

Les bases graphiques : la librairie « Standard Widget Toolkit »

LES BASES GRAPHIQUES : LA LIBRAIRIE « STANDARD WIDGET TOOLKIT »	1
TABLE DES MATIERES	2
QUELQUES CONCEPTS DE BASE (1)	4
<i>une librairie graphique de base</i>	4
<i>les composants</i>	4
QUELQUES CONCEPTS DE BASE (2)	5
<i>un modèle d'événements</i>	5
QUELQUES CONCEPTS DE BASE (3)	6
<i>la libération explicite des ressources graphiques</i>	6
QUELQUES CONCEPTS DE BASE (3)	7
<i>la hiérarchie des principaux widgets</i>	7
LE DISPLAY	8
<i>Le rôle et les caractéristiques de la classe Display</i>	8
<i>quelques méthodes</i>	8
LE SHELL	9
<i>Le rôle et les caractéristiques de la classe Shell</i>	9
<i>quelques méthodes</i>	9
QUELQUES CONTROLS DE BASE (1)	10
<i>le label (la classe Label)</i>	10
<i>quelques méthodes</i>	10
<i>un exemple simplissime</i>	10
QUELQUES CONTROLS DE BASE : LE BUTTON	11
<i>les caractéristiques de la classe Button</i>	11
<i>quelques méthodes</i>	11
<i>un exemple simplissime</i>	11
LE TEXT (LA ZONE DE TEXTE)	12
<i>les caractéristiques de la classe Text</i>	12
<i>quelques méthodes</i>	12
LES LAYOUTS (1)	13
<i>quelques généralités</i>	13
<i>un premier Layout (le FillLayout)</i>	13
LES LAYOUTS (2)	14
<i>un second Layout (le RowLayout)</i>	14
<i>les données de formatage : la classe RowData</i>	14
<i>un exemple simple</i>	14

développement de plugins Eclipse

LES LAYOUTS (3).....	15
<i>un troisième layout (la classe GridLayout).....</i>	15
<i>les données de formatage : la classe GridData</i>	15
<i>un exemple simple.....</i>	15
LES CONTENEURS (1).....	16
<i>la classe Composite et ses sous-classes.....</i>	16
<i>quelques méthodes.....</i>	16
LES CONTENEURS (2).....	17
<i>quelques exemples.....</i>	17
LA GESTION DES EVENEMENTS	18
<i>le modèle de traitement des événements.....</i>	18
<i>les événements.....</i>	18
LES LISTENERS (1).....	19
<i>quelques interfaces Listener.....</i>	19
LES LISTENERS (2).....	20
<i>quelques interfaces Listener (suite).....</i>	20
<i>remarques.....</i>	20
LA GESTION DES EVENEMENTS - UN PREMIER EXEMPLE	21
<i>le réglage de la couleur de fond</i>	21
LA GESTION DES EVENEMENTS - UN SECOND EXEMPLE	22
<i>un jeu de réflexe.....</i>	22

Quelques concepts de base (1)

une librairie graphique de base

- librairie graphique de base
 - s'appuie au maximum sur des mécanismes de bas niveau de l'OS (philosophie inverse de SWING)
 - un ensemble de base de widgets (composants graphiques)
 - widgets de sélection (Menu, List, Combo, Checkbox,)
 - widget d'affichage-saisie (Text, ColorDialog, FileDialog, Tree, Table,)
 - dessins
 - conteneurs (Composite, Group, ScrolledComposite, ...)

les composants

- un modèle de conteneurs
 - création de conteneurs (Composite, dont Shell,)
 - définition des règles d'organisation interne des conteneurs via un Layout

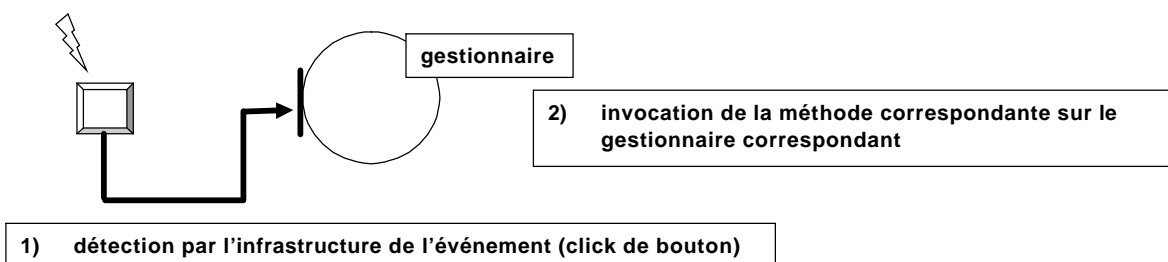
```
conteneur =new Conteneur(...);           //création d'un conteneur
layout = new XXXLayout();                 //création d'un Layout
conteneur.setLayout(layout);              //association d'un Layout au conteneur
new Composant1(conteneur, style1);        //définition des composants avec indication du conteneur
new Composant2(conteneur, style2);
```

Quelques concepts de base (2)

un modèle d'événements

- invocation (par les couches basses) de méthodes sur des interfaces de callback (les Listeners)
 - création de gestionnaires qui implémentent les interfaces
 - association des gestionnaires aux widgets (les widgets se comportent comme des observers)
 - événement : objet qui regroupe les informations caractérisant l'événement

```
.....  
gestion=new XXXListenerImpl( ) ;           //création d'un gestionnaire d'événement  
composant.addXXXListener(gestion) ;       //association du gestionnaire à un composant  
.....
```



Quelques concepts de base (3)

la libération explicite des ressources graphiques

- nécessité de libérer les ressources par un dispose explicite sur le widget
- **règle 1 : « if you created it, you dispose it »**

```
Color col1 = new Color(display, 255, 0, 0);           //création explicite de col1
.....
col1.dispose();                                     //destruction explicite de col1
.....
Color col2 = display.getSystemColor(SWT.COLOR_RED); //obtention de col2 (sans création explicite)
                                                    //pas de destruction explicite
```

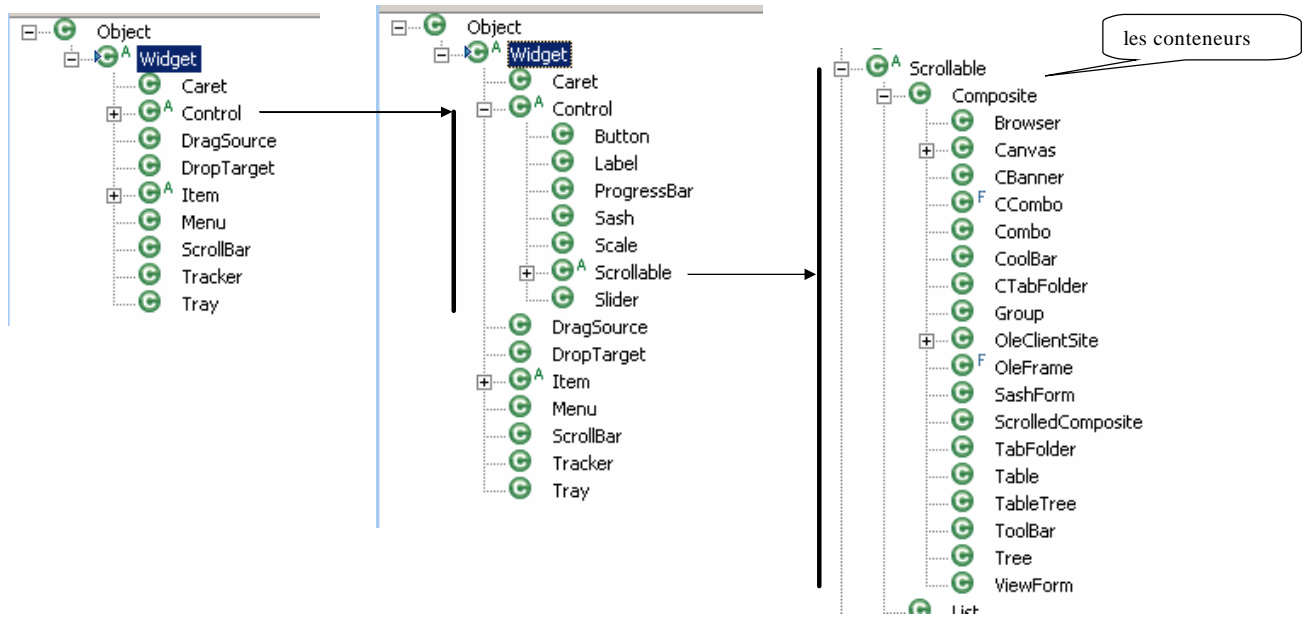
- contrôle du cycle de vie via la relation parent-enfant exprimée via le constructeur
 - ➔ création du conteneur obligatoirement avant le contenu
un composant ne peut pas appartenir à plusieurs conteneurs
- **règle 2 : « disposing the parent, dispose the children »**

la relation parent-enfant correspond généralement à la relation conteneur-contenu

Quelques concepts de base (3)

la hiérarchie des principaux widgets

- **Widget** : classe parente de tous les widgets
définit des méthodes communes à tous les widget : **pack()**, **computeSize()**,



Le rôle et les caractéristiques de la classe Display

possède des méthodes spécifiques au système

- assure le lien entre les classes SWT et le système de fenêtrage système
- offre le mécanisme de traitement des événements (gestion des listeners et des événements)

quelques méthodes

```

Display()
static Display getCurrent(), static Display getDefault()

Rectangle getBounds()           //retourne la taille du Display
Rectangle getClientArea()       //retourne la taille du Display pouvant afficher des données
Color getSystemColor(int id)    //retourne la couleur identifiée par id (constante de SWT)

Shell[ ] getShells()           //retourne les shells associés au Display
Thread getThread()             //retourne le thread propriétaire du Display

void asyncExec(Runnable r), void syncExec(Runnable r), void timerExec(int m, Runnable r)
//requête d'exécution « non bloquante », « bloquante », « différée »
void update()                  //exécution des requêtes de rafraichissement

boolean readAndDispatch()       //retire les événements de l'event queue « système ».
//invoque les listeners correspondants.
//traite les requêtes syncExec ou asyncExec en l'absence d'événement.
//retourne true si il y encore des événements à traiter

void sleep()                   //autorise le thread à libérer la CPU jusqu'à l'arrivée de nouveaux événements

```

le thread créateur d'un objet Display est l'UNIQUE possesseur de ses objets descendants. Tout accès par un autre thread génère l'exception SWT.ERROR_THREAD_INVALID_ACCESS

Le Shell

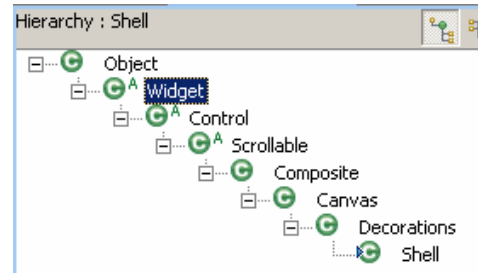
Le rôle et les caractéristiques de la classe Shell

- **Shell** : une fenêtre « racine » indépendante avec une barre de titre et des bordures pas de barre de menu, ni menu, ni scrollbars, ...

Shell() et Shell(int style) dépréciés

quelques méthodes

```
Shell(Display disp, int style);           //fenêtre racine
Shell(Shell parent, int style);         //fenêtre secondaire
void open(), void close( );
void dispose() ;                        //pour une fenêtre racine
void setVisible(boolean mode), boolean isVisible()
void setActive() ;
void setEnabled(boolean mode), boolean isEnabled()
void pack()
```



un shell est affiché par défaut en utilisant une taille recommandée par le « système ».
La méthode pack() calcule puis affecte la taille exacte requise par le shell

- les styles sont des constantes de la classe SWT : BORDER, CLOSE, MIN, MAX, RESIZE, TITLE, NO_TRIM, APPLICATION_MODAL, MODELESS, PRIMARY_MODAL, SYSTEM_MODAL

```
Display display = new Display();
Shell shell = new Shell(display);
shell.open() ;
while ( !shell.isDisposed() ) { if ( !display.readAndDispatch() ) display.sleep() ; }
display.dispose() ;
```

Quelques Controls de base (1)

le label (la classe Label)

- une chaîne de caractère OU une image.
- Quelques styles (constantes de la classe SWT) :
SEPARATOR, HORIZONTAL, VERTICAL, LEFT, RIGHT, WRAP, BORDER, SHADOW_IN, SHADOW_OUT, SHADOW_NONE,

quelques méthodes

```
void setText(String aText) , String getText()  
void setImage(Image almage), Image getImage()  
setBackground(Color aCol), Color getBackground()
```

un exemple simplissime

```
Label l1 = new Label(parent, SWT.BORDER | SWT.LEFT | SWT.SHADOW_IN);  
l1.setBackground(new Color(display,124,110,68)) ;  
l1.setText("ceci est un label");
```

Quelques Controls de base : le Button

les caractéristiques de la classe Button

- Control sélectionnable dont le rôle est de générer un événement sur un click de souris : bouton standard, checkbox, toggle, radio bouton
- quelques styles possibles (constantes de la classe SWT)
 - Type de bouton : ARROW (sens de la flèche indiqué via UP, DOWN, LEFT, RIGHT),
 - CHECK, PUSH, RADIO, TOGGLE, FLAT,
 - Alignement du texte : CENTER, LEFT, RIGHT



quelques méthodes

`String getText(), Image getImage(), boolean getSelection(),
setText(String txt), setImage(Image img), setSelection(boolean isS),`

+ méthodes de gestion des Listeners (voir suite)

un exemple simplissime

```
Button b1 = new Button(parent, SWT.BORDER | SWT.SHADOW_IN);  
b1.setBackground(new Color(display,124,110,68)) ;  
b1.setText("appuyez ici");
```

le Text (la zone de texte)

les caractéristiques de la classe Text

- saisie du texte dans une zone d'une ligne, et de largeur paramétrable (exprimée en nb de caractères)
- quelques styles possibles (constantes de la classe SWT)
BORDER, H_SCROLL, V_SCROLL, MULTI, SINGLE, WRAP, PASSWD, READ_ONLY,

quelques méthodes

String getText(), void setText(String txt)
void append(String str), void insert(String str)
void copy(), void cut(), void paste()

void setEchoChar(char ec), char getEchoChar()
void setTextLimit(int lm), int getTextLimit()
int getTopIndex();, void setTopIndex(int tp)

int getCaretLineNumber(), int getCaretLinePosition()
int getCharCount()
int getLineCount()

+ méthodes de gestion des Listeners (voir suite)

Les Layouts (1)

quelques généralités

- introduisent un découplage entre un composite et les composants contenus
- placement dynamique cherchant à conserver les tailles relatives du contenu
- 4 layouts principaux : FillLayout, RowLayout, GridLayout, FormLayout

un premier Layout (le FillLayout)

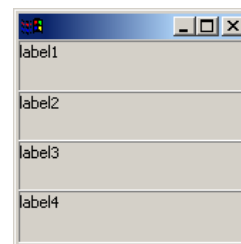
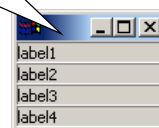
- organisation en une seule ligne ou une seule colonne
- tous les composants ont **même taille**, pas de retour à la ligne, **remplissage** du conteneur (avec modification de taille des widgets contenus)

```
Display display = new Display();
Shell shell = new Shell(display);
shell.setLayout(new FillLayout());
Label l1 = new Label(shell, SWT.BORDER);
l1.setText("un label1 long");
Label l2 = new Label(shell, SWT.BORDER);
l2.setText("label2");
Label l3 = new Label(shell, SWT.BORDER);
l3.setText("label3");
Label l4 = new Label(shell, SWT.BORDER);
l4.setText("label4");
shell.pack();
```

SWT.HORIZONTAL



SWT.VERTICAL



un second Layout (le RowLayout)

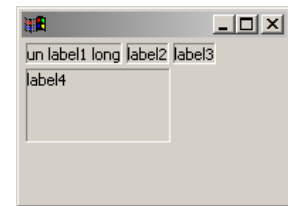
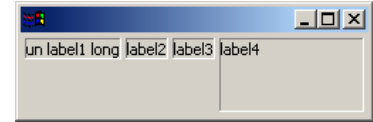
- organisation horizontale ou verticale simple (semblable aux FillLayout)
- les principaux attributs sont : **wrap**, **pack**, **justify**, **spacing**, **marginLeft**, **marginRight**, **marginTop**, ...
- les composants ont leur taille individuelle, retour à la ligne si nécessaire, non remplissage du composite

les données de formatage : la classe RowData

- les constructeurs sont : **RowData()**, **RowData(int width, int height)**
- possibilité de fixer la taille d'un composant via **setLayoutData(RowData data)**

un exemple simple

```
Display display = new Display();
Shell shell = new Shell(display);
shell.setLayout(new RowLayout(SWT.HORIZONTAL));
Label l1 = new Label(shell, SWT.BORDER);
l1.setText("un label1 long");
Label l2 = new Label(shell, SWT.BORDER);
l2.setText("label2");
Label l3 = new Label(shell, SWT.BORDER);
l3.setText("label3");
Label l4 = new Label(shell, SWT.BORDER);
l4.setText("label4");
l4.setLayoutData(new RowData(100,50));
shell.pack();
```



Les Layouts (3)

un troisième layout (la classe GridLayout)

- organisation des composants selon une grille (remplissage par défaut en séquence)
- **constructeurs** : `GridLayout()`, `GridLayout(int nCol, boolean cEqualW)`
- les principaux attributs sont : `horizontalSpacing`, `makeColumnEqualWidth`, `marginWidth`, `numColumns`, `verticalSpacing`

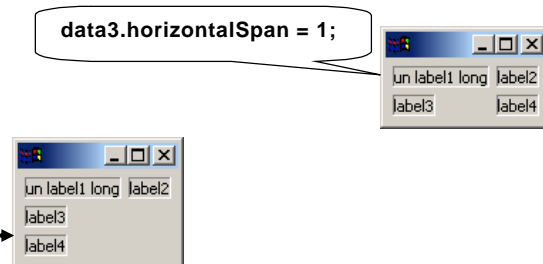
les données de formatage : la classe GridData

- les constructeurs sont : `GridData()`, `GridData(int style)`
- les attributs sont : `horizontalSpan`, `verticalSpan`, `horizontalAlignment`, `verticalAlignment`, `heightHint`, `widthHint`, ...
- possibilité de fixer la taille des composants via `setLayoutData(GridData data)`

BEGINNING, CENTER, END, FILL

un exemple simple

```
shell.setLayout(new GridLayout(2, false));
Label I1 = new Label(shell, SWT.BORDER);
I1.setText("un label1 long");
Label I2 = new Label(shell, SWT.BORDER);
I2.setText("label2");
Label I3 = new Label(shell, SWT.BORDER);
I3.setText("label3");
GridData data3 = new GridData();
data3.horizontalSpan = 2;
I3.setLayoutData(data3);
Label I4 = new Label(shell, SWT.BORDER);
I4.setText("label4");
```



Les conteneurs (1)

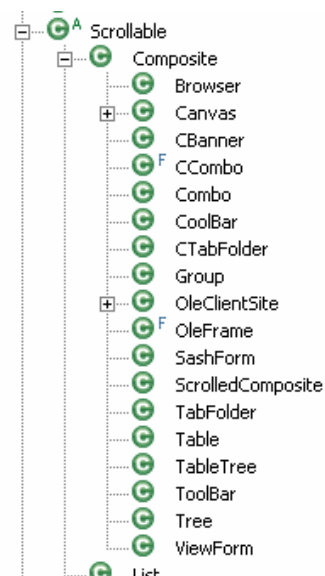
la classe Composite et ses sous-classes

- **Composite**: un objet Control conteneur
- Les styles possibles (constantes de la classe SWT) :
SWT.BORDER, NO_BACKGROUND, H_SCROLL, V_SCROLL, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE, NO_RADIO_GROUP

quelques méthodes

```

void computeSize(int wHint, int hHint, boolean changed)
Control[] getChildren()
void layout()
void setLayout(Layout layout);
void setFocus();
    
```



Les conteneurs (2)

quelques exemples

```
Composite composite1 = new Composite(shell,SWT.BORDER);
composite1.setBounds(10,10,270,250);
composite1.setBackground(new Color(display,31,133,31));
Label label = new Label(composite1,SWT.NONE);
label.setText("Here is a green composite");
label.setBounds(10,10,200,20);
Composite composite2 = new Composite(composite1,SWT.H_SCROLL | SWT.V_SCROLL);
composite2.setBounds(10,40,200,200) ;
List list = new List(composite2,SWT.MULTI);
for (int i=0; i<50; i++) { list.add("Item " + i); }
list.setSize(300,300);
```

```
Group group1 = new Group(shell, SWT.BORDER);
group1.setBounds(30,30,200,200);
group1.setText("Group 1");
Button button = new Button(group1, SWT.PUSH);
button.setBounds(10,20,80,20);
button.setText("I'm in a group");
Label label = new Label(group1, SWT.NONE);
label.setBounds(10,50,80,20);
label.setText("So am I!!");
Group group2 = new Group (group1, SWT.NONE);
group2.setBounds(10,100,150,50) ;
group2.setBackground(new Color(display,233,20,233)) ;
group2.setText("I'm a group inside a group")
Button button2 = new Button(group2, SWT.PUSH);
button2.setBounds(10,20,50,20) ;
button2.setText("Twice..");
```

La gestion des événements

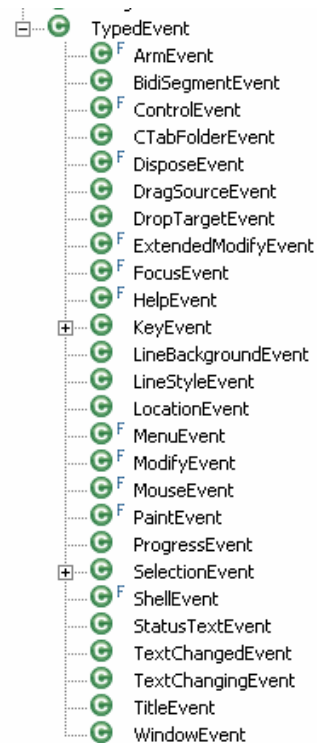
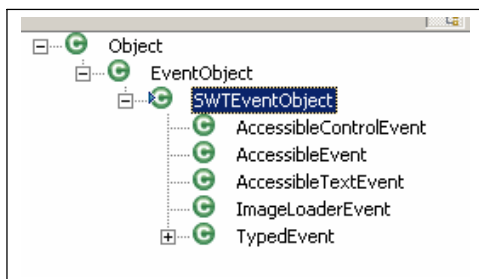
le modèle de traitement des événements

- SWT réagit à l'action d'un utilisateur par la génération d'un événement
- SWT invoque par callback une méthode d'une interface **Listener**
- création d'un gestionnaire d'événements (implémentation de "Listener")
- association aux composants graphiques : **add<X>Listener(<X>Event evt)**

- **Listener** : interface précisant les réactions<aux événements>

les événements

- événement = objet décrivant l'événement (via des champs spécifiques **public**)
- informations toujours disponibles : Widget **widget**, int time, Display **display**
- les types d'événements définis par des constantes de **SWT**



Les Listeners (1)

quelques interfaces Listener

- La gestion des Text (et classes dérivées)

KeyListener	la frappe de touches au clavier	KeyEvent
ModifyListener	les modifications de texte (après)	ModifyEvent
VerifyListener	les modifications de texte (avant)	VerifyEvent

- La gestion de la souris

MouseListener	les clics de souris	MouseEvent
MouseMoveListener	les mouvements de souris	MouseMoveEvent
MouseTrackListener	l'entrée, la sortie de widget par la souris	MouseTrackEvent

- La gestion de l'affichage et du dessin (essentiellement dans les Canvas)

PaintListener	le dessin de widget	PaintEvent
----------------------	----------------------------	-------------------

- La gestion des sélections

SelectionListener	la sélection de widget	SelectionEvent
TreeListener	la manipulation d'arbres	TreeEvent

MenuListener	la manipulation de menus	MenuEvent
---------------------	---------------------------------	------------------

Les Listeners (2)

quelques interfaces Listener (suite)

- Divers

ControlListener	le déplacement et redimensionnement d'un widget	ControlEvent
DisposeListener	la libération d'un widget	DisposeEvent
FocusListener	l'obtention ou la perte du focus	FocusEvent
HelpListener	la demande d'aide	HelpEvent

remarques

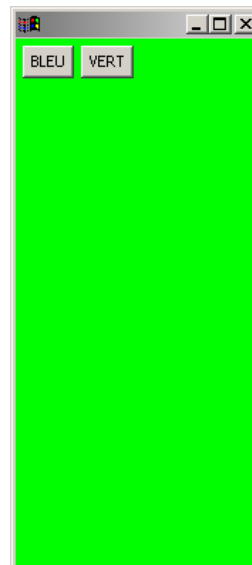
- un gestionnaire peut implémenter plusieurs Listeners
- classes d'Adapter qui implémentent un Listener avec méthodes souvent vides)

le réglage de la couleur de fond ...

.....

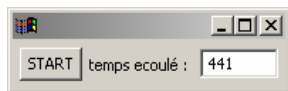
```
public static void main(String[] args) {
    Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout(2,false));
    Button bBlue = new Button(shell, SWT.PUSH);
    bBlue.setText("BLEU");
    final Color blue = display.getSystemColor(SWT.COLOR_BLUE);
    bBlue.addSelectionListener(new SelectionListener() {
        public void widgetSelected(SelectionEvent e) { shell.setBackground(blue);}
        public void widgetDefaultSelected(SelectionEvent e) {}
    });
    final Button bGreen = new Button(shell, SWT.PUSH);
    bGreen.setText("VERT");
    final Color green = display.getSystemColor(SWT.COLOR_GREEN);
    bGreen.addSelectionListener(new SelectionListener() {
        public void widgetSelected(SelectionEvent e) { shell.setBackground(green);}
        public void widgetDefaultSelected(SelectionEvent e) {}
    });

    shell.setBounds(0,0,200,400);
    shell.open() ;
    while ( !shell.isDisposed()) { if ( !display.readAndDispatch()) display.sleep() ; }
    display.dispose() ;
}
```



La gestion des événements - un second exemple

un jeu de réflexe...



```
public static void main(String[] args) {
    final Display display = new Display();
    final Shell shell = new Shell(display);
    shell.setLayout(new GridLayout(3,false));
    final Button button =
        new Button(shell, SWT.PUSH);
    button.setText("START");
    final Label label =
        new Label(shell, SWT.NONE);
    label.setText("temps écoulé : ");
    final Text text =
        new Text(shell, SWT.BORDER);
    text.setText("      ");
    ←
    shell.pack();
    shell.open() ;
    while ( !shell.isDisposed()) { if ( !display.readAndDispatch()) display.sleep() ; }
    display.dispose() ;
}
```

```
button.addSelectionListener(new SelectionListener() {
    boolean demar = false;
    long deb;
    public void widgetSelected(SelectionEvent e) {
        if (!demar) {
            button.setText("START");
            demar = true;
            display.timerExec((int)(Math.random() * 1000), new Runnable() {
                public void run() {
                    deb = System.currentTimeMillis();
                    button.setText("CLICK");
                }
            });
        } else {
            text.setText("" + (System.currentTimeMillis() - deb));
            button.setText("START");
            demar = false;
        }
    }
    public void widgetDefaultSelected(SelectionEvent e) {}
});
```