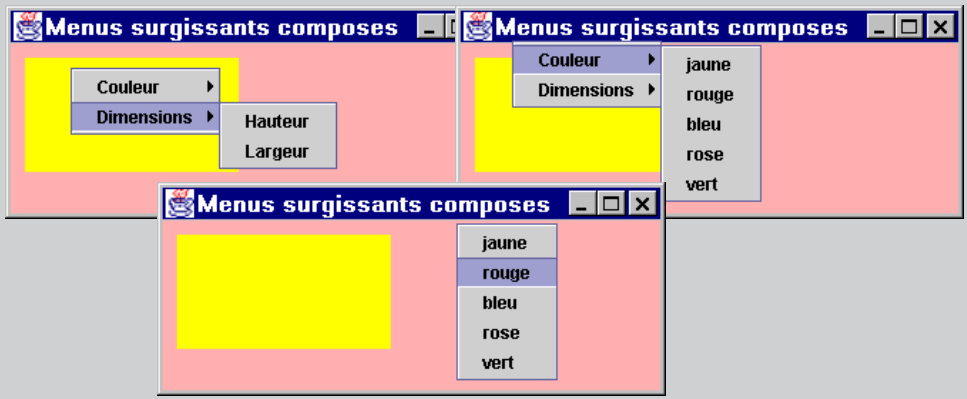
133 choix de couleurs et de dimensions par des menus surgissants

Afficher un rectangle coloré dans une fenêtre. Un clic dans le rectangle fera apparaître un menu surgissant permettant de modifier les dimensions du rectangle ou sa couleur. Un clic en dehors du rectangle fera apparaître un menu surgissant permettant de modifier la couleur du fond.



Les couleurs et leurs noms seront les mêmes pour le fond et pour le rectangle et ils seront fournis sous forme de tableaux en arguments du constructeur de la fenêtre.

Le dessin d'un rectangle de couleur donnée se fait en appliquant au contexte graphique concerné successivement la méthode *setColor* (en argument l'objet de type *Color* voulu) et la méthode *fillRect* (*int abscisse, int ordonnee, int largeur, int hauteur)*).

Note : la résolution de cet exercice sera facilitée par celle de l'exercice .

Solution

Nous dessinons dans un panneau dont nous redéfinissons classiquement la méthode *paintComponent*. Cela nécessite la création d'une classe spécialisée *Panneau* dérivée de *JPanel*.

Dans le constructeur de la fenêtre, nous créons deux menus surgissants *menuForme* et *menuFond*. Le premier est constitué de deux sous-menus (de type *JMenu*) *menuFormeDimensions* et *menuFormeCouleurs*. Leurs options sont de type *JMenuItem*. Nous avons choisi d'écouter les différentes options dans la fenêtre elle-même (les écouter dans le panneau aurait nécessité de lui fournir les références de toutes les options concernées).

Les variables *couleurFond*, *couleurForme*, *l* et *h* servent à mémoriser dans la fenêtre les dernières couleurs sélectionnées et les dimensions du rectangle. La méthode *paintComponent* du panneau y accède à l'aide des méthodes *getCouleurFond*, *getCouleurForme*, *getLargeur* et *getHauteur*.

Les dimensions sont lues dans des boîtes de saisie. On traite comme à l'accoutumée les exceptions de conversion.

Le déclenchement des menus surgissants a lieu en cas de clic dans le panneau dont nous faisons son propre écouteur d'événements *Mouse*. Ici, l'affichage du menu est réalisé lors du relâchement du bouton attribué aux menus surgissants : nous redéfinissons *mouseReleased* et nous testons le bouton concerné à l'aide de la méthode *isPopupTrigger* de la classe *MouseEvent*. L'objet panneau n'a besoin de connaître que la référence de la fenêtre et celles des deux menus surgissants. Celles-ci sont transmises au constructeur.

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

class FenRect extends JFrame implements ActionListener
{ public FenRect (Color [] couleurs, String [] nomsCouleurs)
  { setTitle ("Menus surgissants composes") ;
    setSize (300, 150) ;
    this.couleurs = couleurs ;
    this.nomsCouleurs = nomsCouleurs ;

     /* creation menus surgissants Fond et Forme */
    menuFond = new JPopupMenu () ;
    menuForme = new JPopupMenu () ;
    menuFormeCouleur = new JMenu ("Couleur") ;
    menuForme.add (menuFormeCouleur) ;
    menuFormeDimensions = new JMenu ("Dimensions") ;
    menuForme.add (menuFormeDimensions) ;
     /* creation des options */
    nbCouleurs = couleurs.length ;
    optionsCouleurFond =  new JMenuItem [nbCouleurs] ;
    optionsCouleurForme = new JMenuItem [nbCouleurs] ;
    for (int i=0 ; i<nbCouleurs ; i++)
     { optionsCouleurForme[i] = new JMenuItem (nomsCouleurs[i]) ;
       optionsCouleurForme[i].addActionListener (this) ;
       menuFormeCouleur.add (optionsCouleurForme[i]) ;
       optionsCouleurFond[i] = new JMenuItem (nomsCouleurs[i]) ;
       optionsCouleurFond[i].addActionListener (this) ;
       menuFond.add  (optionsCouleurFond[i]) ;
     }
    optionHauteur = new JMenuItem ("Hauteur") ;
    optionLargeur = new JMenuItem ("Largeur") ;
    menuFormeDimensions.add (optionHauteur) ;
    menuFormeDimensions.add (optionLargeur) ;
```

```
    optionHauteur.addActionListener (this) ;
    optionLargeur.addActionListener (this) ;
     /* creation panneau de dessin */
    panneau = new Panneau (this, menuForme, menuFond) ;
    panneau.addMouseListener (panneau) ;
    getContentPane().add (panneau) ;
  }
  public void actionPerformed (ActionEvent e)
  { Object source = e.getSource() ;
    for (int i=0 ; i<nbCouleurs ; i++)
     { if (source == optionsCouleurFond[i])  couleurFond = couleurs[i] ;
       if (source == optionsCouleurForme[i]) couleurForme = couleurs[i] ;
     }
     if ((source == optionLargeur) || (source == optionHauteur))
     { int valeur=0 ;  String question ;
       boolean ok=false ;
       if (source == optionLargeur) question = "Nouvelle largeur ?" ;
                                else  question = "Nouvelle hauteur ?" ;
       String rep = JOptionPane.showInputDialog (null, question) ;
       try
       { valeur = Integer.parseInt (rep) ;
         ok = true ;
       }
       catch (NumberFormatException ex) { }
       if (ok) if (source == optionLargeur)  l = valeur ;
                                        else h = valeur ;
     }
     panneau.repaint() ;  // pour forcer a repeindre l'ensemble de la fenetre
  }
  public Color getCouleurFond ()      { return couleurFond ;     }
  public Color getCouleurForme ()     { return couleurForme ;    }
  public int getLargeur ()            { return l ; }
  public int getHauteur ()            { return h ; }

  private Color[] couleurs ;
  private String[] nomsCouleurs ;
  private Panneau panneau ;
  private JPopupMenu menuFond, menuForme ;
  private JMenu menuFormeCouleur, menuFormeDimensions ;
  private JMenuItem[] optionsCouleurFond, optionsCouleurForme ;
  private JMenuItem optionHauteur, optionLargeur ;
  private int nbCouleurs ;
  private Color couleurFond=Color.white, couleurForme=Color.black ;
  private int l=100, h=50 ;
}
class Panneau extends JPanel implements MouseListener
{ private static int x=10, y=10 ;
  public Panneau (FenRect fen, JPopupMenu menuForme, JPopupMenu menuFond)
  { this.fen = fen ;
    this.menuForme = menuForme ;
    this.menuFond  = menuFond ;
  }
```

```
  public void mouseReleased (MouseEvent e)
  { if(!e.isPopupTrigger ()) return ;
    int xClic = e.getX(), yClic = e.getY() ;
    if ((xClic>=x) && (xClic<=x+largeur) && (yClic>=y) && (yClic<=y+hauteur) )
       menuForme.show (fen, xClic, yClic) ;
    else
      menuFond.show (fen, xClic, yClic) ;
  }

  public void mousePressed (MouseEvent e) {}
  public void mouseClicked (MouseEvent e) {}
  public void mouseEntered (MouseEvent e) {}
  public void mouseExited  (MouseEvent e) {}

  public void paintComponent (Graphics g)
  { super.paintComponent(g) ;
    setBackground (fen.getCouleurFond()) ;
    g.setColor (fen.getCouleurForme()) ;
    largeur = fen.getLargeur() ;
    hauteur = fen.getHauteur() ;
    g.fillRect (x, y, largeur, hauteur) ;
  }
  private FenRect fen ;
  private int largeur, hauteur ;
  private JPopupMenu menuForme, menuFond ;
}

public class Compsurg
{ private static Color [] couleurs =
         {Color.yellow, Color.red, Color.blue, Color.pink, Color.green } ;
  private static String[] nomsCouleurs =
         {"jaune",      "rouge",   "bleu",     "rose",     "vert"       } ;
  public static void main (String args[])
  { FenRect fen = new FenRect(couleurs, nomsCouleurs) ;
    fen.setVisible(true) ;
  }
}
```

Chapitre 13

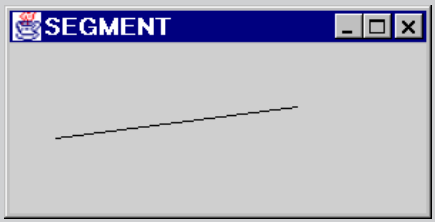
Les événements de bas niveau

Connaissances requises

- Événements de type *MouseEvent* liés aux boutons de la souris (rappel) ; méthodes *mousePressed*, *mouseReleased* et *mouseClicked*
- Identification du bouton de la souris ; méthodes *getModifiers* et constantes correspondantes *InputEvent.BUTTON1_MASK*, *InputEvent.BUTTON2_MASK* et *InputEvent.BUTTON3_MASK*
- Gestion des clics multiples ; méthode *getClickCount*
- Gestion des déplacements de la souris ; méthodes *mouseEntered*, *mouseExited*, *mouseMoved* et *mouseDragged*
- Événements de type *KeyEvent* ; méthodes *keyPressed*, *keyReleased* et *keyTyped* ; identification d'une touche par son code de touche virtuelle (méthode *getKeyCode*) ou par le caractère correspondant (méthode *getKeyChar*) ; connaissance de l'état des touches modificatrices (méthodes *isXXXDown* et *getModifiers*) ; source d'un événement clavier

134 Identification des boutons de la souris

Afficher en permanence un segment dans une fenêtre. Son origine sera définie par un clic sur le bouton de gauche de la souris et elle se modifiera à chaque nouveau clic sur ce même bouton. Son extrémité sera définie de la même manière avec le bouton de droite :



Solution

Pour obtenir la permanence du dessin, nous tracerons notre segment dans un panneau. Ici, il est plus simple d'écouter les clics (*mouseClicked*) dans le panneau lui-même. Pour identifier le bouton de la souris, nous utilisons la méthode *getModifiers* de la classe *MouseEvent*. Elle fournit un entier dans lequel un bit de rang donné, associé à chacun des boutons, prend la valeur 1. La classe *InputEvent* contient des constantes qu'on peut utiliser comme masque pour faciliter les choses ; ici, ce sont les constantes *BUTTON1_MASK* (bouton de gauche) et *BUTTON3_MASK* (bouton de droite) qui nous intéressent.

Le segment est défini par les coordonnées de son origine (*xOr* et *yOr*) et celles de son extrémité (*xExt* et *yExt*). Deux indicateurs booléens *orConnue* et *extConnue* permettent de savoir si ces informations sont disponibles (elles sont en fait placées à *false* au début du programme). Le dessin proprement dit est réalisé dans la méthode *paintComponent* qui exploite ces différentes informations. Notez qu'il est nécessaire d'appeler *repaint* après un clic gauche ou droite, afin de provoquer l'appel de *paintComponent*.

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;
class MaFenetre extends JFrame
{ public MaFenetre ()
  { setTitle ("SEGMENT") ;
    setSize (300, 150) ;
    pan =  new Panneau () ;
    getContentPane().add (pan) ;
    pan.addMouseListener (pan) ;
  }
  private Panneau pan ;
}
```

```
class Panneau extends JPanel implements MouseListener
{ public void paintComponent (Graphics g)
  { super.paintComponent (g) ;
    if (orConnue && extConnue) g.drawLine (xOr, yOr, xExt, yExt) ;
  }
  public void mousePressed (MouseEvent e)
  { int x=e.getX(), y=e.getY() ;
    int modifieurs = e.getModifiers() ;
    if ( (modifieurs & InputEvent.BUTTON1_MASK) != 0)
    { /* clic bouton gauche */
      xOr = x ; yOr = y ;
      orConnue = true ;
      repaint() ;
    }
    if ( (modifieurs & InputEvent.BUTTON3_MASK) != 0)
    { /* clic bouton droite */
      xExt = x ; yExt = y ;
      extConnue = true ;
      repaint() ;
    }
  }
  public void mouseReleased (MouseEvent e) {}
  public void mouseClicked (MouseEvent e)  {}
  public void mouseEntered (MouseEvent e)  {}
  public void mouseExited (MouseEvent e)   {}
  private int xOr, yOr, xExt, yExt ;
  private boolean orConnue=false, extConnue=false ;
}
public class Segments
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible (true) ;
  }
}
```

135 Vrais doubles clics

Java ne dispose que d'un seul compteur de clics pour les différents boutons de la souris. Dans ces conditions, il n'est pas possible de distinguer un véritable double clic de deux clics successifs sur deux boutons différents. Écrire un programme qui détecte les "vrais" doubles clics sur le bouton de gauche et qui affiche alors un message en fenêtre console.

Solution

Nous ferons naturellement de la fenêtre son propre écouteur d'événements souris. Nous utiliserons :

- la méthode *getClickCount* qui fournit le nombre de clics (rapprochés) successifs,
- la méthode *getModifiers* pour identifier le bouton de la souris.

Un indicateur booléen *clicGauche* indique si le dernier clic concernait le bouton de gauche.

Il faut bien prendre garde à :

- mettre l'indicateur *clicGauche* à *false* après un double clic gauche (vrai ou faux) ainsi qu'après tout clic sur un autre bouton,
- mettre l'indicateur *clicGauche* à *true* après un simple clic gauche.

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

class MaFenetre extends JFrame implements MouseListener
{ public MaFenetre ()
  { setTitle ("DOUBLES CLICS") ;
    setSize (300, 150) ;
    clicGauche = false ;
    addMouseListener (this) ;
  }

  public void mousePressed (MouseEvent e)  {}
  public void mouseReleased (MouseEvent e) {}

  public void mouseClicked (MouseEvent e)
  { int modifieurs = e.getModifiers () ;
    if ((modifieurs & InputEvent.BUTTON1_MASK) != 0)
     /* ici, on a affaire a un clic gauche */
    { if ((e.getClickCount() == 2) && clicGauche)
       { System.out.println ("Double clic gauche") ;
         clicGauche = false ;
       }
      else clicGauche = true ;
    }
    else clicGauche = false ;
  }
  public void mouseEntered (MouseEvent e) {}
  public void mouseExited (MouseEvent e ) {}
  private boolean clicGauche ;
}

public class DoubClic
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible (true) ;
  }
}
```

136 Suivi des déplacements de la souris (1)

Créer une fenêtre dotée d'un bouton. Afficher en fenêtre console des messages de suivi des déplacements de la souris comme dans cet exemple :

```
la souris entre dans la fenetre
la souris quitte la fenetre
la souris entre dans le bouton
la souris quitte le bouton
la souris entre dans la fenetre
la souris quitte la fenetre
la souris entre dans la fenetre
la souris quitte la fenetre
la souris entre dans le bouton
la souris quitte le bouton
la souris entre dans la fenetre
la souris quitte la fenetre
```

Solution 1

Il nous suffit de suivre les événements *mouseEntered* et *mouseExited* ayant pour source le bouton ou la fenêtre.

Ici, nous utilisons pour les deux un même écouteur, à savoir la fenêtre elle-même. Nous y redéfinissons les six méthodes prévues par l'interface *MouseListener* ; ici ce sont *MouseEntered* et *MouseExited* qui nous intéressent. Dans chacune de ces deux méthodes, *getSource* nous permet d'identifier la source de l'événement.

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

class MaFenetre extends JFrame implements MouseListener
{ public MaFenetre ()
  { setTitle ("Evenements souris") ;
    setSize (300, 150) ;
    contenu = getContentPane() ;
    contenu.setLayout (new FlowLayout()) ;
    addMouseListener (this) ;
    bouton = new JButton ("A") ;
    contenu.add (bouton) ;
    bouton.addMouseListener (this) ;
  }

  public void mousePressed (MouseEvent e) {}
  public void mouseReleased (MouseEvent e) {}
```

```
  public void mouseClicked (MouseEvent e) {}

  public void mouseEntered (MouseEvent e)
  { if (e.getSource() == this)
      System.out.println ("la souris entre dans la fenetre") ;
    if (e.getSource() == bouton)
      System.out.println ("la souris entre dans le bouton") ;
  }
  public void mouseExited (MouseEvent e)
  { if (e.getSource() == this)
      System.out.println ("la souris quitte la fenetre") ;
    if (e.getSource() == bouton)
      System.out.println ("la souris quitte le bouton") ;
  }
  private JButton bouton ;
  private Container contenu ;
}
public class DpSour
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible (true) ;
  }
```

Remarque L'exemple d'exécution de l'énoncé montre bien que lorsque la souris entre dans le bouton, elle sort de la fenêtre.

Solution 2 Voici une autre solution dans laquelle la fenêtre et le bouton ont été chacun doté d'un écouteur objet d'une classe anonyme (implémentant l'interface *MouseAdapter*). Ici, il n'est plus nécessaire de tester la source d'un événement.

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;
class MaFenetre extends JFrame
{ public MaFenetre ()
  { setTitle ("Evenements souris") ;
    setSize (300, 150) ;
    contenu = getContentPane() ;
    contenu.setLayout (new FlowLayout()) ;
    addMouseListener (new MouseAdapter()
        { public void mouseEntered (MouseEvent e)
          { System.out.println ("la souris entre dans la fenetre") ;
          }
          public void mouseExited (MouseEvent e)
          { System.out.println ("la souris quitte la fenetre") ;
          }
        }) ;
    bouton = new JButton ("A") ;
    contenu.add (bouton) ;
```

```
    bouton.addMouseListener (new MouseAdapter()
        { public void mouseEntered (MouseEvent e)
          { System.out.println ("la souris entre dans le bouton") ;
          }
          public void mouseExited (MouseEvent e)
          { System.out.println ("la souris quitte le bouton") ;
          }
        }) ;
  }

  private JButton bouton ;
  private Container contenu ;
}

public class DpSourb
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible (true) ;
  }
}
```

Solution 3 Voici une troisième solution dans laquelle la fenêtre et le bouton partagent le même écouteur, là encore objet d'une classe anonyme implémentant l'interface *MouseAdapter*.

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

class MaFenetre extends JFrame
{ public MaFenetre ()
  { setTitle ("Evenements souris") ;
    setSize (300, 150) ;
    contenu = getContentPane() ;
    contenu.setLayout (new FlowLayout()) ;
    MouseAdapter ecout = new MouseAdapter()
        { public void mouseEntered (MouseEvent e)
          { if (e.getSource() == contenu)
              System.out.println ("la souris entre dans la fenetre") ;
            if (e.getSource() == bouton)
              System.out.println ("la souris entre dans le bouton") ;
          }
          public void mouseExited (MouseEvent e)
          { if (e.getSource() == contenu)
              System.out.println ("la souris quitte la fenetre") ;
            if (e.getSource() == bouton)
              System.out.println ("la souris quitte le bouton") ;
          }
        } ;

    contenu.addMouseListener (ecout) ;
    bouton = new JButton ("A") ;
```

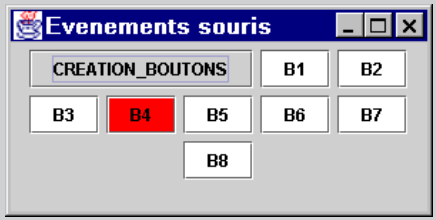
```
      contenu.add (bouton) ;
      bouton.addMouseListener (ecout) ;
    }
    private JButton bouton ;
    private Container contenu ;
}

public class DpSoura
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible (true) ;
  }
}
```

Ici, par souci de simplicité, nous avons intercepté les événements ayant pour source non plus la fenêtre elle-même, mais son contenu. En effet, dans la classe anonyme de l'écouteur, on ne peut plus identifier la fenêtre par *this*. On pourrait le faire en conservant la référence de la fenêtre dans un champ.

137 Suivi des déplacements de la souris (2)

Réaliser une fenêtre disposant d'un bouton marqué *CREATION_BOUTONS* permettant de créer dynamiquement des boutons marqués *B1*, *B2*, *B3*...



Lorsque la souris "passe" sur l'un de ces boutons, il se colore en fonction de son numéro (par exemple, le premier en rouge, le second en jaune, le troisième en vert, le quatrième en bleu, le cinquième à nouveau en rouge...) ; le bouton se colore en blanc lorsque la souris en sort

Solution

Ici, nous faisons de la fenêtre l'unique écouteur des différentes événements :

- *Action* pour le bouton de création,
- *Mouse* pour les boutons dynamiques.