# *Les principaux événements*

# *Quelques notions*

- Les différents concepts
  - Classes "événement" (ex. MouseEvent)
  - Interfaces d'écouteurs (ex MouseListener, MouseMotionListener)
  - Méthodes de traitement (ex. mouseClicked, mouseEntered)
  - Classes d'écouteurs (ex. MouseAdapter)
- Evénement
  - Un objet source génère un événement en appelant automatiquement la méthode de traitement de l'événement en question pour tous les écouteurs enregistrés auprès de l'objet source

# Structuration

- Evénement ↔ méthode de traitement

  Exemples : clic souris, arrivée de la souris sur un composant, ...

- Pour tous les événements nécessitant des infos communes
  - ➔ classe regroupant ces infos (ex. MouseEvent)
  - ➔ paramètre de ce type des méthodes de traitement de l'événement
- Regroupement des événements similaires dans une seule interface (en général...)

  Ex : MouseListener

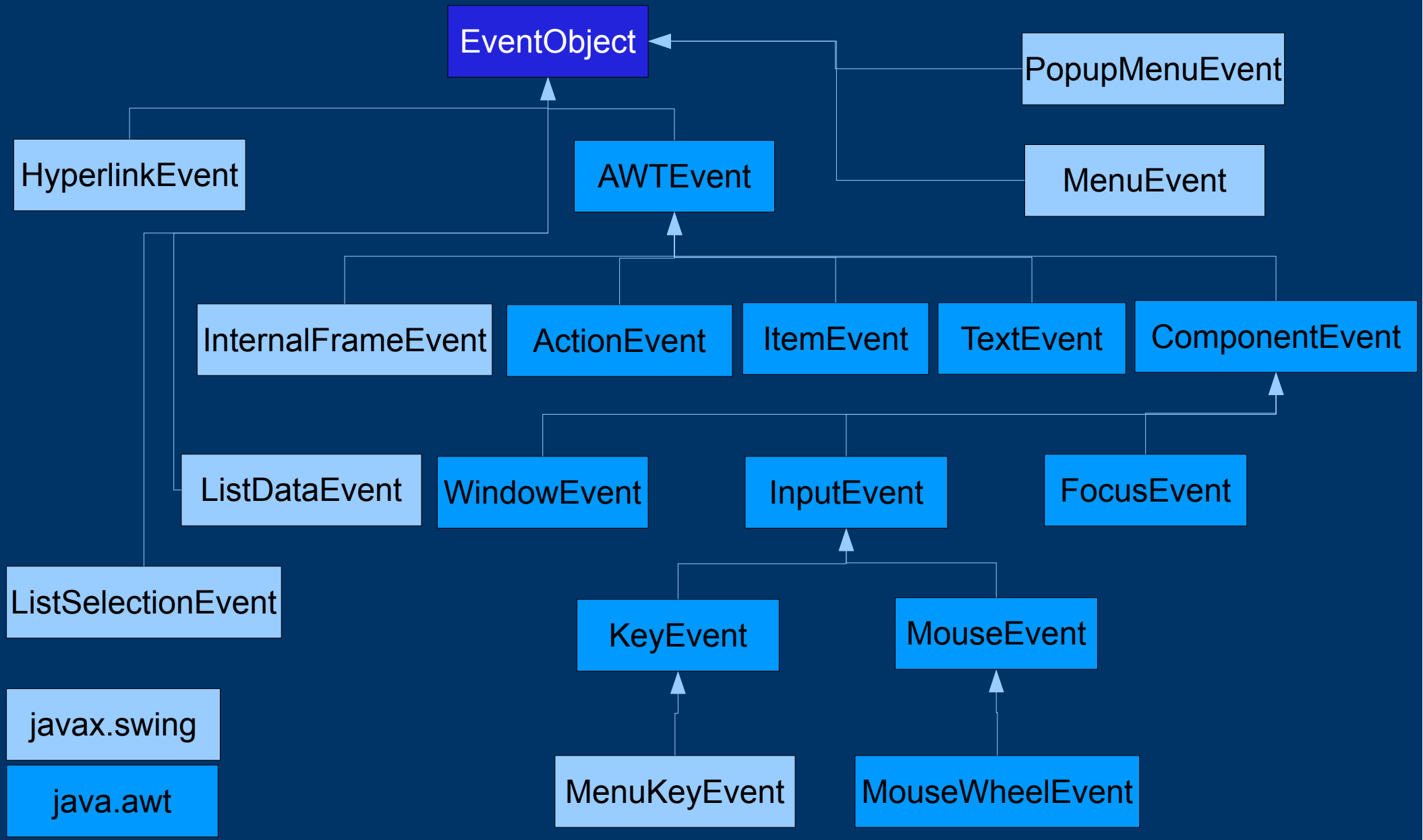
# Traitement d'un événement

- Principe général
  - Définition d'une classe implantant l'interface adéquate
    - En général, classe interne ou classe anonyme
  - Implanter toutes les méthodes de l'interface (à vide si événement inintéressant)
  - Créer un objet de ce type et l'enregistrer comme écouteur auprès de l'objet source
- Classes "Adapter" (ex. MouseAdapter)
  - Implantent à vide toutes les méthodes de l'interface associée
  - Classe de traitement peut en hériter plutôt que d'implanter l'interface adéquate

# Exemple

```
public class TestActionBouton extends JFrame {
    public TestActionBouton() {
        super("Test événement bouton");
        Container cont = getContentPane();
        JButton bouton = new JButton("Quitter");
        bouton.addActionListener (new MonEcouteur());
        cont.add(bouton,BorderLayout.CENTER);
        pack();
        setVisible(true);
    }

    class MonEcouteur implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
            System.exit(0);
        }
    }

    public static void main(String[] args) {
        JFrame fenetre = new TestActionBouton();
// on ne peut pas écrire MonEcouteur m = new MonEcouteur() ;
// on peut écrire MonEcouteur m = fenetre.new MonEcouteur() ;
    }
}
```

# Les principaux types d'événements



# *Les super-classes*

- EventObject
  - Deux méthodes
    - Object getSource()
      - Pour connaître l'objet ayant généré l'événement
    - String toString()
  - Un constructeur
    - EventObject(Object source)
- AWTEvent
  - String paramString()
  - Rien d'autre d'intéressant pour la programmation classique

# Evénements Action
# Classe ActionEvent

- Utilisation

  Événements simples genre validation d'un bouton
- Constructeurs
  - ActionEvent(Object source, int id, String command)
  - ActionEvent(Object source, int id, String command, int modifiers)
  - ActionEvent(Object source, int id, String command, long when, int modifiers)
- Méthodes
  - String getActionCommand()
  - Int getModifiers()
  - Long getWhen()
- Principes
  - Command : nom permettant d'associer des comportements différents pour un même objet source suivant les circonstances
  - Modifiers : touches de modification maintenues enfoncées lors de l'action (alt, ctrl, etc.)
  - When : timestamp de la date à laquelle l'événement est survenu

# *Evénements Action*
# *Interface ActionListener*

- Une seule méthode de traitement

  public void actionPerformed(ActionEvent ae)
- Pas de classe ActionAdapter
- Composants générateurs
  - Boutons (JButton)
  - Menus (JMenuItem)
  - Cases à cocher (JcheckBox)
  - Boutons radios (JradioButton)
  - Listes de choix (JComboBox)

# *Evénements Souris*
# *Classe MouseEvent*

- Constructeurs
  - MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger)
  - MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, int button)
  - MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int xAbs, int yAbs, int clickCount, boolean popupTrigger, int button)
- Quelques Méthodes
  - Int getButton()
  - Int getClickCount()
  - Int getModifiers()
  - Point getPoint(), int getX(), int getY()
  - Point getLocationOnScreen(), int getXOnScreen(), int getYOnScreen()

# *Evénements souris*
# *Interfaces d'écouteurs*

- MouseListener contient les méthodes
  - public void MouseClicked(MouseEvent me)
  - public void MouseEntered(MouseEvent me)
    Le pointeur de la souris vient d'arriver sur le composant
  - public void MouseExited(MouseEvent me)
    Le pointeur de la souris vient de quitter le composant
  - public void MousePressed(MouseEvent me)
  - public void MouseReleased(MouseEvent me)
- MouseMotionListener contient les méthodes
  - public void MouseDragged(MouseEvent me)
  - public void MouseMoved(MouseEvent me)
- MouseInputListener
  - Interface héritant de MouseListener et MouseInputListener

# *Evénements souris*
# *Classes d'écouteurs*

- MouseAdapter

  Classe implantant à vide toutes les méthodes de MouseListener

- MouseMotionAdapter

  Classe implantant à vide toutes les méthodes de MouseMotionListener

- MouseInputAdapter

  Classe implantant à vide toutes les méthodes de MouseListener et MouseMotionListener

# *Evénements Fenêtre*
# *Interface WindowListener*

Méthodes

- public void windowActivated(WindowEvent we)
- public void windowClosed(WindowEvent we)
- public void windowClosing(WindowEvent we)
- public void windowDeactivated(WindowEvent we)
- public void windowDeiconified(WindowEvent we)
- public void windowIconified(WindowEvent we)
- public void windowOpened(WindowEvent we)

# *Evénement Focus*
# *Classe FocusEvent*

- Notion de Focus
  - Composant actuellement actif
  - Peut être changé de façon permanente (avec touche <TAB> par exemple) ou temporaire (fenêtre désactivée par exemple)
- Méthodes
  - Component getOppositeComponent()

    Renvoie l'autre composant impliqué dans le changement de focus
  - Boolean isTemporary()

    Précise s'il s'agit d'un changement permanent ou temporaire

# *Evénement Focus*
# *Interface FocusListener*

Méthodes

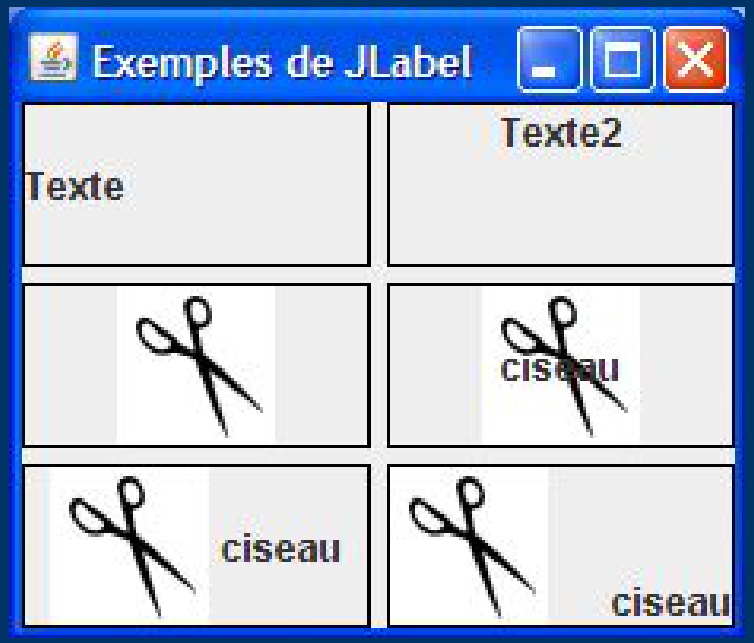
- public void focusGained(FocusEvent fe)
- public void focusLost(FocusEvent fe)

# *Evénement Item*

- ItemEvent
  - Méthode int getStateChange()
  - Peut renvoyer
    - ItemEvent.SELECTED
    - ItemEvent.DESELECTED
- ItemListener
  - Méthode public void itemStateChanged(ItemEvent ie)

# *Détails de quelques composants*

# Les étiquettes (JLabel)



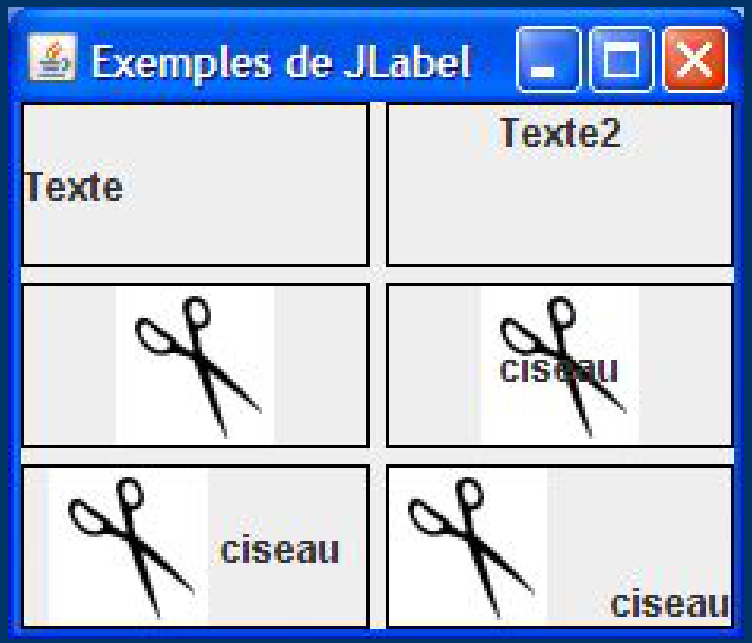

# Les étiquettes (JLabel)

- Rôle
  - Afficher un texte, une icône ou les deux
  - Interaction avec l'utilisateur nulle ou restreinte
  - Essentiellement informatif
- Constructeurs
  - JLabel()
  - JLabel(Icon image)
  - JLabel(String texte)
  - JLabel(Icon image, int alignementHoriz)
  - JLabel(String texte, int alignementHoriz)
  - JLabel(String texte, Icon image, int alignementHoriz)
- Alignement :
  - Position de l'icône, du texte, ou des deux

# Positionnement des éléments

- Horizontalement
  - public void setHorizontalAlignement(int h)
    - LEFT, RIGHT
    - CENTER (défaut si icône seule)
    - LEADING (défaut si texte seul)
    - TREALING
- Verticalement
  - Public void setVerticalAlignement(int v)
    - CENTER (défaut)
    - TOP, BOTTOM
- Positionnement relatif du texte/icône
  - setHorizontalTextPosition(int h)
  - SetVerticalTextPosition(int v)
- Espacement texte/icône
  - setIconTextGap(int pixels)

# Positionnement des éléments




```
JLabel l1 = new JLabel("Texte");
JLabel l2 = new JLabel(icone);
JLabel l3 = new JLabel("ciseau", icone, JLabel.CENTER);

Label l4 = new JLabel("Texte2");
l4.setHorizontalAlignment(JLabel.CENTER);
l4.setVerticalAlignment(JLabel.TOP);

JLabel l5 = new JLabel("ciseau", icone, JLabel.CENTER);
l5.setVerticalTextPosition(JLabel.CENTER);
l5.setHorizontalTextPosition(JLabel.CENTER);

JLabel l6 = new JLabel("ciseau", icone, JLabel.CENTER);
l6.setVerticalTextPosition(JLabel.BOTTOM);
l6.setHorizontalTextPosition(JLabel.RIGHT);
l6.setIconTextGap(20);
```

# *Autres paramétrages*

- void setDisabledIcon(Icone image)
- void setDisplayedMnemonic(char car)
- void setDisplayedMnemonic(int key)
- void setText(String text)
  Le texte d'un JLabel peut être en HTML
- void setLabelFor(Component comp)
  Le code mnémonique associé au label appellera comp.requestFocus()
- Couleurs et polices
  - Méthodes setFont, setBackground, setForeground des super-classes

# Autres paramétrages : exemple


```
        JLabel l1 = new JLabel("Texte");
        JLabel l2 = new JLabel("Texte2");

        l2.setForeground(Color.BLUE);

l2.setBackground(Color.YELLOW);l2.setOpaque(true);
        l2.setFont(new Font("SansSerif",Font.PLAIN,16));
```

# *Les boutons*
# *Vue + Contrôleur(JButton)*

# *Les boutons*
# *Vue + Contrôleur(JButton)*

- Rôle
  - Afficher un texte, une icône ou les deux
  - Le clic sur le bouton doit entraîner l'exécution d'une action
- Constructeurs
  - JButton()
  - JButton(String texte)
  - JButton(Icon image)
  - JButton(String texte, Icon image)
  - JButton(Action a)

    Voir partie sur les actions

# *Les boutons modèle (ButtonModel)*

- Interface de javax.swing
- Contient plusieurs propriétés représentant les états suivants :
  - Armé (boolean isArmed())
  - Pressé (boolean isPressed())
  - Sélectionné (boolean isSelected())
  - SousLaSouris (boolean isRollover())
  - Activé (boolean isEnabled())
- Peut être membre d'un groupe de boutons
- Peut avoir un code mnémonique
- Peut gérer des écouteurs d'ActionEvent, d'ItemEvent et de ChangeEvent

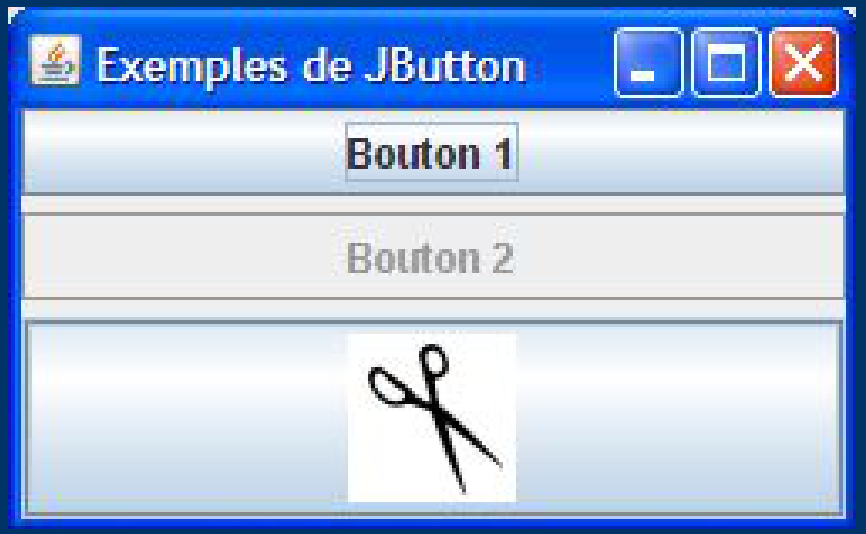
# *Les boutons*
# *Etats d'un bouton*

- Désactivé
  - Bouton grisé, aucun clic possible
- Pressed
  - On a cliqué sur le bouton, mais pas encore relâché le bouton de la souris
- Armed
  - On a cliqué sur le bouton, pas relâché le bouton de la souris, et le pointeur de la souris est sur le bouton
- Rollover
  - La souris est sur le bouton

# *Les boutons*
# *Mise en forme*

- Méthodes essentiellement définies dans AbstractButton, dont hérite JButton
- Mêmes principes que pour les JLabel :
  - Alignements horizontal et vertical
  - Positionnement relatif texte/icône
- Définition d'icônes différentes et du texte
  - setDisabledIcon(Icon image), setDisabledSelectedIcon(Icon image)
  - setIcon(Icon image), setText(String texte)
  - setPressedIcon(Icon image)
  - setRolloverIcon(Icon image), setRolloverSelectedIcon(Icon image)
  - setSelectedIcon(Icon image)
- Autres propriétés
  - setMargin(Insets marges)
  - setBorderPainted(boolean b)

# Les boutons
# Exemples




```
ImageIcon icone = new ImageIcon("c:\\Documents and Settings\\Bruno\\Bureau\\ciseaux.gif");

JButton l1 = new JButton("Bouton 1");
JButton l2 = new JButton("Bouton 2");
JButton l3 = new JButton(icone);

l2.setEnabled(false);
```

# *Les boutons*
# *bouton par défaut*

- Principe

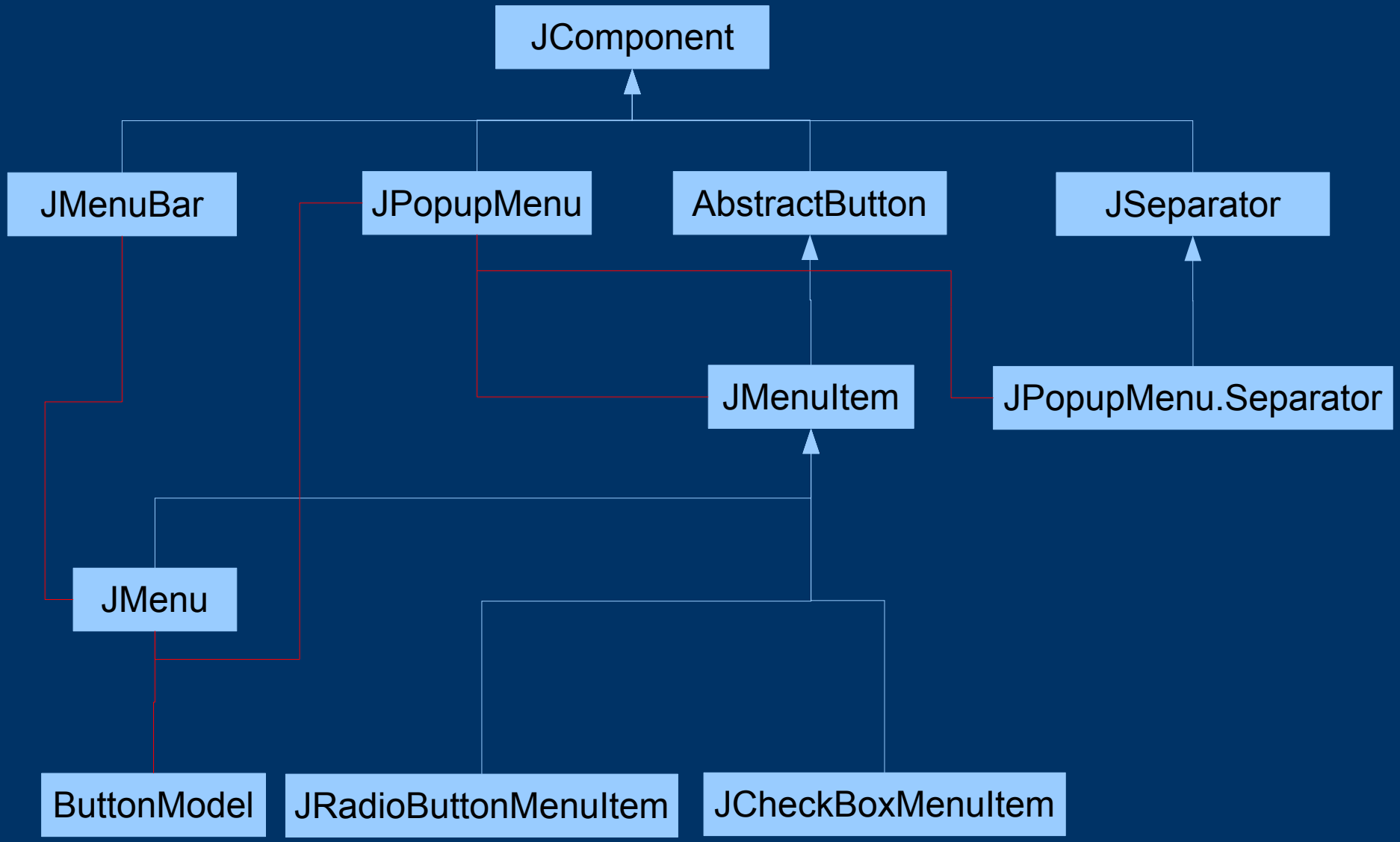
  Bouton pré-sélectionné d'un conteneur C et qui générera un actionPerformed lorsque C est activé et que l'on appuie sur Entrée

- Mise en œuvre

  Appeler setDefaultButton(JButton defaut) sur le conteneur contenant le bouton

# *Les menus*

# Aperçu de l'API



# *Menus : conseils de présentation*

Si possible :

- Pas plus de 10 items par menu
- Pas plus de 3 niveaux d'imbrication
- Limiter les items de type bouton radio ou case à cocher (peu intuitifs)
- Classer les items à l'intérieur des menus
- Utiliser les séparateurs

# *Menus*
# *classe JMenuBar*

- Constructeur : JMenuBar()
- Méthodes essentielles
  - void add(JMenu menu) / void setHelpMenu(JMenu m)
- Ajouter une barre de menu à une fenêtre
  - fenetre.setJMenuBar(JMenuBar maBarre)
- Ajouter un menu à une barre de menu
  - barre.add(JMenu menu)
- Supprimer un menu d'une barre de menu
  - barre.remove(Component menu)

# Menus
# Classe JMenu

- Rôle

  Créer un menu déroulant
  - Soit comme menu d'une barre (barre.add(JMenu menu))
  - Soit comme sous-menu d'un menu (menu.add(JMenuItem sousMenu)
- Constructeurs
  - JMenu(), JMenu(Action a)
  - JMenu(String s), JMenu(String s, boolean tear-off)
- Ajouter un article au menu
  - Component add(String s), Component add(Component c)
  - JMenuItem insert(String s, int pos), JMenuItem insert(JMenuItem m, int pos)
  - void addSeparator(), void insertSeparator()
- Supprimer un article de menu
  - remove(Component), remove(JMenuItem),
  - remove(int pos), removeAll()

# Menus
# Détection de la sélection

- Principe

  Ajouter un ActionListener au composant retourné par la méthode *add* de la classe JMenu

- Exemple

```
public class testMenuSuppression extends JFrame {
   private boolean edite;private JMenu fichier,editer,sousMenu; private JMenuBar barre;
   public testMenuSuppression() {
      super("suppression menu");
      barre = new JMenuBar();fichier = new JMenu("Fichier"); editer = new JMenu("Editer");
      sousMenu = new JMenu("Sous-Menu"); sousMenu.add("option 1");sousMenu.add("option 2");
      sousMenu.addSeparator(); JRadioButtonMenuItem b1 = new JRadioButtonMenuItem("Homme");
      JRadioButtonMenuItem b2 = new JRadioButtonMenuItem("Femme");
      ButtonGroup g = new ButtonGroup(); g.add(b1);g.add(b2);
      sousMenu.add(b1);sousMenu.add(b2) ;b1.setSelected(true); editer.add(sousMenu); edite = false;
      JMenuItem edition = new JMenuItem("edition");
      edition.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent ae) {barre.setVisible(false);
            if (edite) {edite = false;barre.remove(editer);} else {edite = true;barre.add(editer);}
            barre.setVisible(true);
      }});
      fichier.add(edition); barre.add(fichier); setJMenuBar(barre); pack(); setVisible(true);
}}
```

# Menus
# Les menus Pop-Up

- Principe

  Menu qui apparaît lors d'un certain clic sur un composant
- Mise en oeuvre
  - Créer le menu
  - Ajouter un écouteur de souris sur le composant
  - Pour les méthodes appropriées de l'interface MouseListener, tester si l'événement correspond bien à la demande d'un menu popup (méthode isPopupTrigger() de MouseEvent) en fonction du L&F
  - Si oui, afficher le menu au bon endroit

# Menus
# Les menus Pop-Up : exemple

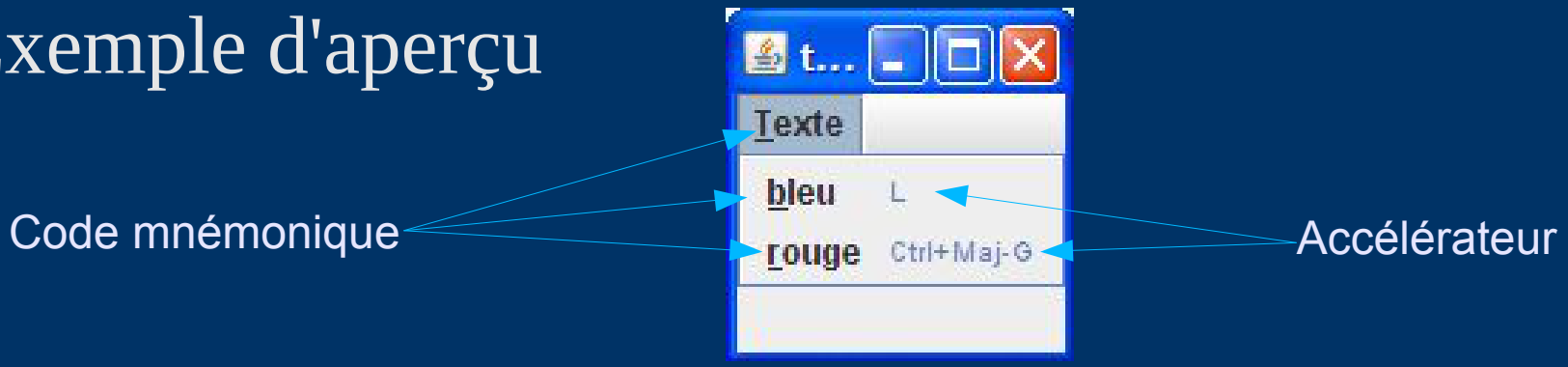
```
public class MenusPopup extends JFrame {
    private JPopupMenu popup;private JLabel etiquette;
    public MenusPopup() {
        super("Exemple pop-up");etiquette = new JLabel("Coucou");popup = new JPopupMenu();
        JMenuItem quitter = new JMenuItem("Quitter"); quitter.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {System.exit(0);}});
        JMenuItem opaque = new JMenuItem("Opaque");opaque.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {etiquette.setBackground(Color.YELLOW);
etiquette.setOpaque(true);}});
        JMenuItem trans = new JMenuItem("Transparent");trans.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {etiquette.setOpaque(false);etiquette.updateUI();}});
        popup.add(quitter);popup.add(opaque);popup.add(trans);
        etiquette.addMouseListener(new EcouteurPopUp());
        add(etiquette);
        pack();setDefaultCloseOperation(EXIT_ON_CLOSE);setVisible(true);
    }
    class EcouteurPopUp extends MouseAdapter {
        private void gerePopup(MouseEvent me) {
            if (me.isPopupTrigger()) {popup.show(etiquette, etiquette.getX(), etiquette.getY());}
        }
        public void mousePressed(MouseEvent me) {gerePopup(me);}
        public void mouseReleased(MouseEvent me) {gerePopup(me);}
    }
}
```

# *Raccourcis Claviers :*

## *Codes mnémoniques et Accélérateurs*

# Codes mnémoniques et raccourcis
## Introduction

- Code mnémonique
  Touche dont la combinaison avec une autre (alt en général) permettra d'activer une option de menu ou un bouton sans la souris lorsque le menu est affiché
- Accélérateur clavier
  Combinaison de touche permettant d'activer immédiatement une option d'un menu (ou le clic sur un bouton), même si aucun menu n'est déroulé
- Exemple d'aperçu



# *Utilisation des raccourcis claviers*

- Souris/Clavier
  - Souris
    - Intuitif, découverte automatique
    - Exécution lente
  - Clavier
    - A priori peu intuitif, nécessite de faire travailler sa mémoire
    - Exécution rapide
- Utilisation typique d'un logiciel avec IHM graphique
  - Phase de découverte : 100% souris
  - Phase d'apprentissage : part souris diminue, part clavier augmente
  - Phase d'utilisation intensive : part clavier majoritaire
- Conclusion
  - Les deux modes souris/clavier doivent cohabiter
  - Les raccourcis claviers doivent être aussi simples et intuitifs que possible

# Codes mnémoniques

- Composants visés
  - Boutons :
    - fait partie du modèle ButtonModel
    - Facilité d'accès depuis AbstractButton
  - Composants effectifs
    - JButton, JCheckBox, JRadioButton
    - JMenuItem, JMenu, JCheckBoxMenuItem, JRadioButtonMenuItem
- Méthodes de AbstractButton
  - public void setMnemonic(int mnemonic)
    souligne la première occurrence du caractère dans le nom du bouton
  - public void setDisplayedMnemonicIndex(int index)
    souligne la index$^{ième}$ occurrence du caractère dans le nom du bouton

# *Codes mnémoniques code du caractère*

Utilisation des constantes définies dans KeyEvent

Exemples

- Touche B : KeyEvent.VK_B (VK = Virtual Key)
- Touche € : KeyEvent.VK_EURO_SIGN
- Touche ↓ du clavier standard : KeyEvent.VK_DOWN
- Touche ↓ du pavé numérique : KeyEvent.VK_KP_DOUWN

# Codes mnémoniques Exemple

```
public class TestRaccourcisClaviers extends JFrame {
        JLabel affichage;

    public TestRaccourcisClaviers() {
        super("test raccourcis claviers");
        JMenuBar barre = new JMenuBar();
        JMenu menu = new JMenu("Texte");
        menu.setMnemonic(KeyEvent.VK_T);
        JPanel cont = new JPanel();
        cont.setLayout(new BorderLayout());
        setContentPane(cont);
        affichage = new JLabel("rien");
        cont.add(affichage,BorderLayout.CENTER);
        JMenuItem bleu = new JMenuItem("bleu");
        bleu.setMnemonic(KeyEvent.VK_B);
        bleu.addActionListener(new
ChangeLabel("bleu",Color.BLUE));
        menu.add(bleu);
        JMenuItem rouge = new JMenuItem("rouge");
        rouge.setMnemonic(KeyEvent.VK_R);
        rouge.addActionListener(new ChangeLabel("rouge",
Color.RED));
        menu.add(rouge);
        barre.add(menu);
        setJMenuBar(barre);
        pack();
        setVisible(true);
    }
    class ChangeLabel implements ActionListener {
        private String nom;
        private Color coul;
        ChangeLabel(String nom, Color coul) {
            this.nom=nom;
            this.coul=coul;
        }
        public void actionPerformed(ActionEvent ae) {
            affichage.setForeground(coul);
            affichage.setText(nom);
        }
    }

    public static void main(String[] args) {
        JFrame fenetre = new TestRaccourcisClaviers();
        fenetre.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

# *Accélérateurs*

- Composants visés
  - Éléments de menus
    - JMenuItem
    - JCheckBoxMenuItem
    - JRadioButtonMenuItem
  - Mais pas les menus
    - Redéfini dans Jmenu pour renvoyer une erreur
- Méthode de JMenuItem

  public void setAccelerator(KeyStroke k)

  KeyStroke : la combinaison de touche servant d'accélérateur

# *Accélérateurs*
# *Construction d'un objet KeyStroke*

- Utiliser les méthodes de classe de la classe KeyStroke parmi lesquelles :
  - static KeyStroke getKeyStroke(int keycode, int modifiers)
  - static KeyStroke getKeyStroke(int keycode, int modifiers, boolean onKeyRelease)
- Arguments :
  - keycode : code du caractère (c.f. Codes mnémoniques)
  - onKeyRelease : lorsqu'on relâche la touche ou bien lorsqu'on presse la touche (par défaut = presse)
  - modifiers : autres touches enfoncées

# *Accélérateurs et KeyStroke*
# *Paramètre modifiers*

Modifiers : "ou" bit-à-bit entre les valeurs suivantes :

- InputEvent.SHIFT_DOWN_MASK
- InputEvent.CTRL_DOWN_MASK
- InputEvent.META_DOWN_MASK
- InputEvent.ALT_DOWN_MASK
- InputEvent.ALT_GRAPH_DOWN_MASK

# Accélérateurs Exemple

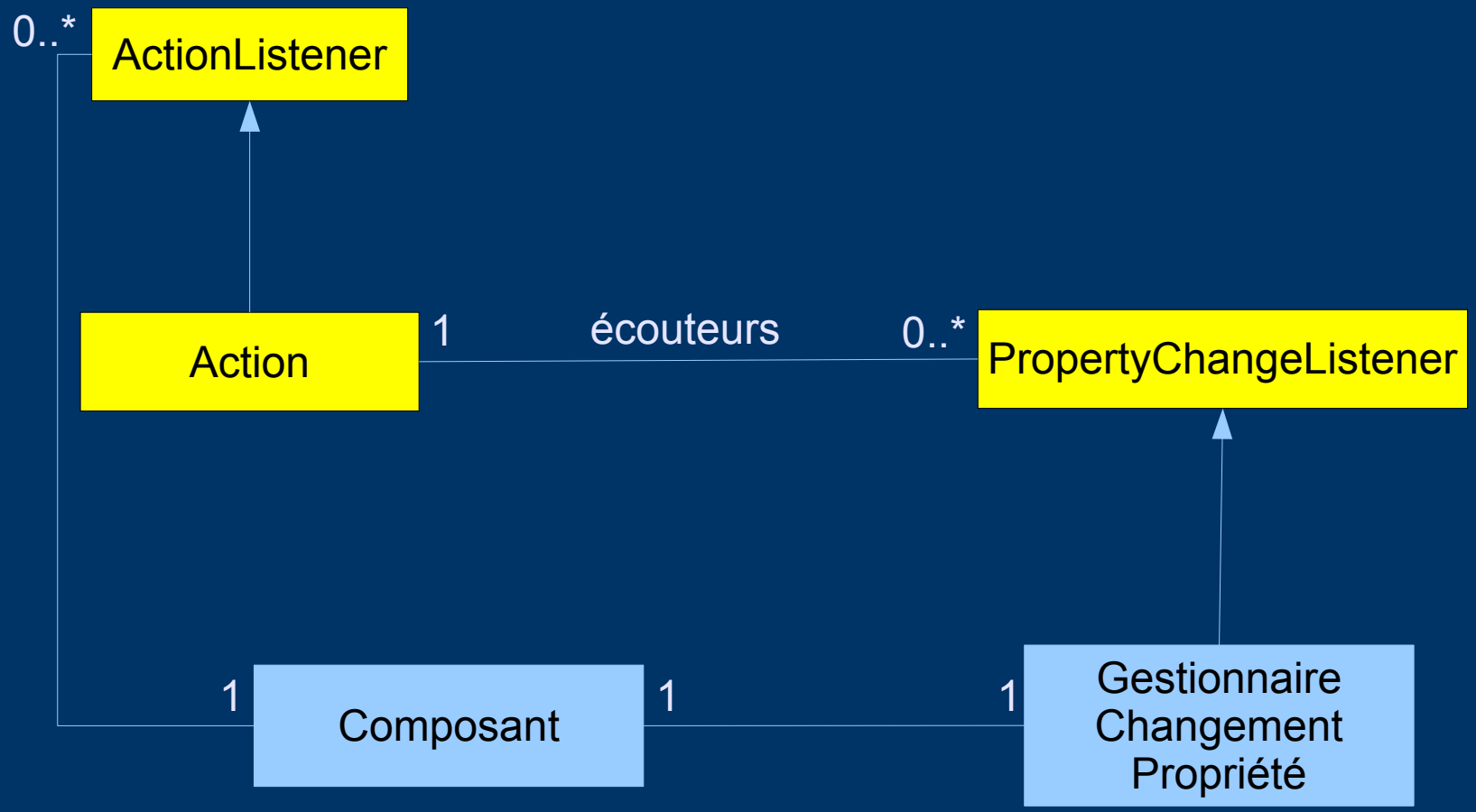
```
public class TestRaccourcisClaviers extends JFrame {
        JLabel affichage;

    public TestRaccourcisClaviers() {
        super("test raccourcis claviers");
        JMenuBar barre = new JMenuBar();
        JMenu menu = new Jmenu("Texte");
        JPanel cont = new JPanel();
        cont.setLayout(new BorderLayout());
        setContentPane(cont);
        affichage = new JLabel("rien");
        cont.add(affichage,BorderLayout.CENTER);
        JMenuItem bleu = new JMenuItem("bleu");
        bleu.setAccelerator( KeyStroke.getKeyStroke(
         KeyEvent.VK_L, 0));
        bleu.addActionListener(new
ChangeLabel("bleu",Color.BLUE));
        menu.add(bleu);
        JMenuItem rouge = new JMenuItem("rouge");
        rouge.setAccelerator(KeyStroke.getKeyStroke(
            KeyEvent.VK_G,
            InputEvent.CTRL_DOWN_MASK|
               InputEvent.SHIFT_DOWN_MASK));
        rouge.addActionListener(new ChangeLabel("rouge",
Color.RED));
        menu.add(rouge);
        barre.add(menu);
        setJMenuBar(barre);
        pack();
        setVisible(true);
    }
    class ChangeLabel implements ActionListener {
        private String nom;
        private Color coul;
        ChangeLabel(String nom, Color coul) {
            this.nom=nom;
            this.coul=coul;
        }
        public void actionPerformed(ActionEvent ae) {
            affichage.setForeground(coul);
            affichage.setText(nom);
        }
    }

    public static void main(String[] args) {
        JFrame fenetre = new TestRaccourcisClaviers();
        fenetre.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

# *Les <<actions>>*

# *Actions*
# *Problématique*

- Souvent plusieurs moyens de faire la même chose
  - Option de menu
  - Bouton d'une barre d'outils
  - Clic sur un bouton
  - Frappe Clavier
  - ...
- Comment valider/invalider d'un coup toutes ces commandes ?
- Solution : relier toutes ces commandes à un seul objet de type Action dont les commandes "observeront" les changements de propriété

# Les actions : architecture




Principes

- L'Action réagit aux événements (actionPerformed) émanant du Composant
- Le Composant réagit aux changements de propriété de l'Action

# Interface Action
# Classe AbstractAction

- Liste des méthodes
  - void addPropertyChangeListener(PropertyChangeListener l)
  - void removePropertyChangeListener(PropertyChangeListener l)
  - void setEnabled(boolean b)
  - boolean IsEnabled()
  - void putValue(String key, Object Value)
  - Object getValue(String key)
- Classe AbstractAction
  - Implante Action
  - Définit toutes les méthodes de AbstractAction sauf actionPerformed
  - Définit également :
    - Des constructeurs
      - AbstractAction(String name)
      - AbstractAction(String name, Icon icone)
    - De nouvelles méthodes
      - quelques autres méthodes :
      - FirePropertyChange(...)
      - Object [] getKeys()
      - PropertyChangeListener[] getPropertyChangeListeners()

# Propriétés prédéfinies de Action

| nom de la propriété de AbstractAction | propriété cible du composant |
|---|---|
| Enabled (via isEnabled) | enabled |
| SHORT_DESCRIPTION | toolTipText |
| ACTION_COMMAND_KEY | actionCommand |
| MNEMONIC_KEY | mnemonic |
| NAME | text |
| LARGE_ICON_KEY | icon (sauf JmenuItem si SMALL_ICON non nul) |
| SMALL_ICON | icon (JmenuItem) |
| ACCELERATOR_KEY | accelerator |
| SELECTED_KEY | selected |

Remarque

Si text et LARGE_ICON_KEY définis, un bouton affichera les deux. Pour qu'il n'affiche que l'icône, faire un setHideActionText(true) sur le bouton

# *Définir et utiliser une action*

- Définir l'action
  - Créer une classe héritant de AbstractAction
  - Appeler le constructeur de la superclasse requis
  - Définir la méthode actionPerformed
- Utiliser l'action
  - Créer les boutons et items de menu à partir de l'action
  - Désactiver éventuellement l'affichage du texte pour les boutons

# *Actions et frappes clavier*

# *Actions et Frappes Clavier*

- Problème
  - Comment définir des sortes d'accélérateur sans item de menu correspondant

  Ou
  - Comment déclencher automatiquement certaines actions sur certaines entrées clavier ?
- Solutions
  - InputMap et ActionMap

# *Actions et Frappes Clavier*
# *Principe général (1)*

- Une *InputMap* associe un objet clé (*Object*) à une frappe clavier (*KeyStroke*)
- Une *ActionMap* associe une action (*Action*) à une clé (*Object*)

=> Frappe clavier --> Object --> Action

Remarque :

La clé peut être n'importe quel objet. En général, on choisit le nom de l'action

# Actions et Frappes Clavier
# Principe général (2)

- Tout composant Swing a
  - 3 InputMap
    - WHEN_FOCUSED

      Valable uniquement quand le composant a le focus
    - WHEN_ANCESTOR_OR_FOCUSED_COMPONENT

      Quand le composant ou un de ses descendants à le focus
    - WHEN_IN_FOCUSED_WINDOW

      Quand la fenêtre du composant a le focus
  - 1 ActionMap

# Actions et Frappes Clavier Exemple

- Mettre un équivalent clavier (ctrl-b) à un bouton (*JButton*) créé à partir d'une action *actionBouton :* bouton = new JButton(actionBouton)
- Méthode 1 : Passer par la table WHEN_ANCESTOR_OR_FOCUSED_COMPONENT de la fenêtre

```
getRootPane().getInputMap(
        Jcomponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT).
        put(KeyStroke.getKeyStroke(KeyEvent.VK_B,
        KeyEvent.CTRL_DOWN_MASK),"bouton");
getRootPane().getActionMap().put("bouton", actionBouton);
```

- Méthode 2 : Passer par la table WHEN_IN_FOCUSED_WINDOW du Jbutton

```
bouton.getInputMap(
      Jcomponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(
      KeyEvent.VK_B, KeyEvent.CTRL_DOWN_MASK),"bouton");
bouton.getActionMap().put("bouton", actionBouton);
```