();
    nom.setPreferredSize(new Dimension(100, 25));
    panNom.setBorder(BorderFactory.createTitledBorder("Nom du personnage"));
    nomLabel = new JLabel("Saisir un nom :");
    panNom.add(nomLabel);
    panNom.add(nom);

    //Le sexe
    JPanel panSexe = new JPanel();
    panSexe.setBackground(Color.white);
    panSexe.setPreferredSize(new Dimension(220, 60));
    panSexe.setBorder(BorderFactory.createTitledBorder("Sexe du personnage"));
    sexe = new JComboBox();
    sexe.addItem("Masculin");
    sexe.addItem("Féminin");
    sexe.addItem("Indéterminé");
    sexeLabel = new JLabel("Sexe : ");
    panSexe.add(sexeLabel);
    panSexe.add(sexe);

    //L'âge
    JPanel panAge = new JPanel();
    panAge.setBackground(Color.white);
    panAge.setBorder(BorderFactory.createTitledBorder("Age du personnage"));
    panAge.setPreferredSize(new Dimension(440, 60));
    tranche1 = new JRadioButton("15 - 25 ans");
    tranche1.setSelected(true);
    tranche2 = new JRadioButton("26 - 35 ans");
    tranche3 = new JRadioButton("36 - 50 ans");
    tranche4 = new JRadioButton("+ de 50 ans");
    ButtonGroup bg = new ButtonGroup();
    bg.add(tranche1);
    bg.add(tranche2);
    bg.add(tranche3);
    bg.add(tranche4);
    panAge.add(tranche1);
    panAge.add(tranche2);
    panAge.add(tranche3);
    panAge.add(tranche4);

    //La taille
    JPanel panTaille = new JPanel();
    panTaille.setBackground(Color.white);
    panTaille.setPreferredSize(new Dimension(220, 60));
    panTaille.setBorder(BorderFactory.createTitledBorder("Taille du personnage"));
    tailleLabel = new JLabel("Taille : ");
    taille2Label = new JLabel(" cm");
```

```
    //La couleur des cheveux
    JPanel panCheveux = new JPanel();
    panCheveux.setBackground(Color.white);
    panCheveux.setPreferredSize(new Dimension(220, 60));
    panCheveux.setBorder(BorderFactory.createTitledBorder("Couleur de cheveux du personnage"));
    cheveux = new JComboBox();
    cheveux.addItem("Blond");
    cheveux.addItem("Brun");
    cheveux.addItem("Roux");
    cheveux.addItem("Blanc");
    cheveuxLabel = new JLabel("Cheveux");
    panCheveux.add(cheveuxLabel);
    panCheveux.add(cheveux);

    JPanel content = new JPanel();
    content.setBackground(Color.white);
    content.add(panNom);
    content.add(panSexe);
    content.add(panAge);
    content.add(panTaille);
    content.add(panCheveux);

    JPanel control = new JPanel();
    JButton okBouton = new JButton("OK");

    okBouton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent arg0) {
        zInfo = new ZDialogInfo(nom.getText(), (String)sexe.getSelectedItem(), getAge(), (String)cheveux.getSelectedItem() ,getTaille());
        setVisible(false);
      }

      public String getAge(){
        return (tranche1.isSelected()) ? tranche1.getText() :
               (tranche2.isSelected()) ? tranche2.getText() :
               (tranche3.isSelected()) ? tranche3.getText() :
               (tranche4.isSelected()) ? tranche4.getText() :
                tranche1.getText();
      }

      public String getTaille(){
        return (taille.getText().equals("")) ? "180" : taille.getText();
      }
    });

    JButton cancelBouton = new JButton("Annuler");
    cancelBouton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent arg0) {
        setVisible(false);
      }
    });

    control.add(okBouton);
    control.add(cancelBouton);

    this.getContentPane().add(panIcon, BorderLayout.WEST);
    this.getContentPane().add(content, BorderLayout.CENTER);
    this.getContentPane().add(control, BorderLayout.SOUTH);
  }
}
```

J'ai ajouté une image, mais vous n'y êtes nullement obligés ! Voici le code source permettant de tester cette boîte :

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Fenetre extends JFrame {
  private JButton bouton = new JButton("Appel à la ZDialog");

  public Fenetre(){
    this.setTitle("Ma JFrame");
    this.setSize(300, 100);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLocationRelativeTo(null);
    this.getContentPane().setLayout(new FlowLayout());
    this.getContentPane().add(bouton);
    bouton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent arg0) {
        ZDialog zd = new ZDialog(null, "Coucou les ZérOs", true);
        ZDialogInfo zInfo = zd.showZDialog();
        JOptionPane jop = new JOptionPane();
        jop.showMessageDialog(null, zInfo.toString(), "Informations personnage", JOptionPane.INFORMATION_MESSAGE);
      }
    });
    this.setVisible(true);
  }

  public static void main(String[] main){
    Fenetre fen = new Fenetre();
  }
}
```

Ce qu'on obtient est montré à la figure suivante.



Différentes copies d'écran de test

Voilà : nous venons de voir comment utiliser des boîtes de dialogue. En route pour l'utilisation des menus, à présent !

Les menus

Faire son premier menu

Vous vous rappelez que j'ai mentionné qu'une `MenuBar` fait partie de la composition de l'objet `JFrame`. Le moment est venu pour vous d'utiliser un composant de ce genre. Néanmoins, celui-ci appartient au package `java.awt`. Dans ce chapitre nous utiliserons son homologue, l'objet `JMenuBar`, issu dans le package `javax.swing`. Pour travailler avec des menus, n aurons besoin :

- de l'objet `JMenu`, le titre principal d'un point de menu (Fichier, Édition…) ;
- d'objets `JMenuItem`, les éléments composant nos menus.

Afin de permettre des interactions avec nos futurs menus, nous allons devoir implémenter l'interface `ActionListener` que vous connaissez déjà bien. Ces implémentations serviront à écouter les objets `JMenuItem` : ce sont ces objets qui déclencheront l'une ou l'autre opération. Les `JMenu`, eux, se comportent automatiquement : si on clique sur un titre de menu, celui déroule tout seul et, dans le cas où nous avons un tel objet présent dans un autre `JMenu`, une autre liste se déroulera toute seule !

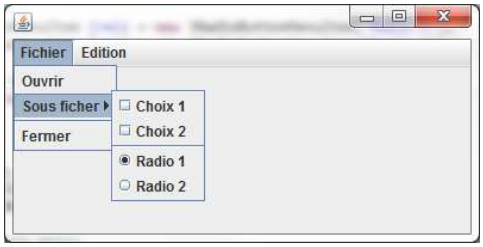
Je vous propose d'enlever tous les composants (boutons, combos, etc.) de notre animation et de gérer tout cela par le biais d'un menu.

Avant de nous lancer dans cette tâche, voici une application de tout cela, histoire de vous familiariser avec les concepts et leur syntaxe.

```
1 import java.awt.event.ActionEvent;
2 import java.awt.event.ActionListener;
3 import javax.swing.ButtonGroup;
4 import javax.swing.JCheckBoxMenuItem;
5 import javax.swing.JFrame;
6 import javax.swing.JMenu;
7 import javax.swing.JMenuBar;
8 import javax.swing.JMenuItem;
9 import javax.swing.JRadioButtonMenuItem;
10
11 public class ZFenetre extends JFrame {
12   private JMenuBar menuBar = new JMenuBar();
13   private JMenu test1 = new JMenu("Fichier");
14   private JMenu test1_2 = new JMenu("Sous ficher");
15   private JMenu test2 = new JMenu("Edition");
16
17   private JMenuItem item1 = new JMenuItem("Ouvrir");
18   private JMenuItem item2 = new JMenuItem("Fermer");
19   private JMenuItem item3 = new JMenuItem("Lancer");
20   private JMenuItem item4 = new JMenuItem("Arrêter");
21
22   private JCheckBoxMenuItem jcmi1 = new JCheckBoxMenuItem("Choix 1");
23   private JCheckBoxMenuItem jcmi2 = new JCheckBoxMenuItem("Choix 2");
24
25   private JRadioButtonMenuItem jrmi1 = new JRadioButtonMenuItem("Radio 1");
26   private JRadioButtonMenuItem jrmi2 = new JRadioButtonMenuItem("Radio 2");
27
28   public static void main(String[] args){
29     ZFenetre zFen = new ZFenetre();
30   }
31
32   public ZFenetre(){
33     this.setSize(400, 200);
34     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
35     this.setLocationRelativeTo(null);
36
37     //On initialise nos menus
38     this.test1.add(item1);
39
40     //On ajoute les éléments dans notre sous-menu
41     this.test1_2.add(jcmi1);
42     this.test1_2.add(jcmi2);
43     //Ajout d'un séparateur
44     this.test1_2.addSeparator();
45     //On met nos radios dans un ButtonGroup
```

```
52      this.test1_2.add(jrmi1);
53      this.test1_2.add(jrmi2);
54
55      //Ajout du sous-menu dans notre menu
56      this.test1.add(this.test1_2);
57      //Ajout d'un séparateur
58      this.test1.addSeparator();
59      item2.addActionListener(new ActionListener(){
60        public void actionPerformed(ActionEvent arg0) {
61          System.exit(0);
62        }
63      });
64      this.test1.add(item2);
65      this.test2.add(item3);
66      this.test2.add(item4);
67
68      //L'ordre d'ajout va déterminer l'ordre d'apparition dans le menu de gauche à droite
69      //Le premier ajouté sera tout à gauche de la barre de menu et inversement pour le dernier
70      this.menuBar.add(test1);
71      this.menuBar.add(test2);
72      this.setJMenuBar(menuBar);
73      this.setVisible(true);
74    }
75  }
```

L'action attachée au `JMenutItemFermer` permet de quitter l'application. Ce que donne le code est affiché à la figure suivante.



Premier menu

Vous voyez qu'il n'y a rien de difficile dans l'élaboration d'un menu. Je vous propose donc d'en créer un pour notre animation. Allons-y petit à petit : nous ne gérerons les événements que p suite. Pour le moment, nous allons avoir besoin :

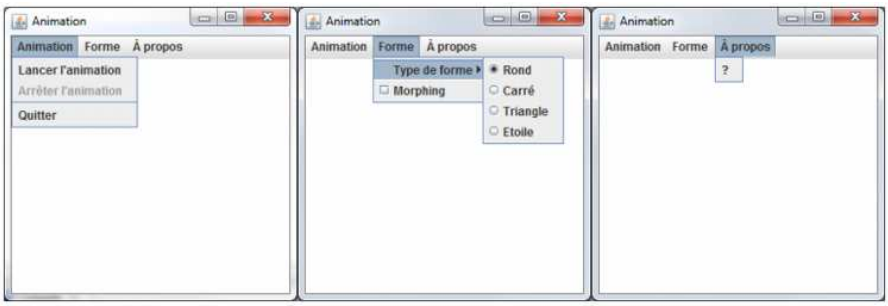
- d'un menu `Animation` pour lancer, interrompre (par défaut à `setEnabled(false)`) ou quitter l'animation ;
- d'un menu `Forme` afin de sélectionner le type de forme utiliser (sous-menu + une radio par forme) et de permettre d'activer le mode morphing (case à cocher) ;
- d'un menu `À propos` avec un joli « ? » qui va ouvrir une boîte de dialogue.

N'effacez surtout pas les implémentations pour les événements : retirez seulement les composants qui les utilisent. Ensuite, créez votre menu !

Voici un code qui ne devrait pas trop différer de ce que vous avez écrit :

```
 1 import java.awt.BorderLayout;
 2 import java.awt.Color;
 3 import java.awt.event.ActionEvent;
 4 import java.awt.event.ActionListener;
 5 import javax.swing.ButtonGroup;
 6 import javax.swing.JButton;
 7 import javax.swing.JCheckBox;
 8 import javax.swing.JCheckBoxMenuItem;
 9 import javax.swing.JComboBox;
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import javax.swing.JMenu;
13 import javax.swing.JMenuBar;
14 import javax.swing.JMenuItem;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17 import javax.swing.JRadioButtonMenuItem;
18
19 public class Fenetre extends JFrame{
20   private Panneau pan = new Panneau();
21   private JPanel container = new JPanel();
22   private int compteur = 0;
23   private boolean animated = true;
24   private boolean backX, backY;
25   private int x,y ;
26   private Thread t;
27
28   private JMenuBar menuBar = new JMenuBar();
29
30   private JMenu animation = new JMenu("Animation"),
31     forme = new JMenu("Forme"),
32     typeForme = new JMenu("Type de forme"),
33     aPropos = new JMenu("À propos");
34
35   private JMenuItem lancer = new JMenuItem("Lancer l'animation"),
36     arreter = new JMenuItem("Arrêter l'animation"),
37     quitter = new JMenuItem("Quitter"),
38     aProposItem = new JMenuItem("?");
39
40   private JCheckBoxMenuItem morph = new JCheckBoxMenuItem("Morphing");
41   private JRadioButtonMenuItem carre = new JRadioButtonMenuItem("Carré"),
42     rond = new JRadioButtonMenuItem("Rond"),
43     triangle = new JRadioButtonMenuItem("Triangle"),
44     etoile = new JRadioButtonMenuItem("Etoile");
45
46   private ButtonGroup bg = new ButtonGroup();
47
48   public Fenetre(){
49     this.setTitle("Animation");
```

```
56    container.add(pan, BorderLayout.CENTER);
57
58    this.setContentPane(container);
59    this.initMenu();
60    this.setVisible(true);
61  }
62
63  private void initMenu(){
64    //Menu animation
65    animation.add(lancer);
66    arreter.setEnabled(false);
67    animation.add(arreter);
68    animation.addSeparator();
69    //Pour quitter l'application
70    quitter.addActionListener(new ActionListener(){
71      public void actionPerformed(ActionEvent event){
72        System.exit(0);
73      }
74    });
75    animation.add(quitter);
76
77    //Menu forme
78    bg.add(carre);
79    bg.add(triangle);
80    bg.add(rond);
81    bg.add(etoile);
82
83    typeForme.add(rond);
84    typeForme.add(carre);
85    typeForme.add(triangle);
86    typeForme.add(etoile);
87
88    rond.setSelected(true);
89
90    forme.add(typeForme);
91    forme.add(morph);
92
93    //Menu À propos
94    aPropos.add(aProposItem);
95
96    //Ajout des menus dans la barre de menus
97    menuBar.add(animation);
98    menuBar.add(forme);
99    menuBar.add(aPropos);
100
101   //Ajout de la barre de menus sur la fenêtre
102   this.setJMenuBar(menuBar);
103 }
104
105 private void go(){
106   //Rien n'a changé ici
107 }
108
109 public class BoutonListener implements ActionListener{
110   public void actionPerformed(ActionEvent arg0) {
111     JOptionPane jop = new JOptionPane();
112     int option = jop.showConfirmDialog(null,
113       "Voulez-vous lancer l'animation ?",
114       "Lancement de l'animation",
115       JOptionPane.YES_NO_OPTION,
116       JOptionPane.QUESTION_MESSAGE);
117
118     if(option == JOptionPane.OK_OPTION){
119       lancer.setEnabled(false);
120       arreter.setEnabled(true);
121       animated = true;
122       t = new Thread(new PlayAnimation());
123       t.start();
124     }
125   }
126 }
127
128 class Bouton2Listener  implements ActionListener{
129   public void actionPerformed(ActionEvent e) {
130     JOptionPane jop = new JOptionPane();
131     int option = jop.showConfirmDialog(null,
132       "Voulez-vous arrêter l'animation ?",
133       "Arrêt de l'animation",
134       JOptionPane.YES_NO_CANCEL_OPTION,
135       JOptionPane.QUESTION_MESSAGE);
136
137     if(option != JOptionPane.NO_OPTION && option != JOptionPane.CANCEL_OPTION && option != JOptionPane.CLOSED_OPTION){
138       animated = false;
139       //On remplace nos boutons par nos JMenuItem
140       lancer.setEnabled(true);
141       arreter.setEnabled(false);
142     }
143   }
144 }
145
146 class PlayAnimation implements Runnable{
147   public void run() {
148     go();
149   }
150 }
151
152 class FormeListener implements ActionListener{
153   public void actionPerformed(ActionEvent e) {
154     //On commente cette ligne pour l'instant
155     //***************************************
156     //pan.setForme(combo.getSelectedItem().toString());
157   }
158 }
159
160 class MorphListener implements ActionListener{
161   public void actionPerformed(ActionEvent e) {
162     //Si la case est cochée, activation du mode morphing
163     if(morph.isSelected())pan.setMorph(true);
164       //Sinon rien !
165     else pan.setMorph(false);
166   }
```



Notre menu et son animation

Il ne reste plus qu'à faire communiquer nos menus et notre animation ! Pour cela, rien de plus simple, il suffit d'indiquer à nos `MenuItem` qu'on les écoute. En fait, cela revient à faire com nous cliquions sur des boutons (à l'exception des cases à cocher et des radios où, là, nous pouvons utiliser une implémentation d'`ActionListener` ou de `ItemListener`), nous utili donc la première méthode.

Afin que l'application fonctionne bien, j'ai apporté deux modifications mineures dans la classe `Panneau`. J'ai ajouté une instruction dans une condition :

```
1 //J'ai ajouté :  || this.forme.equals("CARRÉ")
2 if(this.forme.equals("CARRE") || this.forme.equals("CARRÉ")){
3   g.fillRect(posX, posY, 50, 50);
4 }
```

… ainsi, on accepte les deux graphies ! J'ai également ajouté un `toUpperCase()` :

```
1 public void setForme(String form){
2   this.forme = form.toUpperCase();
3 }
```

Ainsi, on s'assure que cette chaîne de caractères est en majuscules.

Voici le code de notre animation avec un beau menu pour tout contrôler :

```
1 //Les imports
2
3 public class Fenetre extends JFrame{
4
5   //La déclaration des variables reste inchangée
6
7   public Fenetre(){
8     //Le constructeur est inchangé
9   }
10
11   private void initMenu(){
12     //Menu Animation
13     //Ajout du listener pour lancer l'animation
14     lancer.addActionListener(new StartAnimationListener());
15     animation.add(lancer);
16
17     //Ajout du listener pour arrêter l'animation
18     arreter.addActionListener(new StopAnimationListener());
19     arreter.setEnabled(false);
20     animation.add(arreter);
21
22     animation.addSeparator();
23     quitter.addActionListener(new ActionListener(){
24       public void actionPerformed(ActionEvent event){
25         System.exit(0);
26       }
27     });
28     animation.add(quitter);
29
30     //Menu Forme
31
32     bg.add(carre);
33     bg.add(triangle);
34     bg.add(rond);
35     bg.add(etoile);
36
37     //On crée un nouvel écouteur, inutile de créer 4 instances différentes
38     FormeListener fl = new FormeListener();
39     carre.addActionListener(fl);
40     rond.addActionListener(fl);
41     triangle.addActionListener(fl);
42     etoile.addActionListener(fl);
43
44     typeForme.add(rond);
45     typeForme.add(carre);
46     typeForme.add(triangle);
47     typeForme.add(etoile);
48
49     rond.setSelected(true);
50
51     forme.add(typeForme);
52
53     //Ajout du listener pour le morphing
54     morph.addActionListener(new MorphListener());
55     forme.add(morph);
56
57     //Menu À propos
58
59     //Ajout de ce que doit faire le "?"
60     aProposItem.addActionListener(new ActionListener(){
61       public void actionPerformed(ActionEvent arg0) {
62         JOptionPane jop = new JOptionPane();
63         ImageIcon img = new ImageIcon("images/cysboy.gif");
64         String mess = "Merci ! \n J'espère que vous vous amusez bien !\n";
65         mess += "Je crois qu'il est temps d'ajouter des accélérateurs et des "+" mnémoniques dans tout ça…\n";
```

```
72      //Ajout des menus dans la barre de menus
73      menuBar.add(animation);
74      menuBar.add(forme);
75      menuBar.add(aPropos);
76
77      //Ajout de la barre de menus sur la fenêtre
78      this.setJMenuBar(menuBar);
79    }
80
81    private void go(){
82      //Idem
83    }
84
85    public class StartAnimationListener implements ActionListener{
86      public void actionPerformed(ActionEvent arg0) {
87        //Idem
88      }
89    }
90
91    /**
92     * Écouteur du menu Quitter
93     * @author CHerby
94     */
95    class StopAnimationListener  implements ActionListener{
96      public void actionPerformed(ActionEvent e) {
97        //Idem
98      }
99    }
100
101   class PlayAnimation implements Runnable{
102     public void run() {
103       go();
104     }
105   }
106
107   /**
108    * Écoute les menus Forme
109    * @author CHerby
110    */
111   class FormeListener implements ActionListener{
112     public void actionPerformed(ActionEvent e) {
113       pan.setForme(((JRadioButtonMenuItem)e.getSource()).getText());
114     }
115   }
116
117   /**
118    * Écoute le menu Morphing
119    * @author CHerby
120    */
121   class MorphListener implements ActionListener{
122     public void actionPerformed(ActionEvent e) {
123       //Si la case est cochée, activation du mode morphing
124       if(morph.isSelected()) pan.setMorph(true);
125       //Sinon rien !
126       else pan.setMorph(false);
127     }
128   }
129 }
```

Comme je l'ai indiqué dans le dialogue du menu `À propos`, je crois qu'il est temps d'ajouter des raccourcis clavier à notre application ! Vous êtes prêts ?

Les raccourcis clavier

À nouveau, il est très simple d'insérer des raccourcis clavier. Pour ajouter un « accélérateur » (raccourcis clavier des éléments de menu) sur un `JMenu`, nous appellerons la méthode `setAccelerator();` et pour ajouter un mnémonique (raccourcis permettant de simuler le clic sur un point de menu) sur un `JMenuItem`, nous nous servirons de la méthode `setMnemonic();`.

Attribuons le mnémonique « A » au menu `Animation`, le mnémonique « F » pour le menu `Forme` et enfin « P » pour `À Propos`. Vous allez voir, c'est très simple : il vous suffit d'invoquer méthode `setMnemonic(char mnemonic);` sur le `JMenu` que vous désirez.

Ce qui nous donne, dans notre cas :

```
1  private void initMenu(){
2    //Menu animation
3    //Le début de la méthode reste inchangé
4
5    //Ajout des menus dans la barre de menus et ajout de mnémoniques !
6    animation.setMnemonic('A');
7    menuBar.add(animation);
8
9    forme.setMnemonic('F');
10   menuBar.add(forme);
11
12   aPropos.setMnemonic('P');
13   menuBar.add(aPropos);
14   //Ajout de la barre de menus sur la fenêtre
15   this.setJMenuBar(menuBar);
16 }
```

Nous avons à présent les lettres correspondant au mnémonique soulignées dans nos menus. Et il y a mieux : si vous tapez `ALT` + `<la lettre>`, le menu correspondant se déroule ! La f suivante correspond à ce que j'obtiens.

Mnémonique sur votre menu

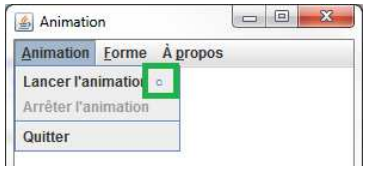
Sachez que vous pouvez aussi mettre des mnémoniques sur les objets `JMenuItem`. Je dois également vous dire qu'il existe une autre façon d'ajouter un mnémonique sur un `JMenu` (ma uniquement valable avec un `JMenu`) : en passant le mnémonique en deuxième paramètre du constructeur de l'objet, comme ceci :

un simple caractère comme accélérateur à notre `JMenuItemLancer` en utilisant la méthode `getKeyStroke(char caracter);` de l'objet `KeyStroke`.

Ajoutez cette ligne de code au début de la méthode `initMenu()` (vous aurez besoin des packages `javax.swing.KeyStroke` et `java.awt.event.ActionEvent`) :

```
1 //Cette instruction ajoute l'accélérateur 'c' à notre objet
2 lancer.setAccelerator(KeyStroke.getKeyStroke('c'));
```

Testez votre application, un petit « c » est apparu à côté du menu `Lancer`. La figure suivante illustre le phénomène.



Un accélérateur sur votre menu

Appuyez sur la touche c de votre clavier : celle-ci a le même effet qu'un clic sur le menu « Lancer » !

Si vous mettez le caractère « C », vous serez obligés d'appuyer simultanément sur SHIFT + c ou d'activer la touche MAJ !

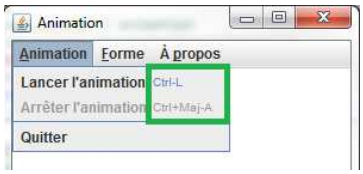
Si le principe est bon, dites-vous aussi que maintenant, presser la touche c lancera systématiquement votre animation ! C'est l'une des raisons pour laquelle les accélérateurs sont, en gén des combinaisons de touches du genre CTRL + c ou encore CTRL + SHIFT + S .

Pour cela, nous allons utiliser une méthode `getKeyStroke()` un peu différente : elle ne prendra pas le caractère de notre touche en argument, mais son code ainsi qu'une ou plusieurs touche(s) composant la combinaison. Pour obtenir le code d'une touche, nous utiliserons l'objet `KeyEvent`, un objet qui stocke tous les codes des touches !

Dans le code qui suit, je crée un accélérateur CTRL + L pour le menu `Lancer` et un accélérateur CTRL + SHIFT + A pour le menu `Arrêter` :

```
1 lancer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L, KeyEvent.CTRL_MASK));
2 animation.add(lancer);
3 //Ajout du listener pour arrêter l'animation
4 arreter.addActionListener(new StopAnimationListener());
5 arreter.setEnabled(false);
6 arreter.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A, KeyEvent.CTRL_DOWN_MASK + KeyEvent.SHIFT_DOWN_MASK));
7 animation.add(arreter);
```

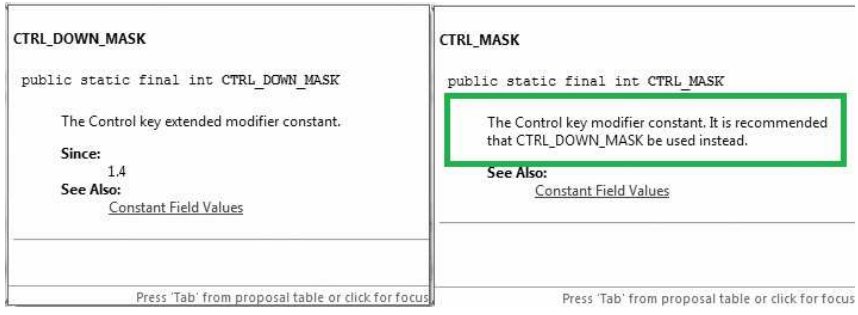
La figure suivante présente le résultat obtenu.



Combinaison de touches pour un accélérateur

J'imagine que vous êtes perturbés par `KeyEvent.VK_L` et les appels du même genre. En fait, la classe `KeyEvent` répertorie tous les codes de toutes les touches du clavier. Une grand majorité d'entre eux sont sous la forme VK_<le caractère ou le nom de la touche>. Lisez-le ainsi : Value of Key <nom de la touche>.
À part certaines touches de contrôle comme CTRL , ALT , SHIFT , etc. vous pouvez facilement retrouver le code d'une touche grâce à cet objet !

Ensuite, vous avez dû remarquer qu'en tapant `KeyEvent.CTRL_DOWN_MASK`, Eclipse vous a proposé quasiment la même chose (figure suivante).



Versions différentes

Vous pouvez aisément voir qu'Eclipse vous dit que la version `CTRL_DOWN_MASK` est la plus récente et qu'il est vivement conseillé de l'utiliser ! Vous voilà donc avec un menu comprenan mnémoniques et des accélérateurs. Il est maintenant temps de voir comment créer un menu contextuel !

Faire un menu contextuel

Vous avez déjà fait le plus dur, je suis sûr que vous n'allez pas tarder à vous en rendre compte. Nous allons simplement utiliser un autre objet, un `JPopupMenu`, dans lequel nous mettron `JMenuItem` ou/et `JMenu`. Bon il faudra tout de même indiquer à notre menu contextuel comment et où s'afficher, mais vous verrez que c'est très simple. Maintenant que vous commenc bien connaître les bases de la programmation événementielle, nous passons à la vitesse supérieure !

Les points importants de notre menu contextuel

- Il ne doit s'afficher que lorsqu'on fait un clic droit, et rien d'autre !

Nous allons mettre dans notre menu contextuel les actions « Lancer l'animation », « Arrêter l'animation » ainsi que deux nouveautés :

- changer la couleur du fond de notre animation ;
- changer la couleur de notre forme.

Avant d'implémenter les deux nouvelles fonctionnalités, nous allons travailler sur les deux premières.

Lorsque nous lancerons l'animation, nous devrons mettre les deux menus `Lancer l'animation` dans l'état `setEnabled(false);` et les deux menus `Arrêter l'animatio` l'état `setEnabled(true);` (et pour l'arrêter, il faudra faire l'inverse).

Comme je vous l'ai dit plus haut, nous allons utiliser le même objet qui écoute pour les deux menus. Il nous faudra créer une véritable instance de ces objets et signaler à l'application que objets écoutent non seulement le menu du haut, mais aussi le menu contextuel.

Nous avons parfaitement le droit de le faire : plusieurs objets peuvent écouter un même composant et plusieurs composants peuvent avoir le même objet qui les écoute ! Vous êtes presqu à créer votre menu contextuel, il ne vous manque que ces informations :

- comment indiquer à notre panneau quand et où afficher le menu contextuel ;
- comment lui spécifier qu'il doit le faire uniquement suite à un clic droit.

Le déclenchement de l'affichage du pop-up doit se faire lors d'un clic de souris. Vous connaissez une interface qui gère ce type d'événement : l'interface `MouseListener`. Nous allons d indiquer à notre panneau qu'un objet du type de cette interface va l'écouter !

Tout comme dans le chapitre sur les zones de saisie, il existe une classe qui contient toutes les méthodes de ladite interface : la classe `MouseAdapter`. Vous pouvez implémenter celle-ci afin de ne redéfinir que la méthode dont vous avez besoin ! C'est cette solution que nous allons utiliser.

Si vous préférez, vous pouvez utiliser l'événement `mouseClicked`, mais je pensais plutôt à `mouseReleased()`, pour une raison simple à laquelle vous n'avez peut-être pas pensé : si deux événements sont quasiment identiques, dans un certain cas, seul l'événement `mouseClicked()` sera appelé. Il s'agit du cas où vous cliquez sur une zone, déplacez votre souris e dehors de la zone tout en maintenant le clic et relâchez le bouton de la souris. C'est pour cette raison que je préfère utiliser la méthode `mouseReleased()`. Ensuite, pour préciser où affi menu contextuel, nous allons utiliser la méthode `show(Component invoker, int x, int y);` de la classe `JPopupMenu` :

- `Component invoker` : désigne l'objet invoquant le menu contextuel, dans notre cas, l'instance de `Panneau`.
- `int x` : coordonnée x du menu.
- `int y` : coordonnée y du menu.

Souvenez-vous que vous pouvez déterminer les coordonnées de la souris grâce à l'objet passé en paramètre de la méthode `mouseReleased(MouseEvent event)`.

Je suis sûr que vous savez comment vous y prendre pour indiquer au menu contextuel de s'afficher et qu'il ne vous manque plus qu'à détecter le clic droit. C'est là que l'objet `MouseEver` vous sauver la mise ! En effet, il possède une méthode `isPopupTrigger()` qui renvoie `vrai` s'il s'agit d'un clic droit. Vous avez toutes les cartes en main pour élaborer votre menu cor (rappelez-vous que nous ne gérons pas encore les nouvelles fonctionnalités).

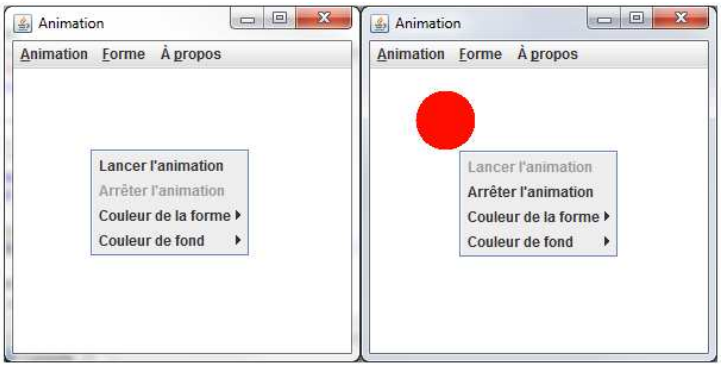
Je vous laisse quelques instants de réflexion…

Vous avez fini ? Nous pouvons comparer nos codes ? Je vous invite à consulter le code ci-dessous (il ne vous montre que les nouveautés).

```
1 //Les imports habituels
2 import javax.swing.JPopupMenu;
3
4 public class Fenetre extends JFrame{
5   //Nos variables habituelles
6
7   //La déclaration pour le menu contextuel
8   private JPopupMenu jpm = new JPopupMenu();
9   private JMenu background = new JMenu("Couleur de fond");
10  private JMenu couleur = new JMenu("Couleur de la forme");
11
12  private JMenuItem launch = new JMenuItem("Lancer l'animation");
13  private JMenuItem stop = new JMenuItem("Arrêter l'animation");
14  private JMenuItem  rouge = new JMenuItem("Rouge"),
15    bleu = new JMenuItem("Bleu"),
16    vert = new JMenuItem("Vert"),
17    rougeBack = new JMenuItem("Rouge"),
18    bleuBack = new JMenuItem("Bleu"),
19    vertBack = new JMenuItem("Vert");
20
21  //On crée des listeners globaux
22  private StopAnimationListener stopAnimation=new StopAnimationListener();
23  private StartAnimationListener startAnimation=new StartAnimationListener();
24
25  public Fenetre(){
26    this.setTitle("Animation");
27    this.setSize(300, 300);
28    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29    this.setLocationRelativeTo(null);
30
31    container.setBackground(Color.white);
32    container.setLayout(new BorderLayout());
33
34    //On initialise le menu stop
35    stop.setEnabled(false);
36    //On affecte les écouteurs
37    stop.addActionListener(stopAnimation);
38    launch.addActionListener(startAnimation);
39
40    //On crée et on passe l'écouteur pour afficher le menu contextuel
41    //Création d'une implémentation de MouseAdapter
42    //avec redéfinition de la méthode adéquate
43    pan.addMouseListener(new MouseAdapter(){
```

```
50          background.add(vertBack);
51
52          couleur.add(rouge);
53          couleur.add(bleu);
54          couleur.add(vert);
55
56          jpm.add(launch);
57          jpm.add(stop);
58          jpm.add(couleur);
59          jpm.add(background);
60          //La méthode qui va afficher le menu
61          jpm.show(pan, event.getX(), event.getY());
62        }
63      }
64    });
65
66    container.add(pan, BorderLayout.CENTER);
67
68    this.setContentPane(container);
69    this.initMenu();
70    this.setVisible(true);
71
72  }
73
74  private void initMenu(){
75    //Ajout du listener pour lancer l'animation
76    //ATTENTION, LE LISTENER EST GLOBAL !!!
77    lancer.addActionListener(startAnimation);
78    //On attribue l'accélerateur c
79    lancer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L, KeyEvent.CTRL_MASK));
80    animation.add(lancer);
81
82    //Ajout du listener pour arrêter l'animation
83    //LISTENER A CHANGER ICI AUSSI
84    arreter.addActionListener(stopAnimation);
85    arreter.setEnabled(false);
86    arreter.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A, KeyEvent.CTRL_DOWN_MASK + KeyEvent.SHIFT_DOWN_MASK));
87    animation.add(arreter);
88
89    //Le reste est inchangé
90  }
91
92  private void go(){
93    //La méthode n'a pas changé
94  }
95
96  public class StartAnimationListener implements ActionListener{
97    public void actionPerformed(ActionEvent arg0) {
98      JOptionPane jop = new JOptionPane();
99      int option = jop.showConfirmDialog(null,
100       "Voulez-vous lancer l'animation ?",
101       "Lancement de l'animation",
102       JOptionPane.YES_NO_OPTION,
103       JOptionPane.QUESTION_MESSAGE);
104
105     if(option == JOptionPane.OK_OPTION){
106       lancer.setEnabled(false);
107       arreter.setEnabled(true);
108
109       //On ajoute l'instruction pour le menu contextuel
110       launch.setEnabled(false);
111       stop.setEnabled(true);
112
113       animated = true;
114       t = new Thread(new PlayAnimation());
115       t.start();
116     }
117   }
118 }
119
120 /**
121 * Écouteur du menu Quitter
122 * @author CHerby
123 */
124 class StopAnimationListener  implements ActionListener{
125
126   public void actionPerformed(ActionEvent e) {
127     JOptionPane jop = new JOptionPane();
128     int option = jop.showConfirmDialog(null,
129       "Voulez-vous arrêter l'animation ?",
130       "Arrêt de l'animation",
131       JOptionPane.YES_NO_CANCEL_OPTION,
132       JOptionPane.QUESTION_MESSAGE);
133
134     if(option != JOptionPane.NO_OPTION && option != JOptionPane.CANCEL_OPTION && option != JOptionPane.CLOSED_OPTION){
135       animated = false;
136       //On remplace nos boutons par nos JMenuItem
137       lancer.setEnabled(true);
138       arreter.setEnabled(false);
139
140       //On ajoute l'instruction pour le menu contextuel
141       launch.setEnabled(true);
142       stop.setEnabled(false);
143     }
144   }
145 }
146
147 class PlayAnimation implements Runnable{
148   //Inchangé
149 }
150
151 class FormeListener implements ActionListener{
152   //Inchangé
153 }
154
155 class MorphListener implements ActionListener{
156   //Inchangé
157 }
158 }
```

La figure suivante vous montre ce que j'obtiens.



Menu contextuel

Il est beau, il est fonctionnel notre menu contextuel !

Je sens que vous êtes prêts pour mettre les nouvelles options en place, même si je me doute que certains d'entre vous ont déjà fait ce qu'il fallait. Allez, il n'est pas très difficile de coder ce de choses (surtout que vous êtes habitués, maintenant). Dans notre classe `Panneau`, nous utilisons des couleurs prédéfinies. Ainsi, il nous suffit de mettre ces couleurs dans des variables permettre leur modification.

Rien de difficile ici, voici donc les codes sources de nos deux classes.

Panneau.java

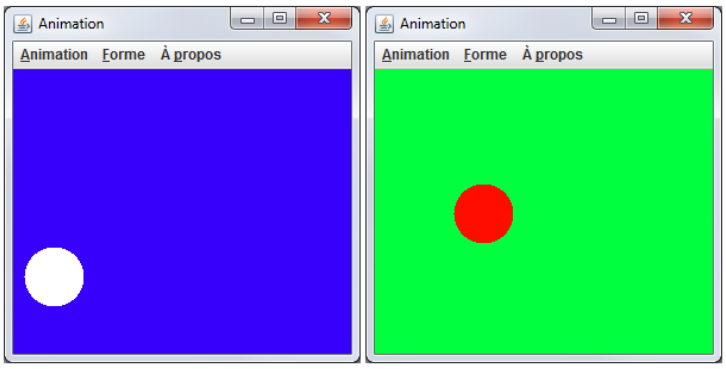
```
1 import java.awt.Color;
2 //Les autres imports
3
4 public class Panneau extends JPanel {
5   //Les variables définies auparavant ne changent pas
6   //On y ajoute nos deux couleurs
7   private Color couleurForme = Color.red;
8   private Color couleurFond = Color.white;
9
10  public void paintComponent(Graphics g){
11    //Affectation de la couleur de fond
12    g.setColor(couleurFond);
13    g.fillRect(0, 0, this.getWidth(), this.getHeight());
14
15    //Affectation de la couleur de la forme
16    g.setColor(couleurForme);
17    //Si le mode morphing est activé, on peint le morphing
18    if(this.morph)
19      drawMorph(g);
20    //Sinon, mode normal
21    else
22      draw(g);
23  }
24
25  //Méthode qui redéfinit la couleur du fond
26  public void setCouleurFond(Color color){
27    this.couleurFond = color;
28  }
29
30  //Méthode qui redéfinit la couleur de la forme
31  public void setCouleurForme(Color color){
32    this.couleurForme = color;
33  }
34
35  //Les autres méthodes sont inchangées
36
37 }
```

Fenetre.java

```
1 //Nos imports habituels
2
3 public class Fenetre extends JFrame{
4   //Nos variables n'ont pas changé
5
6   //On crée des listeners globaux
7   private StopAnimationListener stopAnimation = new StopAnimationListener();
8   private StartAnimationListener startAnimation = new StartAnimationListener()
9   //Avec des listeners pour les couleurs
10  private CouleurFondListener bgColor = new CouleurFondListener();
11  private CouleurFormeListener frmColor = new CouleurFormeListener();
12
13  public Fenetre(){
14    this.setTitle("Animation");
15    this.setSize(300, 300);
16    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17    this.setLocationRelativeTo(null);
18
19    container.setBackground(Color.white);
20    container.setLayout(new BorderLayout());
21
22    //On initialise le menu stop
23    stop.setEnabled(false);
24    //On affecte les écouteurs
25    stop.addActionListener(stopAnimation);
26    launch.addActionListener(startAnimation);
27
28    //On affecte les écouteurs aux points de menu
29    rouge.addActionListener(frmColor);
30    bleu.addActionListener(frmColor);
31    vert.addActionListener(frmColor);
32    blanc.addActionListener(frmColor);
33
```

```
40    //avec redéfinition de la méthode adéquate
41    pan.addMouseListener(new MouseAdapter(){
42      public void mouseReleased(MouseEvent event){
43        //Seulement s'il s'agit d'un clic droit
44        if(event.isPopupTrigger()){
45          background.add(blancBack);
46          background.add(rougeBack);
47          background.add(bleuBack);
48          background.add(vertBack);
49
50          couleur.add(blanc);
51          couleur.add(rouge);
52          couleur.add(bleu);
53          couleur.add(vert);
54
55          jpm.add(launch);
56          jpm.add(stop);
57          jpm.add(couleur);
58          jpm.add(background);
59
60          //La méthode qui va afficher le menu
61          jpm.show(pan, event.getX(), event.getY());
62        }
63      }
64    });
65
66    container.add(pan, BorderLayout.CENTER);
67    this.setContentPane(container);
68    this.initMenu();
69    this.setVisible(true);
70  }
71
72  private void initMenu(){
73    //Le menu n'a pas changé
74  }
75
76  private void go(){
77    //La méthode go() est identique
78  }
79
80  //Les classes internes :
81  // -> StartAnimationListener
82  // -> StopAnimationListener
83  // -> PlayAnimation
84  // -> FormeListener
85  // -> MorphListener
86  //sont inchangées !
87
88  //Écoute le changement de couleur du fond
89  class CouleurFondListener implements ActionListener{
90    public void actionPerformed(ActionEvent e) {
91
92      if(e.getSource() == vertBack)
93        pan.setCouleurFond(Color.green);
94      else if (e.getSource() == bleuBack)
95        pan.setCouleurFond(Color.blue);
96      else if(e.getSource() == rougeBack)
97        pan.setCouleurFond(Color.red);
98      else
99        pan.setCouleurFond(Color.white);
100   }
101 }
102
103 //Écoute le changement de couleur du fond
104 class CouleurFormeListener implements ActionListener{
105   public void actionPerformed(ActionEvent e) {
106     if(e.getSource() == vert)
107       pan.setCouleurForme(Color.green);
108     else if (e.getSource() == bleu)
109       pan.setCouleurForme(Color.blue);
110     else if(e.getSource() == rouge)
111       pan.setCouleurForme(Color.red);
112     else
113       pan.setCouleurForme(Color.white);
114   }
115 }
116 }
```

la figure suivante représente deux résultats ainsi obtenus.



Changement de couleur via le menu contextuel

Vous conviendrez que les menus et les menus contextuels peuvent s'avérer vraiment utiles et ergonomiques ! En plus, ils sont relativement simples à implémenter (et à utiliser). Cependant avez sans doute remarqué qu'il y a beaucoup de clics superflus, que ce soit pour utiliser un menu ou menu contextuel : il faut au moins un clic pour afficher leur contenu (sauf dans le cas d l'accélérateur).

Exemple de barre d'outils

Pour faire simple, la barre d'outils sert à effectuer des actions disponibles dans le menu, mais sans devoir fouiller dans celui-ci ou mémoriser le raccourci clavier (accélérateur) qui y est lié. permet donc des actions rapides.

Elle est généralement composée d'une multitude de boutons, une image apposée sur chacun d'entre eux symbolisant l'opération qu'il peut effectuer.

Pour créer et utiliser une barre d'outils, nous allons utiliser l'objet `JToolBar`. Je vous rassure tout de suite, cet objet fonctionne comme un menu classique, à une différence près : celui-ci des boutons (`JButton`) en arguments, et il n'y a pas d'endroit spécifique où incorporer votre barre d'outils (il faudra l'expliciter lors de sa création).

Tout d'abord, il nous faut des images à mettre sur nos boutons… J'en ai fait de toutes simples (figure suivante), mais libre à vous d'en choisir d'autres.

Images pour la barre d'outils

Au niveau des actions à gérer, pour le lancement de l'animation et l'arrêt, il faudra penser à éditer le comportement des boutons de la barre d'outils comme on l'a fait pour les deux actions menu contextuel. Concernant les boutons pour les formes, c'est un peu plus délicat. Les autres composants qui éditaient la forme de notre animation étaient des boutons radios. Or, ici, nc avons des boutons standard. Outre le fait qu'il va falloir une instance précise de la classe `FormeListener`, nous aurons à modifier un peu son comportement…

Il nous faut savoir si l'action vient d'un bouton radio du menu ou d'un bouton de la barre d'outils : c'est l'objet `ActionEvent` qui nous permettra d'accéder à cette information. Nous n'a pas tester tous les boutons radio un par un, pour ces composants, le système utilisé jusque-là était très bien. Non, nous allons simplement vérifier si celui qui a déclenché l'action est un `JRadioButtonMenuItem`, et si c'est le cas, nous testerons les boutons.

Rappelez-vous le chapitre sur la réflexivité ! La méthode `getSource()` nous retourne un objet, il est donc possible de connaître la classe de celui-ci avec la méthode `getClass()` et p conséquent d'en obtenir le nom grâce à la méthode `getName()`.

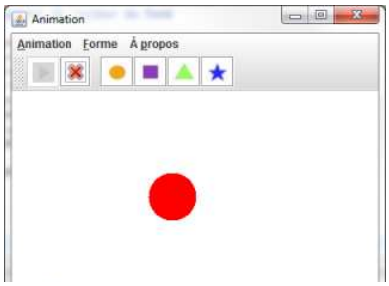
Il va falloir qu'on pense à mettre à jour le bouton radio sélectionné dans le menu. Et là, pour votre plus grand bonheur, je connais une astuce qui marche pas mal du tout : lors du clic sur ur bouton de la barre d'outils, il suffit de déclencher l'événement sur le bouton radio correspondant ! Dans la classe `AbstractButton`, dont héritent tous les boutons, il y a la méthode `doClick()`. Cette méthode déclenche un événement identique à un vrai clic de souris sur le composant ! Ainsi, plutôt que de gérer la même façon de faire à deux endroits, nous allons re l'action effectuée sur un composant vers un autre.

Vous avez toutes les cartes en main pour réaliser votre barre d'outils. N'oubliez pas que vous devez spécifier sa position sur le conteneur principal ! Bon. Faites des tests, comparez, codez, effacez… au final, vous devriez avoir quelque chose comme ceci :

```
1 import javax.swing.JToolBar;
2 //Nos imports habituels
3
4 public class Fenetre extends JFrame{
5   //Les variables declarées précédemment
6
7   //Création de notre barre d'outils
8   private JToolBar toolBar = new JToolBar();
9
10  //Les boutons de la barre d'outils
11  private JButton   play = new JButton(new ImageIcon("images/play.jpg")),
12    cancel = new JButton(new ImageIcon("images/stop.jpg")),
13    square = new JButton(new ImageIcon("images/carré.jpg")),
14    tri = new JButton(new ImageIcon("images/triangle.jpg")),
15    circle = new JButton(new ImageIcon("images/rond.jpg")),
16    star = new JButton(new ImageIcon("images/étoile.jpg"));
17
18  private Color fondBouton = Color.white;
19  private FormeListener fListener = new FormeListener();
20
21  public Fenetre(){
22    //La seule nouveauté est la méthode ci-dessous
23    this.initToolBar();
24    this.setVisible(true);
25  }
26
27  private void initToolBar(){
28    this.cancel.setEnabled(false);
29    this.cancel.addActionListener(stopAnimation);
30    this.cancel.setBackground(fondBouton);
31    this.play.addActionListener(startAnimation);
32    this.play.setBackground(fondBouton);
33
34    this.toolBar.add(play);
35    this.toolBar.add(cancel);
36    this.toolBar.addSeparator();
37
38    //Ajout des Listeners
39    this.circle.addActionListener(fListener);
40    this.circle.setBackground(fondBouton);
41    this.toolBar.add(circle);
42
43    this.square.addActionListener(fListener);
44    this.square.setBackground(fondBouton);
45    this.toolBar.add(square);
46
47    this.tri.setBackground(fondBouton);
```

```
54
55    this.add(toolBar, BorderLayout.NORTH);
56  }
57
58  private void initMenu(){
59    //Méthode inchangée
60  }
61
62  private void go(){
63    //Méthode inchangée
64  }
65
66  public class StartAnimationListener implements ActionListener{
67    public void actionPerformed(ActionEvent arg0) {
68      //Toujours la même boîte de dialogue…
69
70      if(option == JOptionPane.OK_OPTION){
71        lancer.setEnabled(false);
72        arreter.setEnabled(true);
73
74        //ON AJOUTE L'INSTRUCTION POUR LE MENU CONTEXTUEL
75        //************************************************
76        launch.setEnabled(false);
77        stop.setEnabled(true);
78
79        play.setEnabled(false);
80        cancel.setEnabled(true);
81
82        animated = true;
83        t = new Thread(new PlayAnimation());
84        t.start();
85      }
86    }
87  }
88
89  /**
90   * Écouteur du menu Quitter
91   * @author CHerby
92   */
93  class StopAnimationListener  implements ActionListener{
94
95    public void actionPerformed(ActionEvent e) {
96      //Toujours la même boîte de dialogue…
97
98      if(option != JOptionPane.NO_OPTION && option != JOptionPane.CANCEL_OPTION && option != JOptionPane.CLOSED_OPTION){
99        animated = false;
100       //On remplace nos boutons par nos MenuItem
101       lancer.setEnabled(true);
102       arreter.setEnabled(false);
103
104       //ON AJOUTE L'INSTRUCTION POUR LE MENU CONTEXTUEL
105       //************************************************
106       launch.setEnabled(true);
107       stop.setEnabled(false);
108
109       play.setEnabled(true);
110       cancel.setEnabled(false);
111     }
112   }
113 }
114
115 class FormeListener implements ActionListener{
116   public void actionPerformed(ActionEvent e) {
117
118     //Si l'action vient d'un bouton radio du menu
119     if(e.getSource().getClass().getName().equals("javax.swing.JRadioButtonMenuItem"))
120       pan.setForme(((JRadioButtonMenuItem)e.getSource()).getText());
121     else{
122       if(e.getSource() == square){
123         carre.doClick();
124       }
125       else if(e.getSource() == tri){
126         triangle.doClick();
127       }
128       else if(e.getSource() == star){
129         etoile.doClick();
130       }
131       else{
132         rond.doClick();
133       }
134     }
135   }
136 }
137
138 //Les classes internes :
139 // -> CouleurFondListener
140 // -> CouleurFormeListener
141 // -> PlayAnimation
142 // -> MorphListener
143 //sont inchangées !
144
145 }
```

Vous devez obtenir une IHM semblable à la figure suivante.

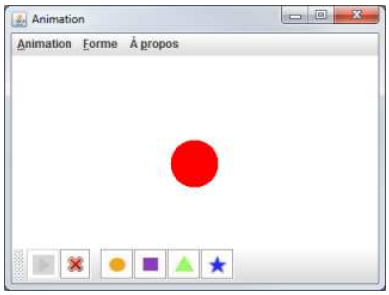


oui, vous avez appris à penser en orienté objet et connaissez les points principaux de la programmation événementielle. Maintenant, il vous reste simplement à acquérir des détails techni spécifiques (par exemple, la manière d'utiliser certains objets).

Pour ceux qui l'auraient remarqué, la barre d'outils est déplaçable ! Si vous cliquez sur la zone mise en évidence à la figure suivante, vous pourrez la repositionner.

Zone de déplacement

Il suffit de maintenir le clic et de faire glisser votre souris vers la droite, la gauche ou encore le bas. Vous verrez alors un carré se déplacer et, lorsque vous relâcherez le bouton, votre barre a changé de place, comme le montre la figure suivante.



Déplacement de la barre d'outils

Elles sont fortes ces barres d'outils, tout de même ! En plus de tout ça, vous pouvez utiliser autre chose qu'un composant sur une barre d'outils...

Utiliser les actions abstraites

Nous avons vu précédemment comment centraliser des actions sur différents composants. Il existe une classe abstraite qui permet de gérer ce genre de choses, car elle peut s'adapter à beaucoup de composants (en général à ceux qui ne font qu'une action, comme un bouton, une case à cocher, mais pas une liste).

Le rôle de cette classe est d'attribuer automatiquement une action à un ou plusieurs composants. Le principal avantage de ce procédé est que plusieurs composants travaillent avec une implémentation de la classe `AbstractAction`, mais son gros inconvénient réside dans le fait que vous devrez programmer une implémentation par action :

- une action pour la couleur de la forme en rouge ;
- une action pour la couleur de la forme en bleu ;
- une action pour la couleur de la forme en vert ;
- une action pour la couleur du fond en rouge ;
- une action pour la couleur du fond en bleu ;
- etc.

Cela peut être très lourd à faire, mais je laisse votre bon sens déterminer s'il est pertinent d'utiliser cette méthode ou non !

Voici comment s'implémente cette classe :

```
1 public class Fenetre extends JFrame{
2   //Nous pouvons utiliser les actions abstraites directement dans un JButton
3   private JButton bouton1 = new JButton(new RougeAction("ActionRouge", new ImageIcon("images/rouge.jpg"));
4
5   //Ou créer une instance concrète
6   private RougeAction rAct = new RougeAction("ActionRouge", new ImageIcon("images/rouge.jpg"));
7   private JToolBar toolBar = new JToolBar();
8
9   //…
10
11  //Utiliser une action directement dans une barre d'outils
12  private void initToolBar(){
13    toolBar.add(rAct);
14  }
15
16  //…
17
18  class RougeAction extends AbstractAction{
19    //Constructeur avec le nom uniquement
20    public RougeAction(String name){super(name);}
21
22    //Le constructeur prend le nom et une icône en paramètre
23    public RougeAction(String name, ImageIcon){super(name, img);}
24
25    public void actionPerformed(ActionEvent){
26      //Vous connaissez la marche à suivre
27    }
28  }
29 }
```

Vous pouvez voir que cela peut être très pratique. Désormais, si vous ajoutez une action sur une barre d'outils, celle-ci crée automatiquement un bouton correspondant ! Utiliser les action abstraites plutôt que des implémentations de telle ou telle interface est un choix qui vous revient. Nous pouvons d'ailleurs très bien appliquer ce principe au code de notre animation, mai constaterez qu'il s'alourdira, nous éviterons donc de le faire… Mais comme je vous le disais, c'est une question de choix et de conception.

- Les boîtes de dialogue s'utilisent, à l'exception des boîtes personnalisées, avec l'objet `JOptionPane`.

- La méthode citée ci-dessus retourne un entier correspondant au bouton sur lequel vous avez cliqué.
- La méthode `showInputDialog()` affiche une boîte attendant une saisie clavier ou une sélection dans une liste.
- Cette méthode retourne soit un `String` dans le cas d'une saisie, soit un `Object` dans le cas d'une liste.
- La méthode `showOptionDialog()` affiche une boîte attendant que l'utilisateur effectue un clic sur une option.
- Celle-ci retourne l'indice de l'élément sur lequel vous avez cliqué ou un indice négatif dans tous les autres cas.
- Les boîtes de dialogue sont dites « modales » : aucune interaction hors de la boîte n'est possible tant que celle-ci n'est pas fermée !
- Pour faire une boîte de dialogue personnalisée, vous devez créer une classe héritée de `JDialog`.
- Pour les boîtes personnalisées, le dialogue commence lorsque `setVisible(true)` est invoquée et se termine lorsque la méthode `setVisible(false)` est appelée.
- L'objet servant à insérer une barre de menus sur vos IHM `swing` est un `JMenuBar`.
- Dans cet objet, vous pouvez mettre des objets `JMenu` afin de créer un menu déroulant.
- L'objet cité ci-dessus accepte des objets `JMenu, JMenuItem, JCheckBoxMenuItem` et `JRadioButtonMenuItem`.
- Afin d'interagir avec vos points de menu, vous pouvez utiliser une implémentation de l'interface `ActionListener`.
- Pour faciliter l'accès aux menus de la barre de menus, vous pouvez ajouter des *mnémoniques* à ceux-ci.
- L'ajout d'accélérateurs permet de déclencher des actions, le plus souvent par des combinaisons de touches.
- Afin de récupérer les codes des touches du clavier, vous devrez utiliser un objet `KeyStroke` ainsi qu'un objet `KeyEvent`.
- Un menu contextuel fonctionne comme un menu normal, à la différence qu'il s'agit d'un objet `JPopupMenu`. Vous devez toutefois spécifier le composant sur lequel doit s'afficher l contextuel.
- La détection du clic droit se fait grâce à la méthode `isPopupTrigger()` de l'objet `MouseEvent`.
- L'ajout d'une barre d'outils nécessite l'utilisation de l'objet `JToolBar`.

Les champs de formulaire

TP : l'ardoise magique