



Tutoriel Java – SWT (Standard Widget Toolkit)

Introduction

- **Introduction**

Dans ce tutoriel, vous découvrirez comment mettre en œuvre une interface graphique Java à l'aide de **SWT** (Standard Widget Toolkit).

Pour cela, nous utiliserons l'outil **WindowBuilder** sous Eclipse.

1

Présentation

- **Présentation**

En Java, on connaît trois principales bibliothèques graphiques:

- **AWT** : la première historiquement et proche des widgets natifs des OS
- **SWING** : la bibliothèque purement Java
- **SWT** : qui utilise les widgets natifs lorsqu'ils sont disponibles, et des widgets Java quand ils ne le sont pas.

1

Présentation

- **Présentation**

Eclipse est un environnement de développement intégré (IDE) initialement destiné au développement Java – et lui-même écrit en Java.

A la sortie d'Eclipse, les utilisateurs ont apprécié la réactivité de l'interface utilisateur.

En effet, "à l'époque", les applications avec interface graphique portable avaient une faible vitesse de réaction.

1

Présentation

- **Présentation**

Eclipse n'utilise pas les bibliothèques graphiques Java standard : **AWT** et **Swing**.

Il a une bibliothèque spécifiquement développée par IBM et la fondation Eclipse: **SWT** qui dispose d'une architecture logicielle particulière.

1

Présentation

- **Présentation**

Avec **Swing**, la bibliothèque graphique (écrite en Java) est indépendante de la plate-forme d'exécution.

Le point faible de cette approche est que c'est Java qui prend en charge toute la gestion de l'interface graphique.

Ceci empêche de se reposer sur les services offerts par l'environnement d'exécution hôte (Windows, Mac OSX, ...).

1

Présentation

- **Présentation**

A l'inverse, **SWT** s'appuie sur les systèmes d'exploitation qui offrent des composants logiciels capables de gérer de façon **performante** et **efficace** les éléments de l'**IHM**.

On peut dire que **SWT** est une couche d'interface qui permet à une application Java d'utiliser les composants natifs du système hôte.

1

Présentation

- **Présentation**

Inconvénients :

- SWT est fortement lié au système hôte. Il est nécessaire d'installer une version spécifique de SWT adaptée à la plate-forme d'exécution, ce qui est en contradiction avec la philosophie de Java « Write once, run everywhere ».

1

Présentation

- **Présentation**

- Pour exécuter une application utilisant SWT, il faut être sur une plate-forme sur laquelle SWT a été porté.

Heureusement, SWT tourne sous les principaux environnements graphiques PC, Mac et Unix, ainsi que sur certains systèmes embarqués comme WindowsMobile.

- En revanche, pour développer sur une plate-forme un peu exotique AWT/Swing serait peut-être un meilleur choix.

1

Présentation

- **Présentation**

- Second inconvénient : les objets de l'environnement hôte utilisés par SWT échappent au ramasse miette de Java.

C'est au programmeur qu'incombe la responsabilité de désigner les ressources SWT qui ne lui seront plus utiles afin qu'elles puissent être immédiatement libérées.

Dans la pratique, cela consiste à appeler la méthode « dispose ». A ne pas oublier...

1

Présentation

- **Présentation**

Avantages :

- des performances très satisfaisantes sur tous les matériels pour lesquels SWT a été porté,
- les applications utilisant SWT possèdent le même « look and feel » que les applications natives : avantage important du point de vue de l'utilisateur.

1

Présentation

- **SWT : Standard Widget Toolkit**

SWT offre une solution technique qui permet de mélanger composants Swing et SWT.

Le développement d'une application SWT est assez similaire à celui des applications AWT et Swing : on y retrouve notamment la notion de « layout » et le même principe de gestion des événements.

1

Présentation

- **SWT : Standard Widget Toolkit**

La librairie SWT s'interface avec les composants natifs.

Elle est constituée :

- d'une **partie de code Java commune** à tous les systèmes d'exploitation
- d'une **partie native propre** à chaque SE

Sur Windows déploiement transparent car ces DLL sont intégrées dans un des fichiers JAR d'Eclipse.

1

Présentation

- **SWT : Standard Widget Toolkit**

SWT est principalement utilisée dans Eclipse, mais tourne également en-dehors, dans des applications stand-alone.

Problème : SWT seul ne facilite pas la mise en œuvre de patrons d'architecture tels que MVC.

Solution : pour cela, il y a une librairie au-dessus de SWT qui s'appelle **JFace**.

1

Présentation

- **Présentation**

En résumé :

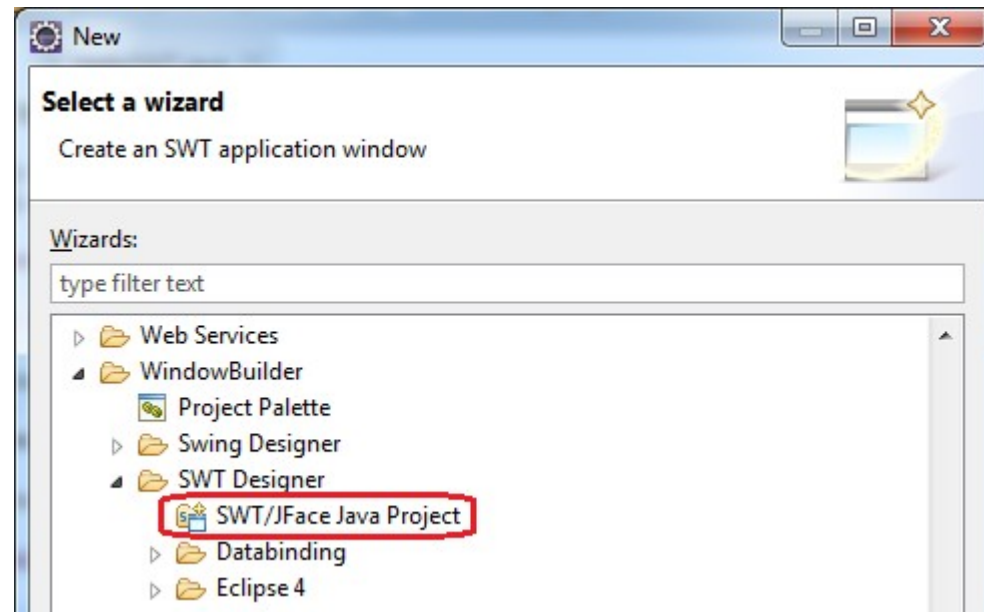
- SWT a ses points forts et ses points faibles.
- SWT n'est pas un remplacement pour AWT ou Swing. C'est juste une autre bibliothèque d'interface graphique, basée sur d'autres principes que les bibliothèques standard Java.
- Donc, utiliser l'une ou l'autre de ces bibliothèques doit être un choix conscient et réfléchi.

2

Création projet

- **Création d'un projet SWT**

Création d'un projet SWT/JFace Java Project
File > New > Other



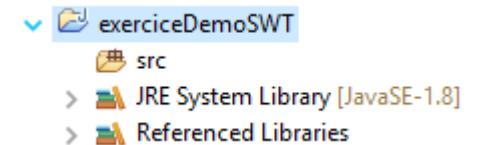
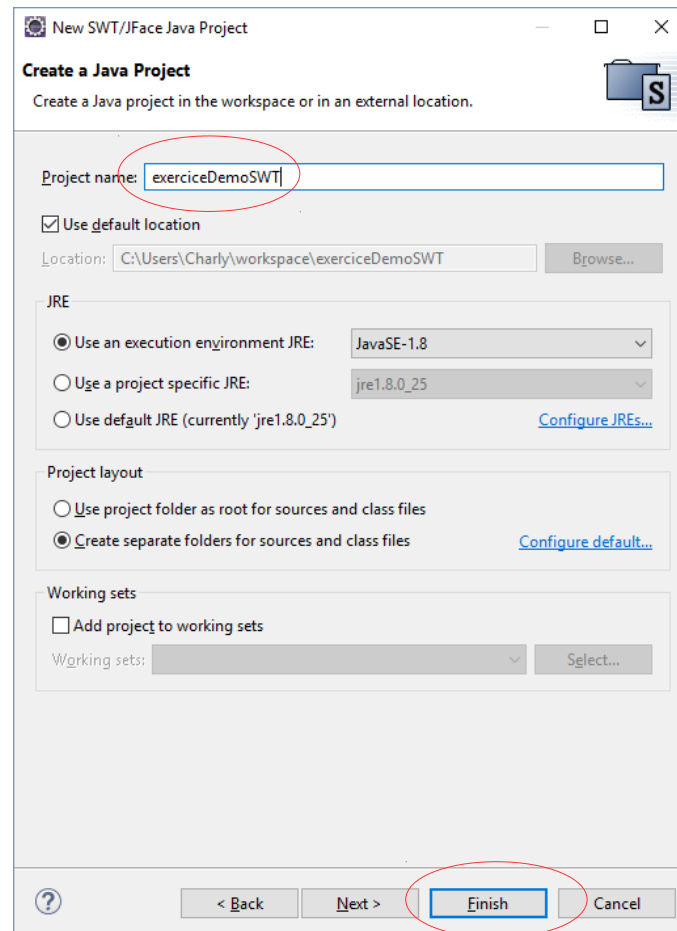
Choisir «SWT/JFace Java Project», puis Next

2

Création projet

- **Création d'un projet**

Complétez le nom puis cliquez sur « Finish ».

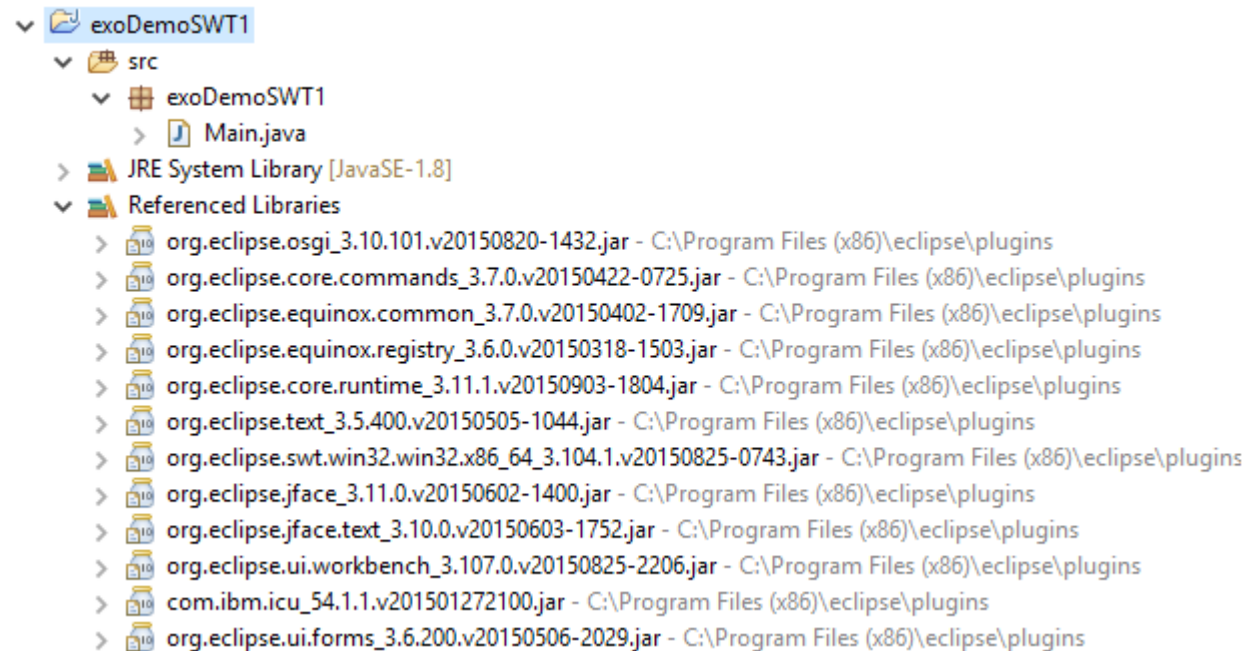


2

Création projet

- **Création d'un projet**

Dans le projet créé, on a les librairies qui vont permettre la mise en œuvre de SWT



2

Création projet

- **Création d'un projet**

Deux solutions pour créer un projet SWT :

- 1)Écrire le code « à la main »
- 2)Utiliser une interface graphique.

Dans un but pédagogique, nous allons aborder ces deux méthodes de travail.

3

Le code

- **Création d'un projet SWT – méthode 1**

Toute application SWT doit créer une instance de la classe **org.eclipse.swt.widgets.Display**.

C'est cette instance qui servira à SWT d'interface avec le système natif.

Comme tous les objets SWT qui interagissent directement avec le système hôte, il faudra penser à appeler la méthode ***dispose()*** lorsqu'il ne sera plus nécessaire à l'application.

Dans le cas d'une instance de la classe Display, ce sera à la fin du programme.

3

Le code

• Création d'un projet SWT – méthode 1

```
package main;

import org.eclipse.swt.widgets.Display;

public class Main {

    public static void main(String[] args) {
        Display display = new Display();

        /* Tout le code utilisant SWT ici */
        display.dispose();

    }
}
```

Le code utilisant SWT est situé entre la création de l'instance de la classe **Display** et la libération des ressources associées à cet objet (**display.dispose()**).

3

Le code

- **Création d'un projet SWT – méthode 1**

Ajouter une fenêtre en insérant les commandes suivantes :

```
Shell shell = new Shell(display);  
shell.open();
```

A noter :

- Ajout de `import org.eclipse.swt.widgets.Shell;`
- par défaut, la fenêtre est automatiquement « disposée » lors de sa fermeture (par un clic sur la case de fermeture par exemple). Donc, la méthode `dispose()` n'est pas indispensable ici.

3

Le code

- **Création d'un projet SWT – méthode 1**

L'application ouvre une fenêtre qui se referme immédiatement.

Il faut maintenant ajouter **la boucle événementielle**, une boucle qui tourne en attendant un événement (exemple : action de l'utilisateur).

Une fois un événement reçu, celui-ci est traité par l'application.

La boucle ne devrait se terminer que quand l'utilisateur a décidé de quitter l'application.

3

Le code

- **Création d'un projet SWT – méthode 1**

Méthodes utilisées dans la boucle (toutes issues de **org.eclipse.swt.widgets.***) :

- **Shell.isDisposed()** : tourne jusqu'à ce que l'utilisateur ferme la fenêtre principale
- **Display.readAndDispatch()** : traite tous les événements en attente.
- **Display.sleep()** : s'il n'y a aucun événement en attente de traitement, l'application va dormir jusqu'à ce qu'un événement se présente.

3

Le code

- **Création d'un projet SWT – méthode 1**

```
while (!shell.isDisposed()) {  
    if (!display.readAndDispatch())  
        display.sleep();  
}
```

Lancez le programme : la fenêtre s'ouvre et reste affichée tant que l'utilisateur ne clique pas sur sa case de fermeture.

Dès la fenêtre fermée, le programme se termine.

3

Le code

- **Création d'un projet SWT – méthode 1**

Ajout d'un contenu dans la fenêtre

Les éléments d'interface utilisateur qui apparaissent dans une fenêtre sont matérialisés sous la forme de **widgets** (boutons, textes, menus, etc...).

Ici nous allons créer un **texte éditable** sous la forme d'un widget de la classe **org.eclipse.swt.widgets.Text**.

3

Le code

- **Création d'un projet SWT – méthode 1**

Créer un widget se fait en deux étapes:

1) **Instanciation de la classe** avec les options désirées (variant d'un widget à l'autre) permettant de modifier l'apparence ou le comportement.

2) **Affectation** au widget des caractéristiques désirées.

```
Text text = new Text(shell, SWT.CENTER);  
{  
    text.setText("Hello");  
    text.pack();  
}
```

3

Le code

La méthode **pack()**
recalcule la taille
optimale du composant

• Création d'un projet SWT – méthode 1

```
package main;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

public class Main {

    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);

        Text text = new Text(shell, SWT.CENTER);
        text.setText("Hello World");
        text.pack();

        shell.pack();
        shell.open();

        // Boucle événementielle
        while(!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```



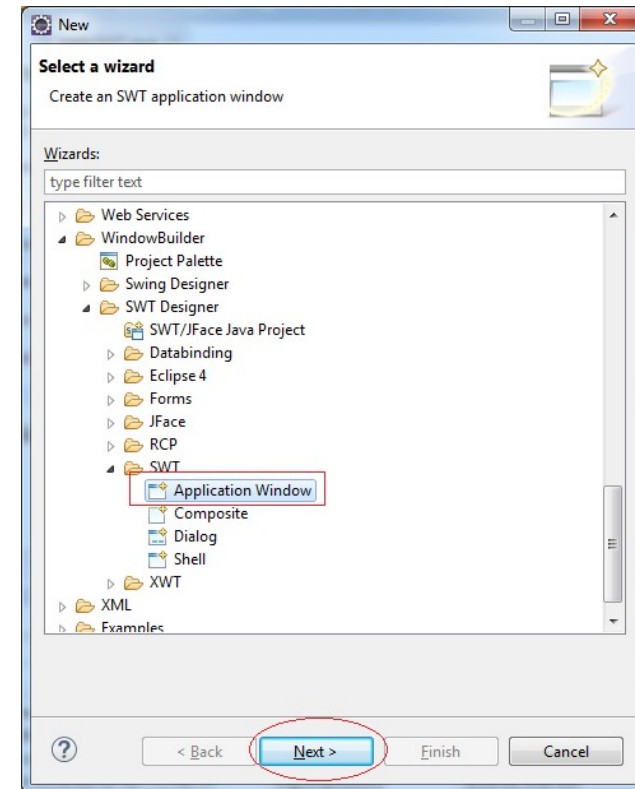
4

WindowBuilder

- **Création d'un projet – méthode 2**

Dans le projet, ajout d'une application Window.

File>New>Other

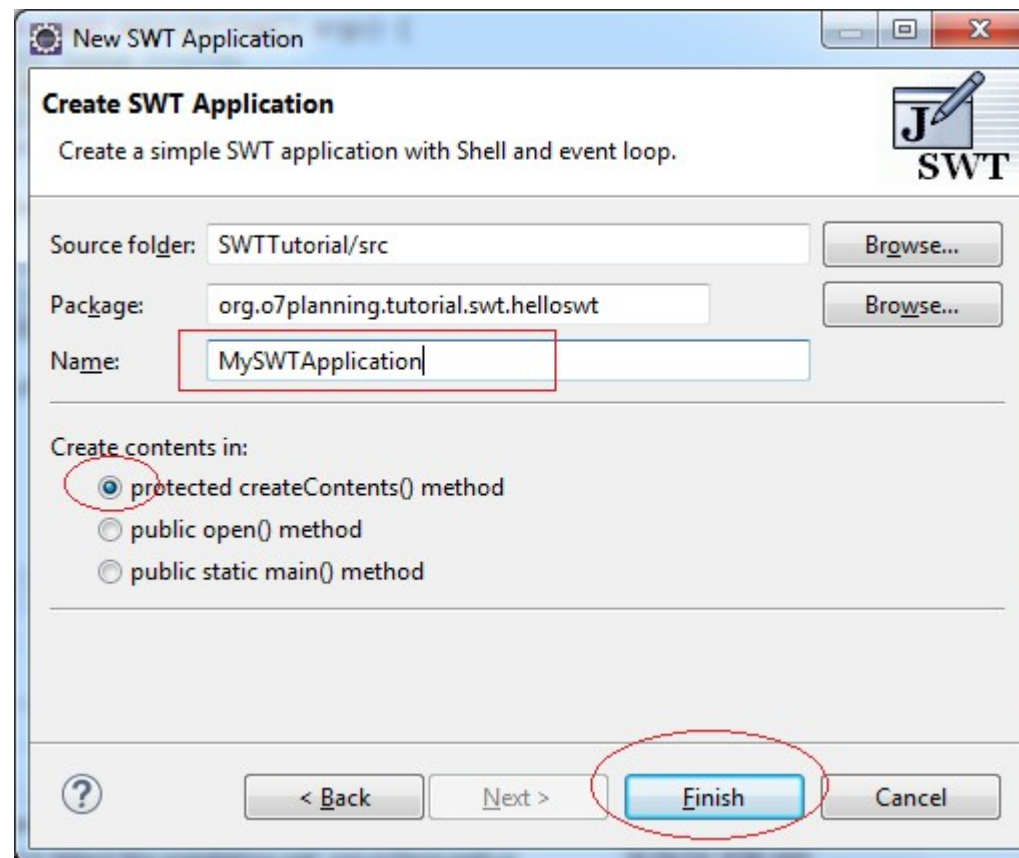


Choisir « Application Window », puis Next

4

WindowBuilder

- **Création d'un projet – méthode 2**



4

WindowBuilder

• Les widgets SWT – méthode 2

Voici quelques uns des widgets SWT



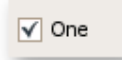
Browser

[javadoc - snippets](#)



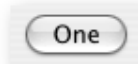
Button (SWT.ARROW)

[javadoc - snippets](#)



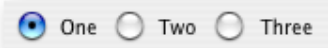
Button (SWT.CHECK)

[javadoc - snippets](#)



Button (SWT.PUSH)

[javadoc - snippets](#)



Button (SWT.RADIO)

[javadoc - snippets](#)



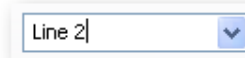
Button (SWT.TOGGLE)

[javadoc - snippets](#)



Canvas

[javadoc - snippets](#)



Combo

[javadoc - snippets](#)



Composite

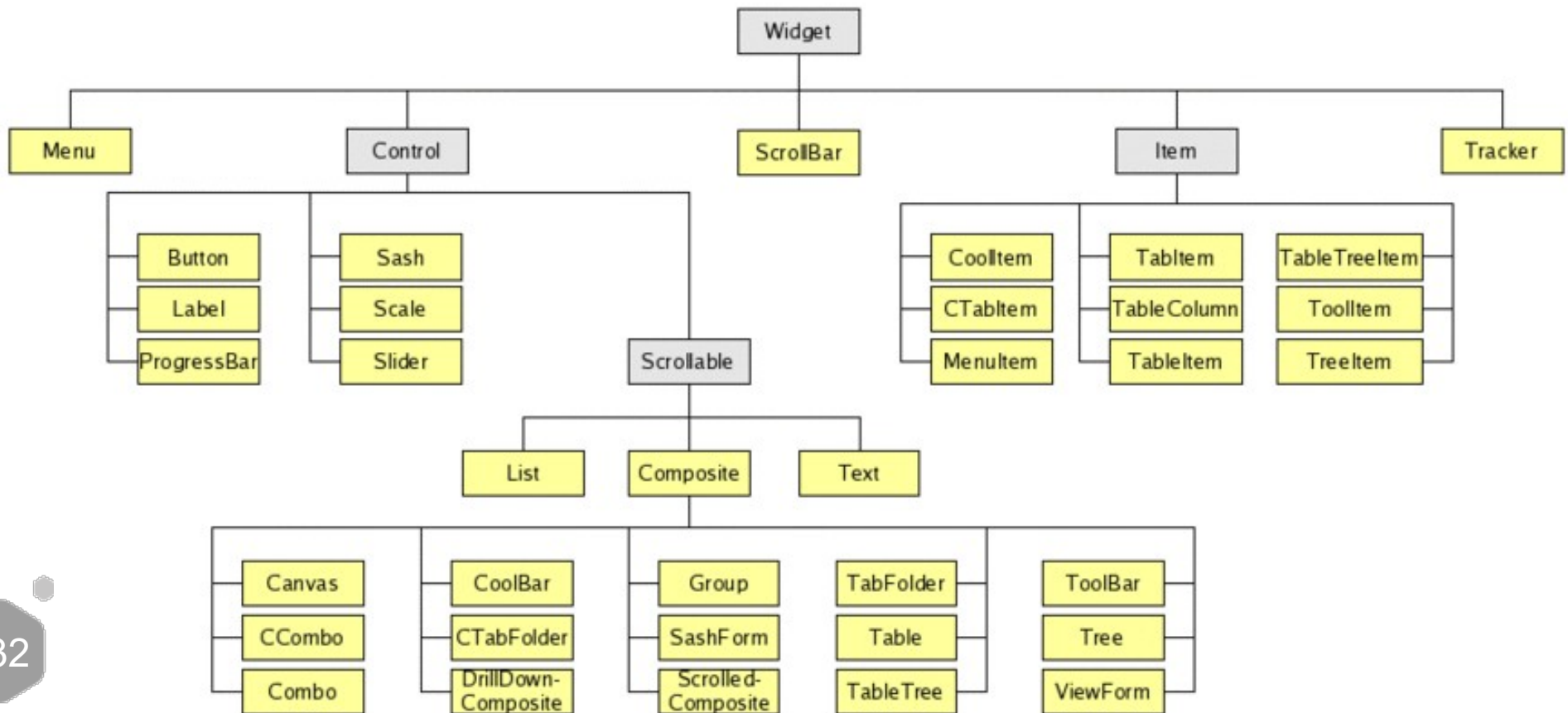
[javadoc - snippets](#)

4

WindowBuilder

• Les widgets SWT – méthode 2

Hiérarchie des widgets SWT

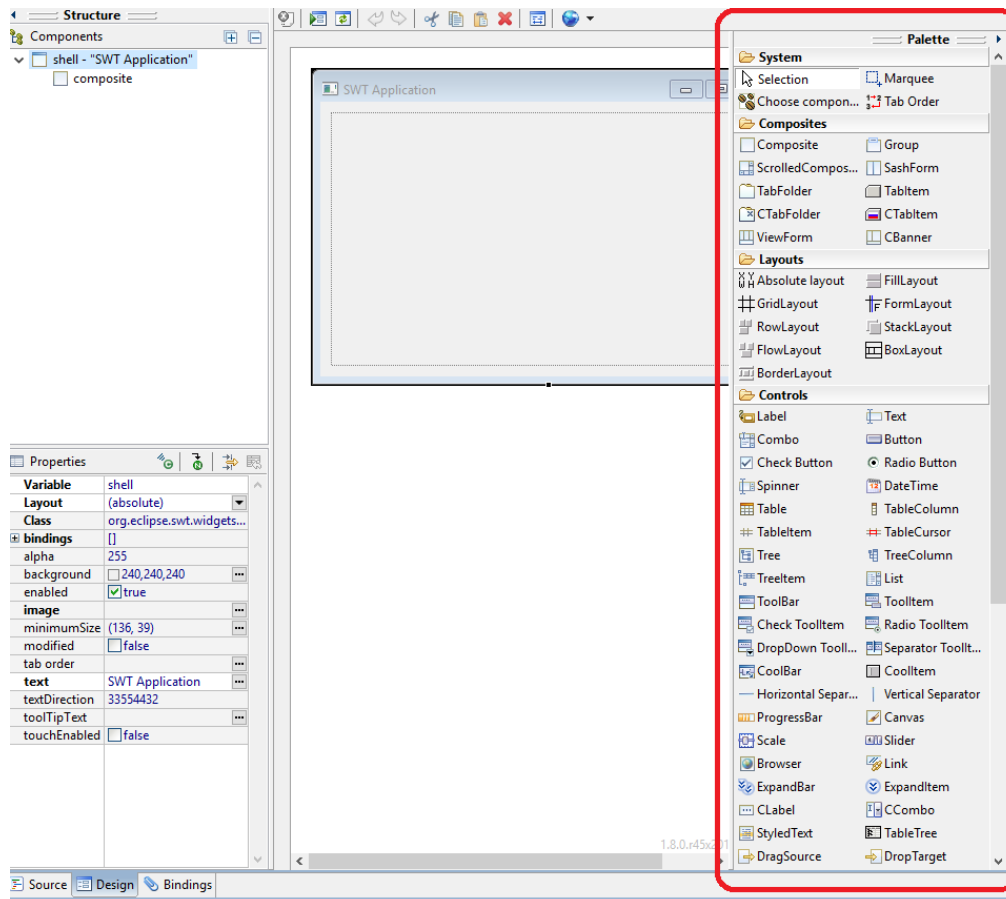


4

WindowBuilder

• Les widgets SWT – méthode 2

Voyons plus en détail les widgets disponibles

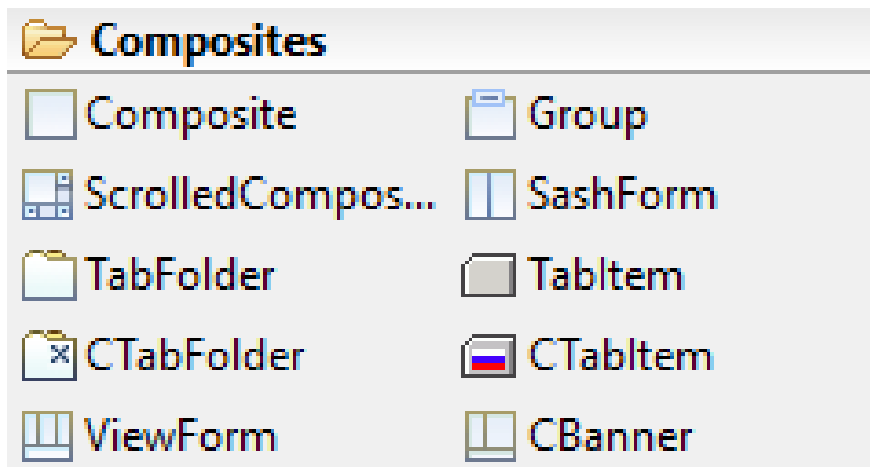


4

WindowBuilder

- **Les widgets SWT – méthode 2**

Certains widgets contiennent d'autres widgets :
ce sont des conteneurs (container)

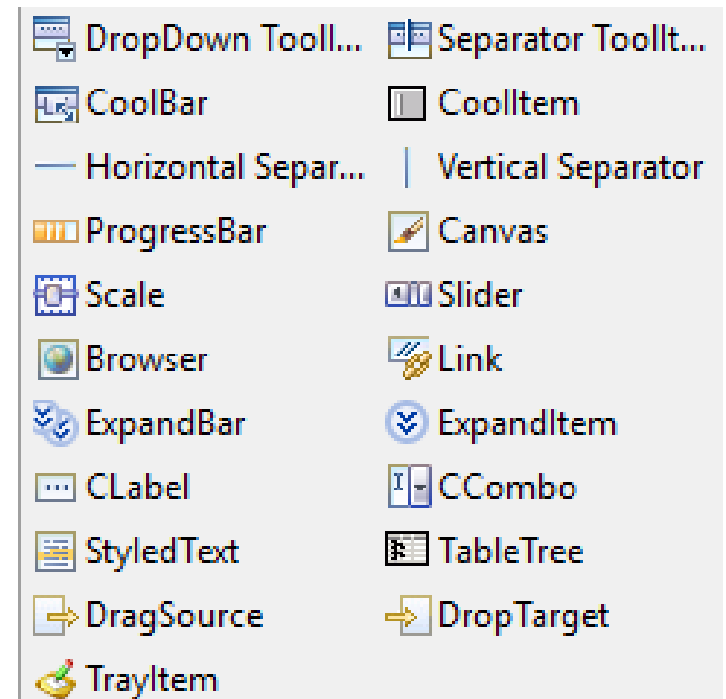
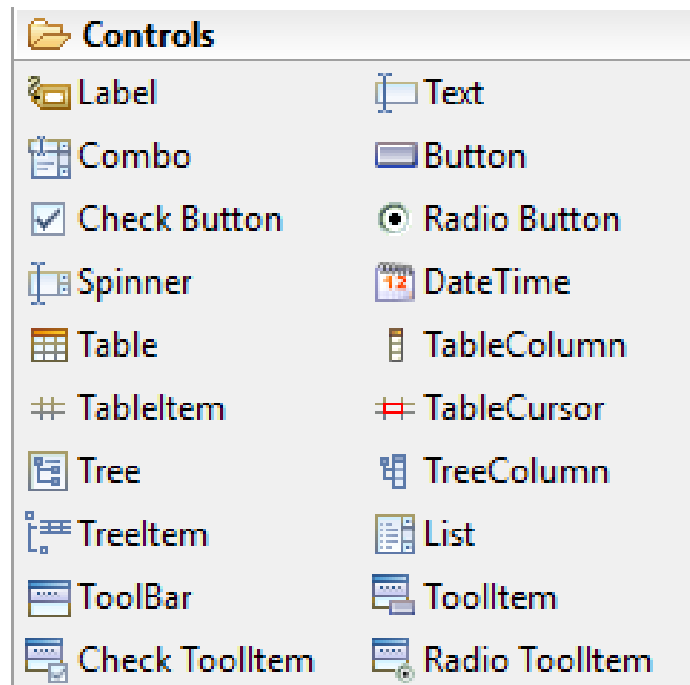


4

WindowBuilder

• Les widgets SWT – méthode 2

Widgets de contrôle



4

WindowBuilder

• **Les widgets SWT – méthode 2**

Voir la démo des contrôles sur le site
<http://rap.eclipsesource.com/demo/release/controls>

RAP Controls Demo

Button

Browser
Canvas
CBanner
CLabel
Combo
Composite
CoolBar
CTabFolder
DateTime
Dialogs
DropDown
ExpandBar
Focus
Group
Label

Button

Push Button
Toggle
☐ Check
☐ Check with image
☐ Radio 1
☐ Radio 2
☐ Radio 3
▲

Default Button

Enter some text and press Return

Default Button

Styles and Parameters

☐ BORDER
☐ FLAT
☐ LEFT
☐ CENTER
☐ RIGHT
☐ UP
...
☐ DOWN
☐ WRAP
☒ Visible
☒ Enabled
☐ Push Button with image
☐ Push Button with markup
☐ Grayed Check Buttons

Background
☐ Background Image
Font
Cursor: null
Badge:
☐ Add Selection List
Toggle Button
Foreground

5

Layout

- **Les layout**

Le layout (mise en page) permet d'organiser les composants sur l'interface

Exemple :



5

Layout

- **Les layout**

Les layouts standards de SWT sont :

- **FillLayout** : positionne des widgets de taille égale dans une seule ligne ou colonne
- **RowLayout** : positionne des widgets dans une ou plusieurs lignes.
- **GridLayout** : positionne des widgets dans une grille

5

Layout

- **Les layout - FillLayout**

FillLayout est la classe de mise en page la plus simple.

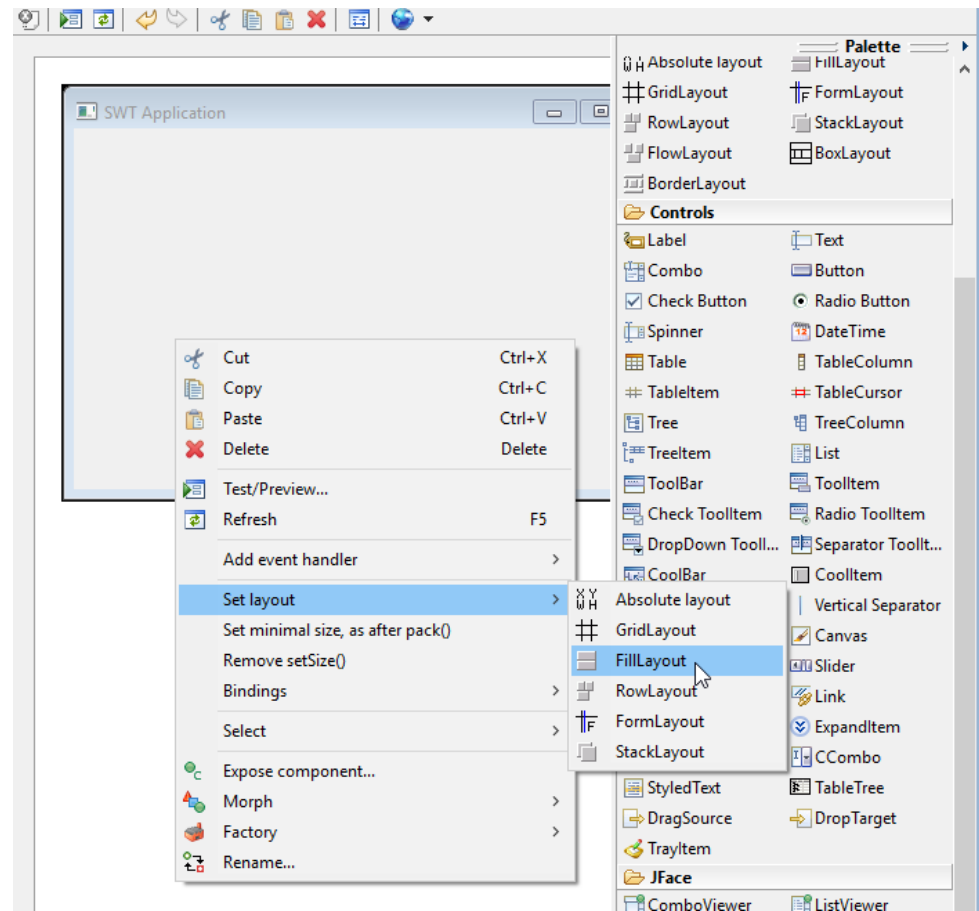
Elle dispose les widgets sur une seule ligne ou colonne en les forçant à être la même taille (en se calant sur le grand et le plus large)

5

Layout

- **Les layout - FillLayout**

Faire un clic-droit sur la fenêtre > Set Layout > FillLayout

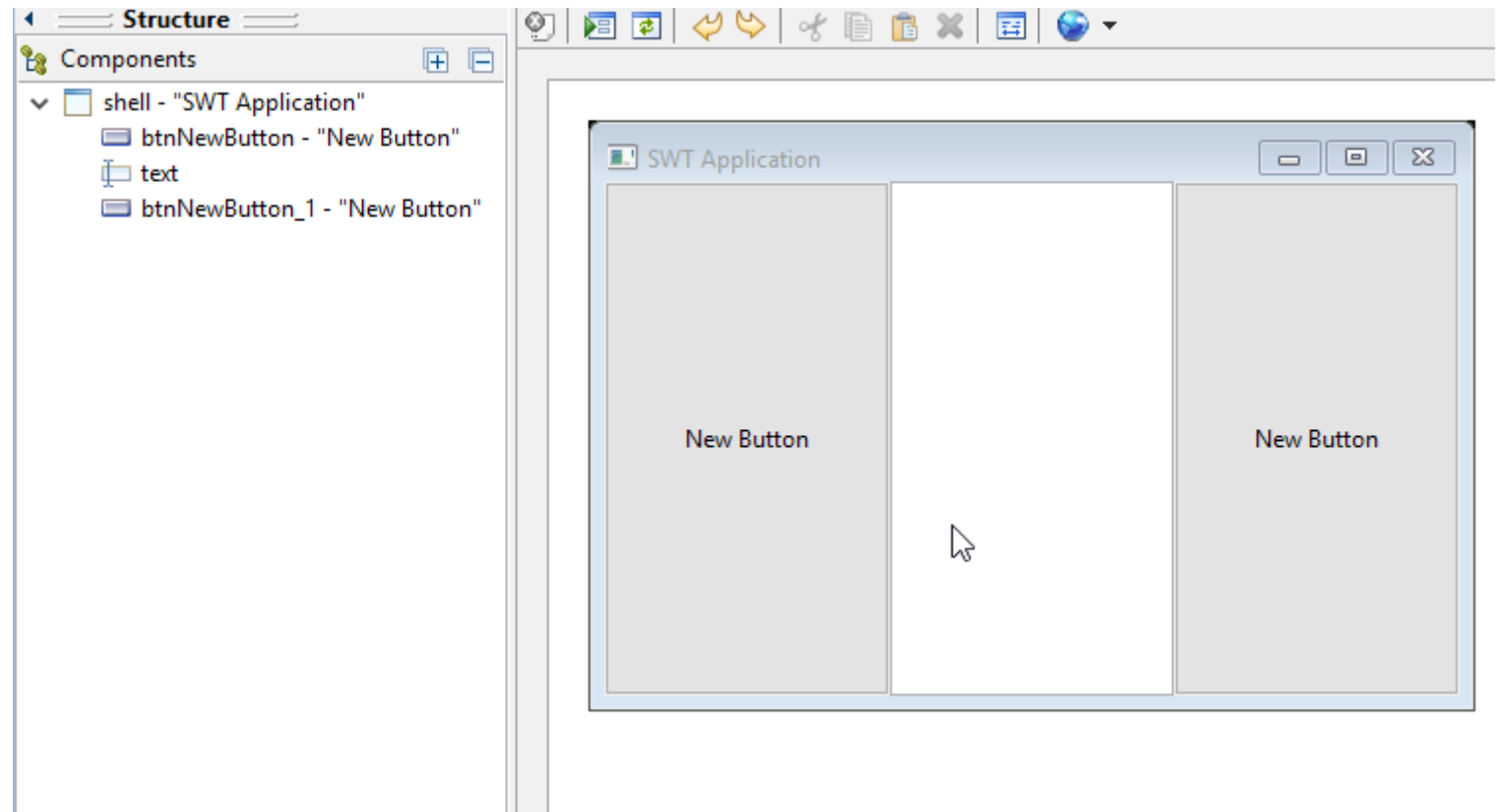


5

Layout

- **Les layout - FillLayout**

Ajouter des éléments, qui se disposeront de manière proportionnelle dans le layout.

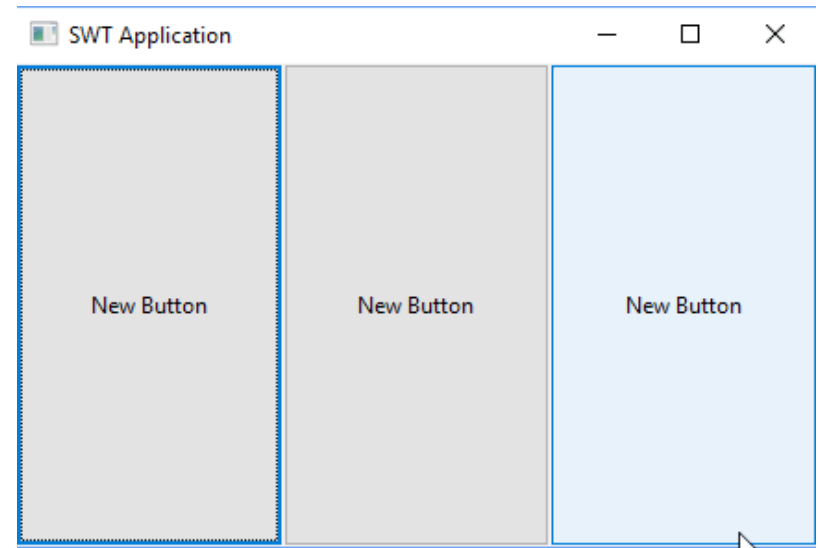
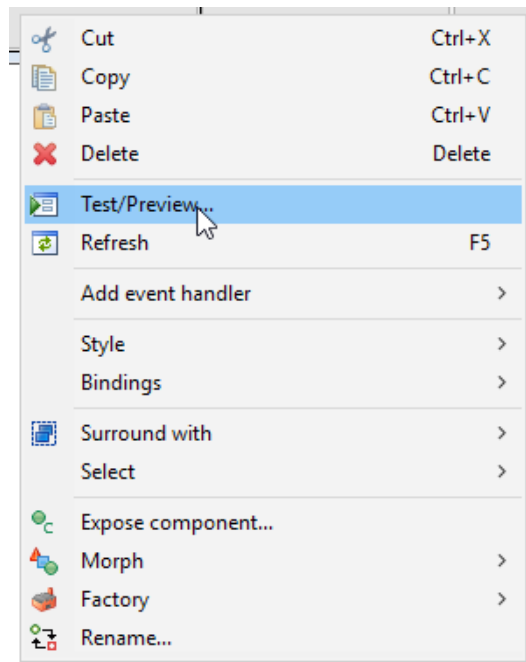


5

Layout

- **Les layout - FillLayout**

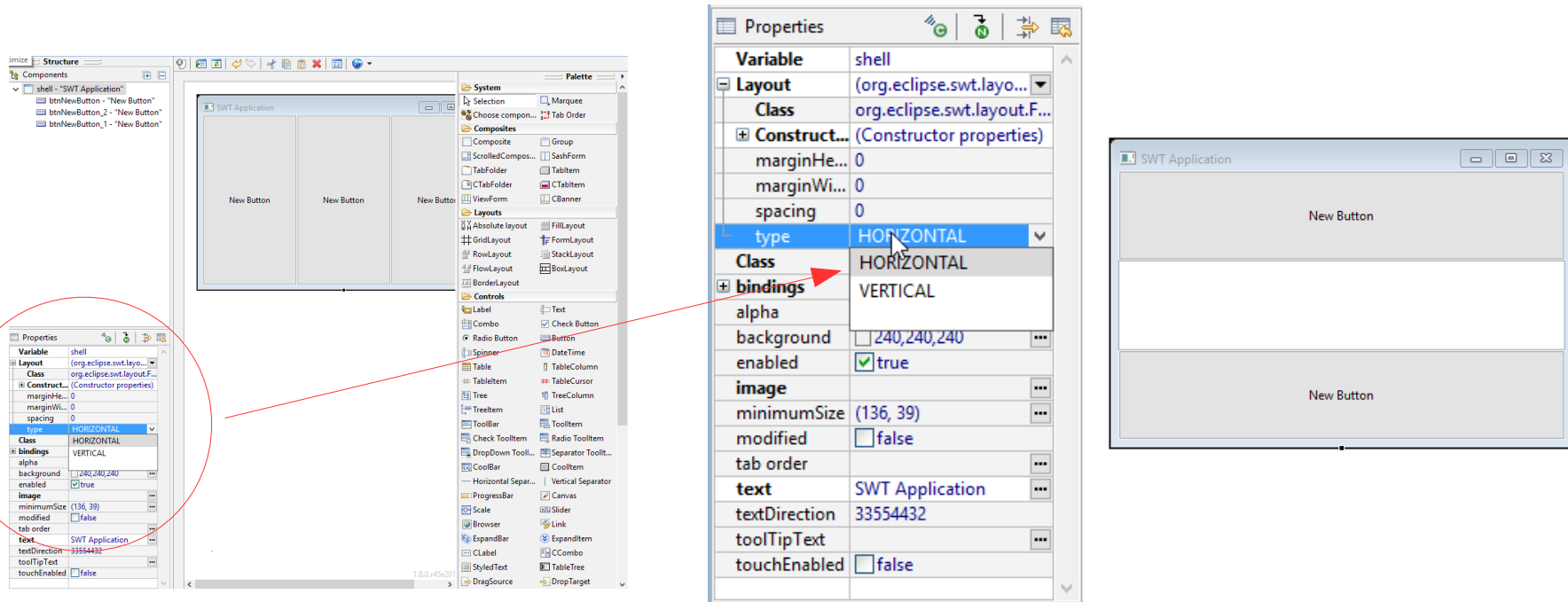
Pour voir le résultat, clic-droit > Text/Preview



5
Layout

• **Les layout - FillLayout**

Pour passer d'un alignement horizontal à vertical, aller dans les **propriétés** et déplier Layout pour voir le type à modifier



The screenshot shows the Eclipse IDE with the Properties window open for a shell widget. The 'Layout' property is expanded, showing 'type' set to 'HORIZONTAL'. A red arrow points to the 'type' dropdown menu, which is open, showing 'HORIZONTAL' and 'VERTICAL' options. The 'bindings' section is also visible, showing 'alpha' set to 0, 'background' set to (240,240,240), 'enabled' checked, 'image' set to SWT Application, 'minimumSize' set to (136, 39), 'modified' unchecked, 'tab order' set to 33554432, 'text' set to SWT Application, 'textDirection' set to 33554432, 'toolTipText' set to false, and 'touchEnabled' unchecked.

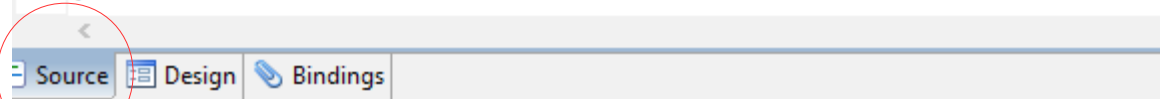
5

Layout

- **Les layout - FillLayout**

Lorsqu'on clique sur « source », on voit le code Java généré par les précédentes opérations.

```
45
46- /**
47   * Create contents of the window.
48   */
49- protected void createContents() {
50     shell = new Shell();
51     shell.setSize(450, 300);
52     shell.setText("SWT Application");
53     shell.setLayout(new FillLayout(SWT.VERTICAL));
54
55     Button btnNewButton = new Button(shell, SWT.NONE);
56     btnNewButton.setText("New Button");
57
58     text = new Text(shell, SWT.BORDER);
59
60     Button btnNewButton_1 = new Button(shell, SWT.NONE);
61     btnNewButton_1.setText("New Button");
62
63 }
64 }
```



5

Layout

- **Les layout - RowLayout**

RowLayout est offre d'avantage de souplesse dans la mise en page.

Il dispose des champs de configuration :









- Wrap
- Pack
- Justify

Ce sont des booléens.

5

Layout

• **Les layout - RowLayout**

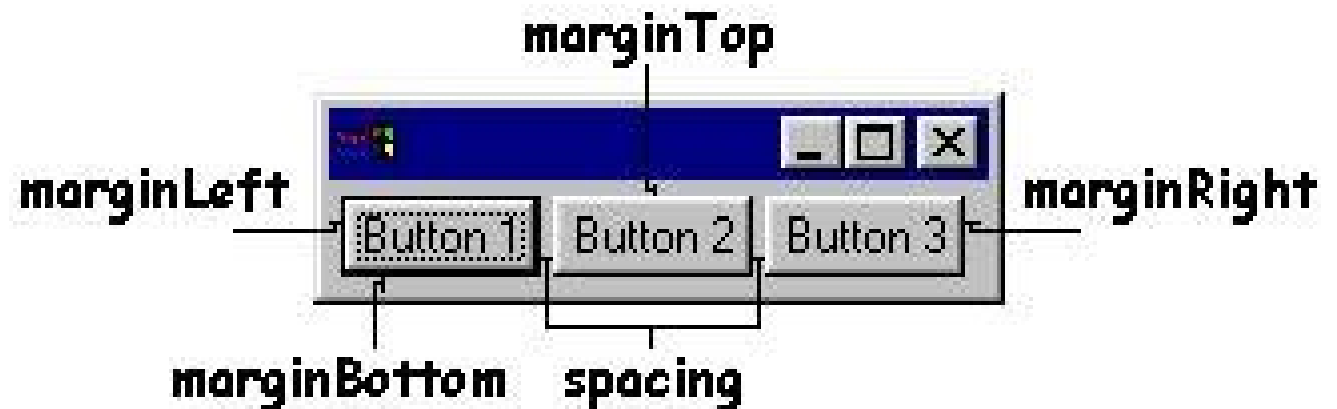
	Initial	Après redimensionnement
<p>wrap = true pack = true justify = false</p> <p>(par défaut)</p>		
<p>wrap = false (coupé si pas assez d'espace)</p>		
<p>pack = false (tous les widgets sont de même taille)</p>		
<p>justify = true (les widgets sont répartis sur l'espace disponible)</p>		

5

Layout

- **Les layout - RowLayout**

Paramètres de marges:



5

Layout

- **Les layout - GridLayout**

GridLayout est le plus utile et puissant des layout standard. C'est aussi le plus compliqué.

Le principe d'un GridLayout est que les widgets enfants d'un composite sont disposés dans une grille.

numColumns = 1



numColumns = 2



numColumns = 3



5

Layout

- **Les layout - GridLayout**

GridLayout a un certain nombre de champs de configuration et, comme RowLayout, les widgets qu'il dispose peuvent avoir un objet de données de mise en page associée, appelée `gridData`.

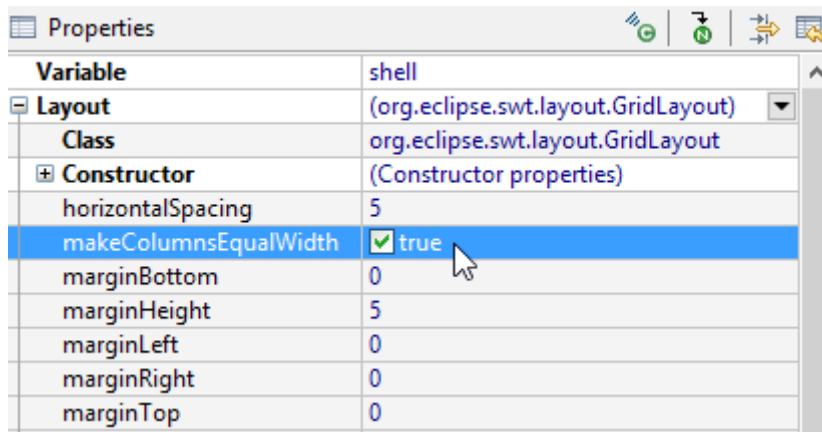
La puissance de GridLayout réside dans la possibilité de configurer `gridData` pour chaque widget de contrôle par le GridLayout.

5

Layout

- **Les layout - GridLayout**

Le paramètre ***MakeColumnsEqualWidth*** permet d'affecter la même taille à chaque colonne.



Chaque colonne se cale sur la largeur de l'élément le plus large (ici « Wide Button 2 »)

5

Layout

- **Les layout - StackLayout**

Ce layout empile toutes les widgets les uns sur les autres et redimensionne tous les contrôles pour qu'ils aient la même taille et le même emplacement.

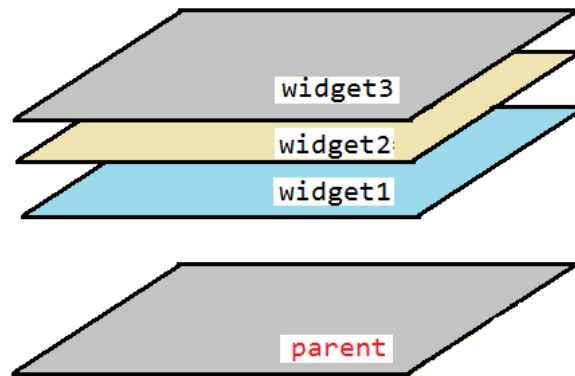
Seul le contrôle spécifié dans TopControl est visible.

5

Layout

- **Les layout - StackLayout**

Fonctionnement d'un StackLayout (empilement de widgets)



```
Composite parent= ... ;

StackLayout layout= new StackLayout();
parent.setLayout(layout);

Button widget1 = new Button(parent, SWT.NONE);
widget1.setText("Button 1");

Composite widget2= new Composite(parent, SWT.BORDER);

Button widget3 = new Button(parent, SWT.NONE);
widget3.setText("Button 3");

layout.topControl= widget3;
parent.layout();
```

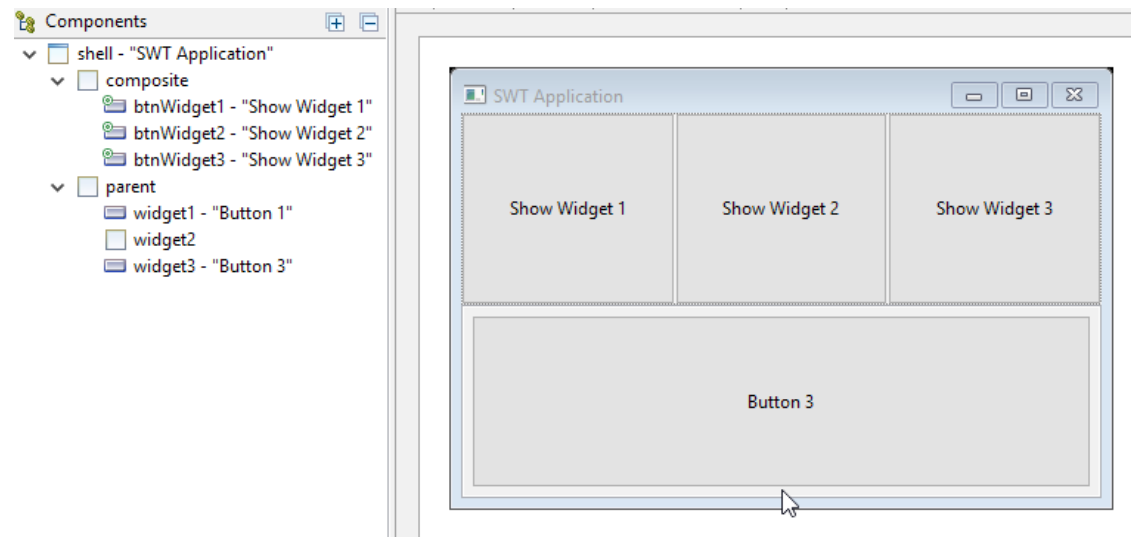
5

Layout

- **Les layout - StackLayout**

Sur cet exemple, on a une fenêtre en 2 parties.

- La 1ère avec un **FillLayout** contenant 3 boutons
- La 2de avec un **StackLayout** contenant 3 widgets (2 boutons et un Composite)



5

Layout

- **Les layout - StackLayout**

L'idée est qu'en cliquant sur un bouton dans la partie supérieur, on affiche un des widget dans la partie inférieure.

Se positionner sur le bouton, faire un clic droit
> Add event Handler > Selection >
WidgetSelected

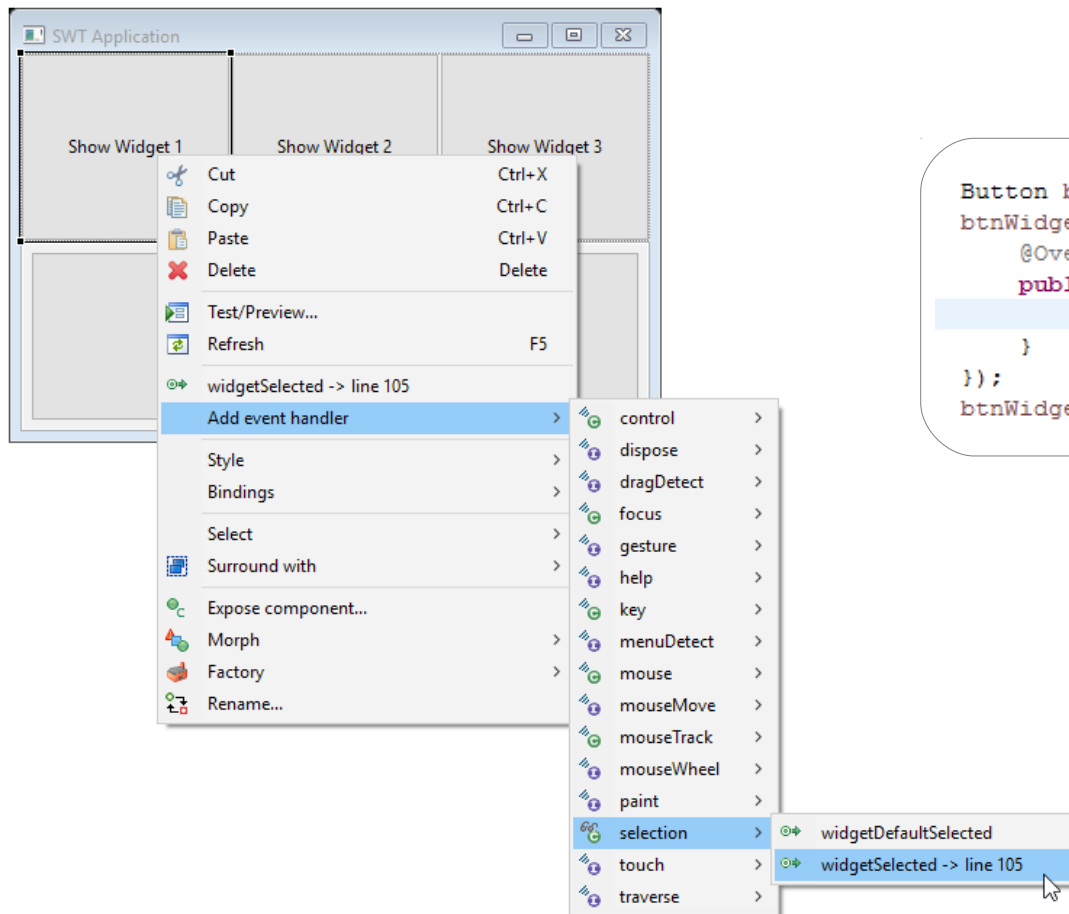
On se retrouve alors dans la partie code.

5

Layout

• Les layout - StackLayout

Il faut alors ajouter la méthode que l'on veut appeler.



```
Button btnWidget1 = new Button(composite, SWT.NONE);
btnWidget1.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        showWidget1();
    }
});
btnWidget1.setText("Show Widget 1");
```

5
Layout

• **Les layout - StackLayout**

```

63 Button btnWidget1 = new Button(composite, SWT.NONE);
64 btnWidget1.addSelectionListener(new SelectionAdapter() {
65     @Override
66     public void widgetSelected(SelectionEvent e) {
67         showWidget1();
68     }
69 });
70 btnWidget1.setText("Show Widget 1");
71
72 Button btnWidget2 = new Button(composite, SWT.NONE);
73 btnWidget2.addSelectionListener(new SelectionAdapter() {
74     @Override
75     public void widgetSelected(SelectionEvent e) {
76         showWidget2();
77     }
78 });
79 btnWidget2.setText("Show Widget 2");
80
81 Button btnWidget3 = new Button(composite, SWT.NONE);
82 btnWidget3.addSelectionListener(new SelectionAdapter() {
83     @Override
84     public void widgetSelected(SelectionEvent e) {
85         showWidget3();
86     }
87 });
88 btnWidget3.setText("Show Widget 3");
89
90 parent = new Composite(shell, SWT.BORDER);
91 StackLayout sl_parent = new StackLayout();
92 sl_parent.marginWidth = 5;
93 sl_parent.marginHeight = 5;
94 parent.setLayout(sl_parent);
95
96 widget1 = new Button(parent, SWT.NONE);
97 widget1.setText("Button 1");
98
99 widget2 = new Composite(parent, SWT.NONE);
100
101 widget3 = new Button(parent, SWT.NONE);
102 widget3.setText("Button 3");
103
104 }
105
106 private void showWidget1() {
107     StackLayout layout = (StackLayout) this.parent.getLayout();
108     layout.topControl = this.widget1;
109     this.parent.layout();
110 }

```

```

private void showWidget1() {
    StackLayout layout = (StackLayout) this.parent.getLayout();
    layout.topControl = this.widget1;
    this.parent.layout();
}

```


6

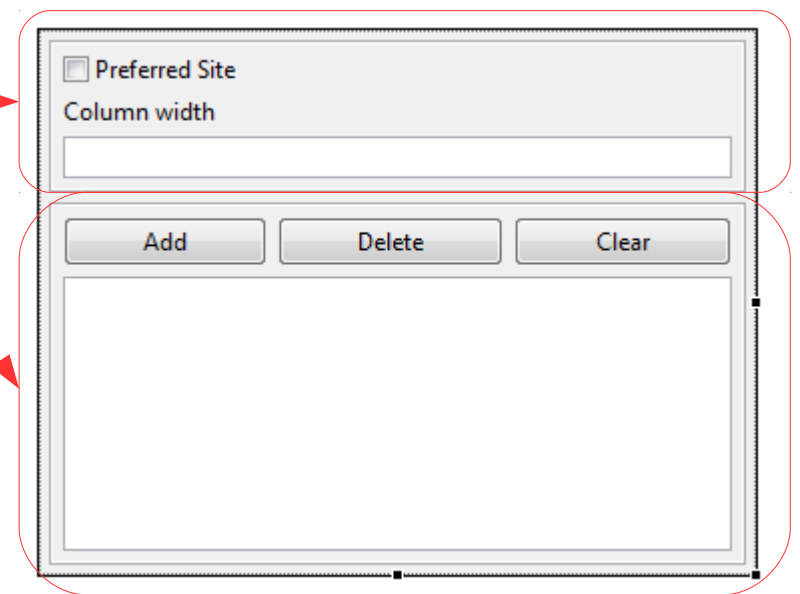
Modular

- **Interface avec composants modulaire**

La conception d'une interface complexe nécessite le fractionnement de ses éléments.

Prenons par exemple une interface composée de deux parties :

- TopComposite
- BottomComposite

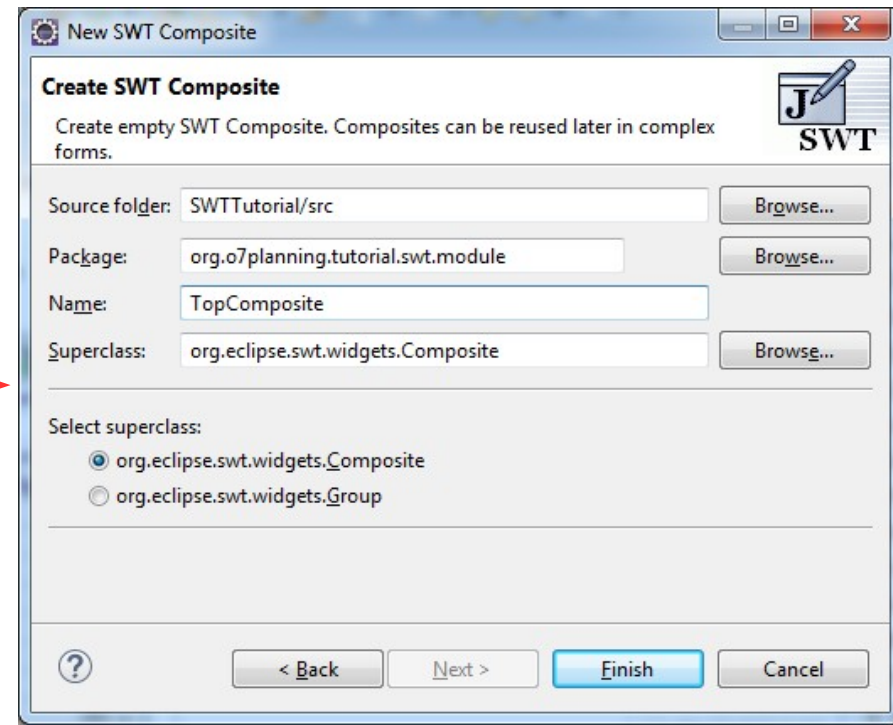
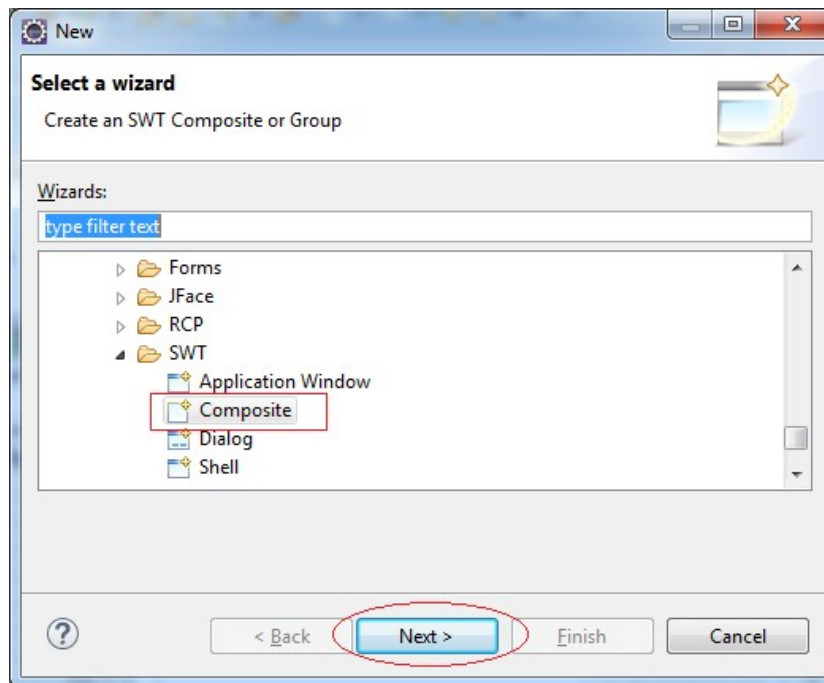


6

Modular

- **Interface avec composants modulaire**

Pour créer un composant, aller dans File > New > Other puis choisir dans SWT « Composite »

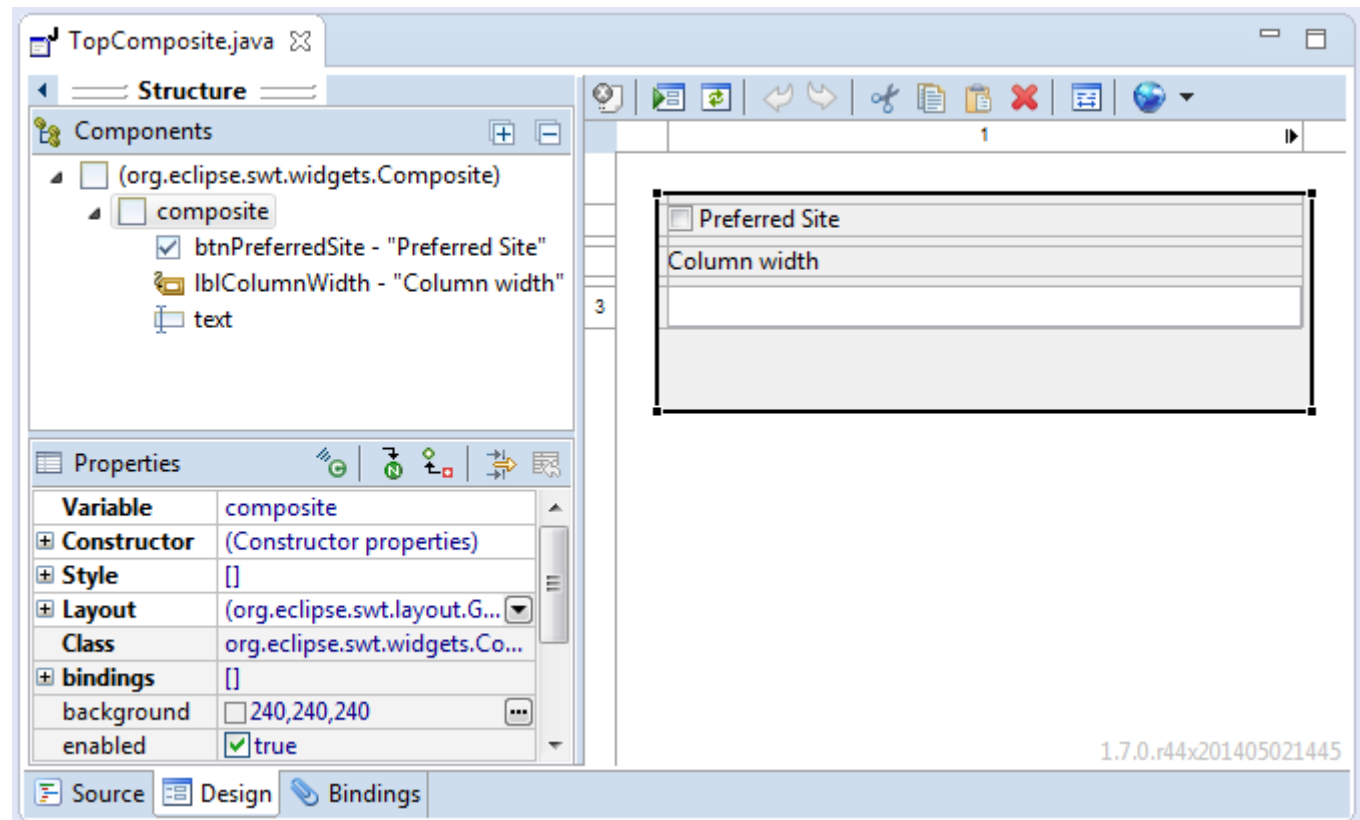


6

Modular

- Interface avec composants modulaire

Voici l'interface **TopComposite**



6

Modular

- Interface avec composants modulaire

... et le code Java correspondant

```
public class TopComposite extends Composite {
    private Text text;

    /**
     * Create the composite.
     * @param parent
     * @param style
     */
    public TopComposite(Composite parent, int style) {
        super(parent, style);
        setLayout(new FillLayout(SWT.HORIZONTAL));

        Composite composite = new Composite(this, SWT.NONE);
        composite.setLayout(new GridLayout(1, false));

        Button btnPreferredSite = new Button(composite, SWT.CHECK);
        btnPreferredSite.setText("Preferred Site");

        Label lblColumnWidth = new Label(composite, SWT.NONE);
        lblColumnWidth.setText("Column width");

        text = new Text(composite, SWT.BORDER);
        text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));
    }

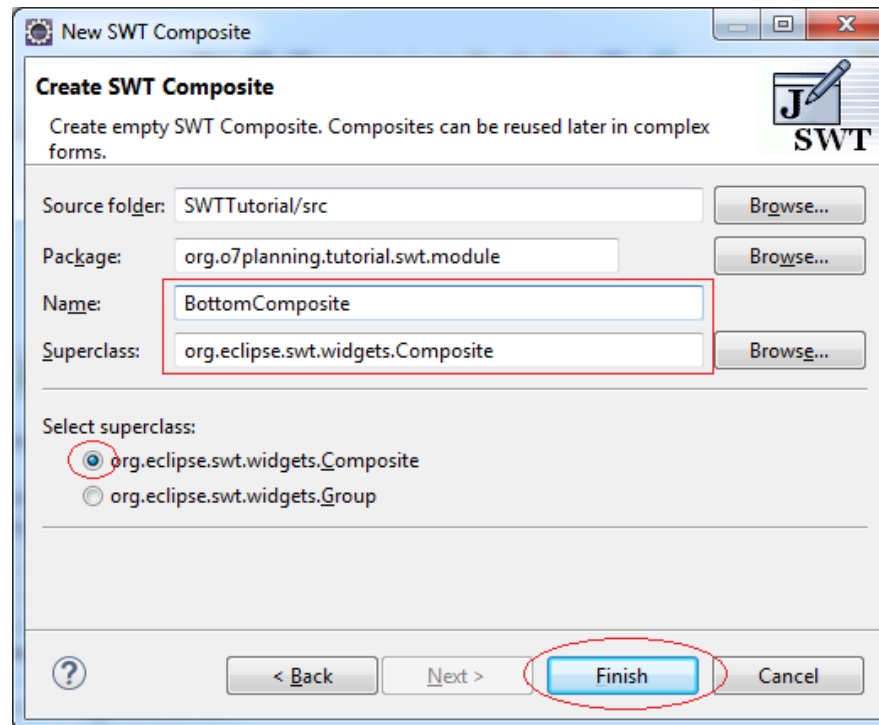
    @Override
    protected void checkSubclass() {
    }
}
```

6

Modular

- **Interface avec composants modulaire**

On crée de la même façon le composant **BottomComposite**



6 Modular

• Interface avec composants modulaire

```
public class BottomComposite extends Composite {
    private Text text;

    /**
     * Create the composite.
     * @param parent
     * @param style
     */
    public BottomComposite(Composite parent, int style) {
        super(parent, style);
        setLayout(new FillLayout(SWT.HORIZONTAL));

        Composite composite = new Composite(this, SWT.NONE);
        composite.setLayout(new GridLayout(1, false));

        Composite composite_1 = new Composite(composite, SWT.NONE);
        GridLayout gl_composite_1 = new GridLayout(3, false);
        gl_composite_1.marginHeight = 0;
        gl_composite_1.marginWidth = 0;
        composite_1.setLayout(gl_composite_1);
        composite_1.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));

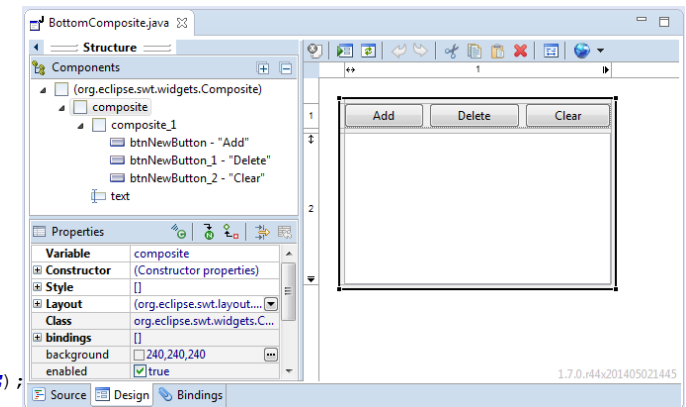
        Button btnNewButton = new Button(composite_1, SWT.NONE);
        btnNewButton.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));
        btnNewButton.setText("Add");

        Button btnNewButton_1 = new Button(composite_1, SWT.NONE);
        btnNewButton_1.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));
        btnNewButton_1.setText("Delete");

        Button btnNewButton_2 = new Button(composite_1, SWT.NONE);
        btnNewButton_2.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));
        btnNewButton_2.setText("Clear");

        text = new Text(composite, SWT.BORDER | SWT.MULTI);
        text.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1));
    }

    @Override
    protected void checkSubclass() {
        // Disable the check that prevents subclassing of SWT components
    }
}
```

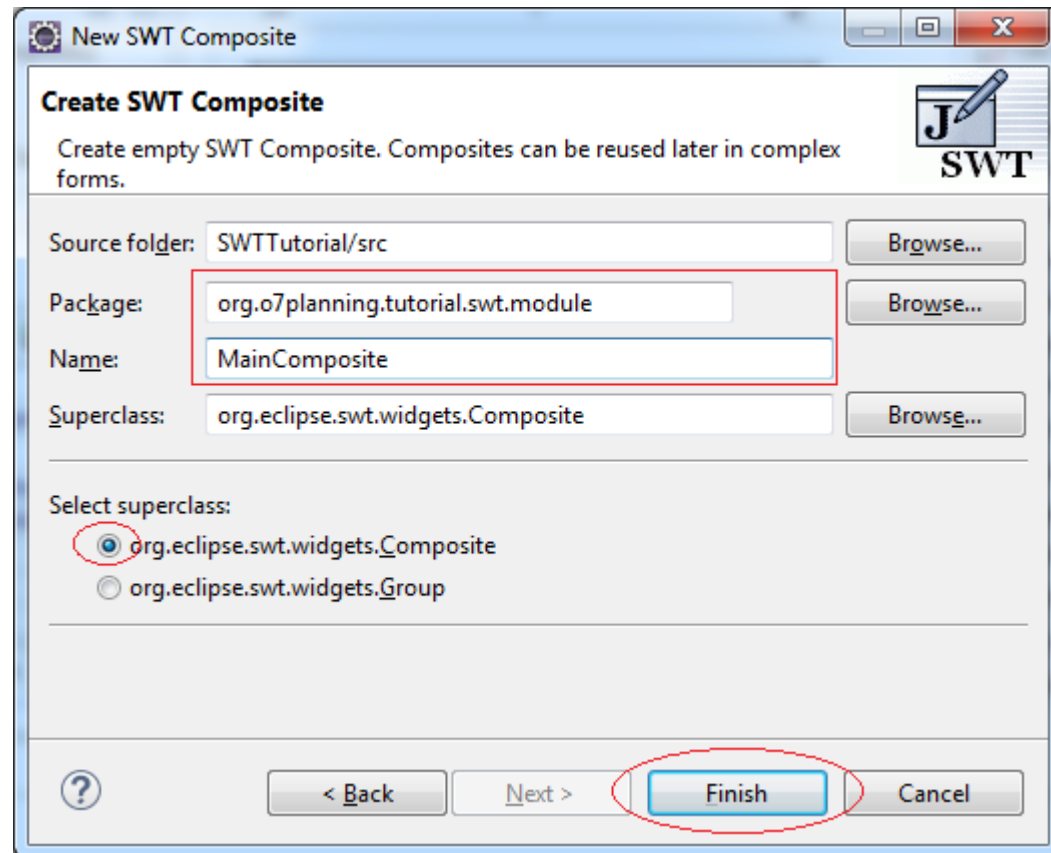


6

Modular

- **Interface avec composants modulaire**

Enfin, le **MainComposite** va rassembler ces deux éléments.

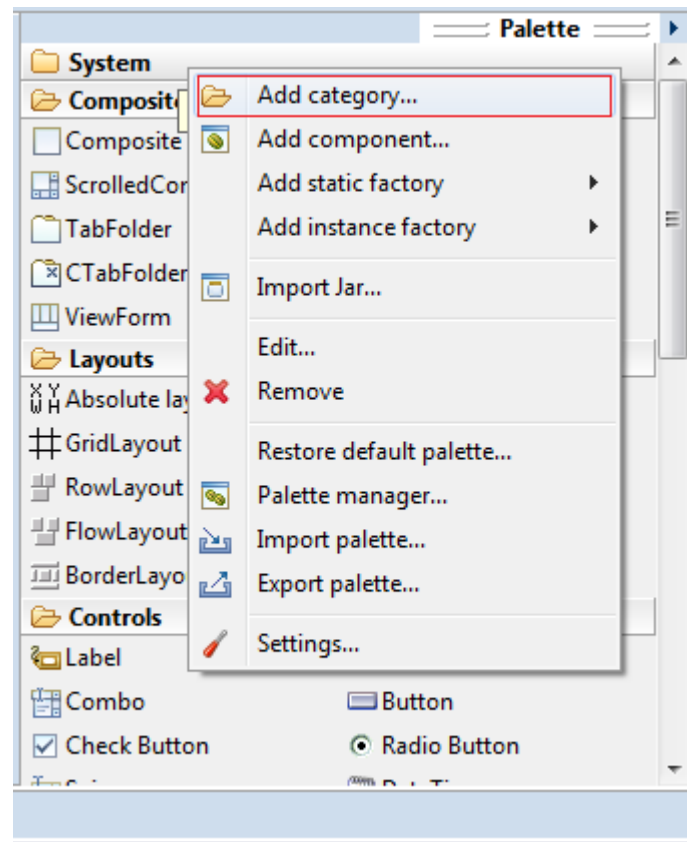


6

Modular

- **Interface avec composants modulaire**

Faire un clic-droit sur la palette et choisir Add category ..

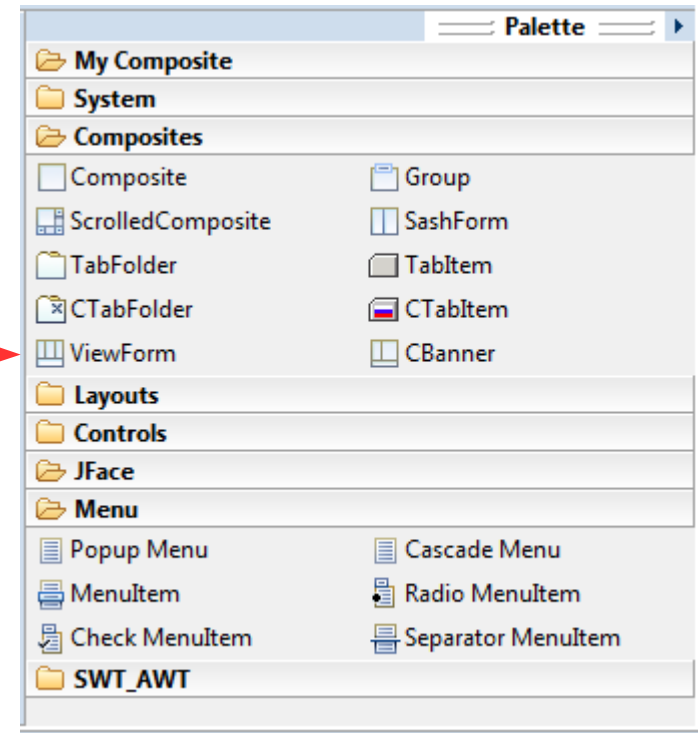
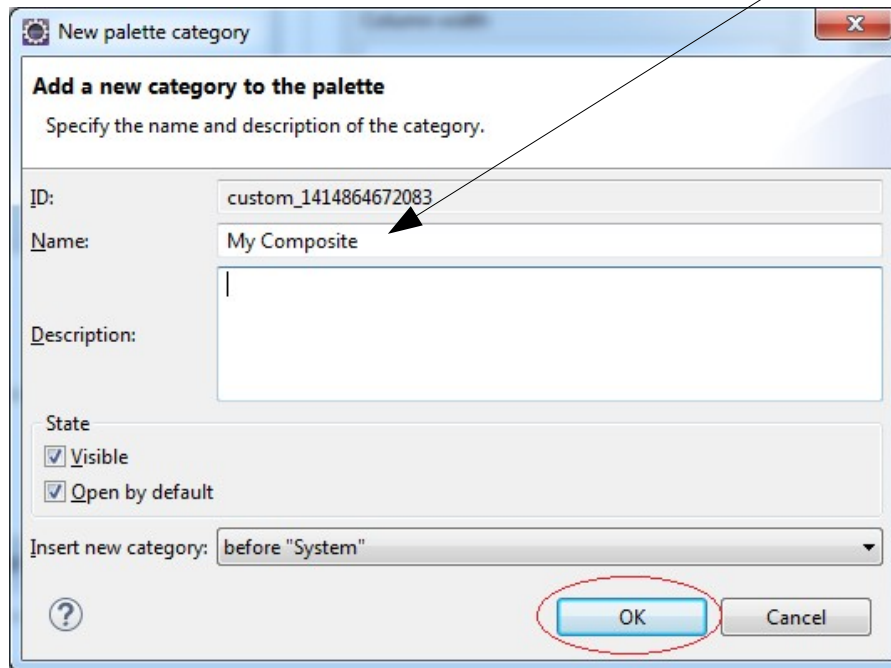


6

Modular

• Interface avec composants modulaire

Indiquez le nom « ***My Composite*** » et valider.

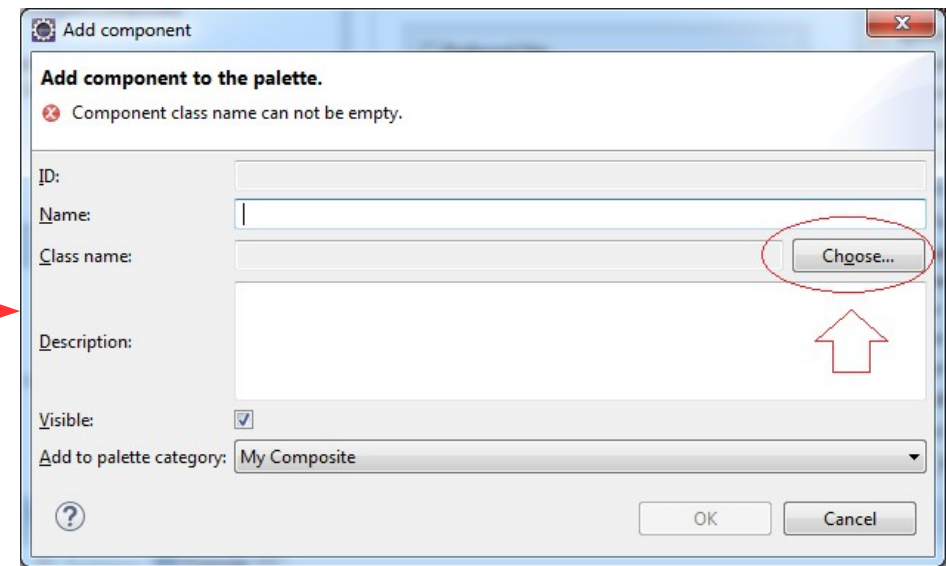
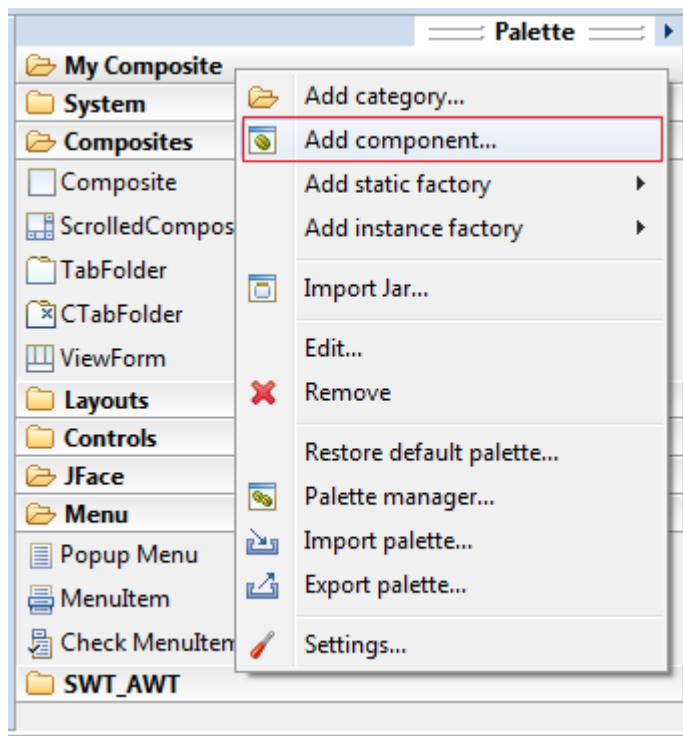


6

Modular

- **Interface avec composants modulaire**

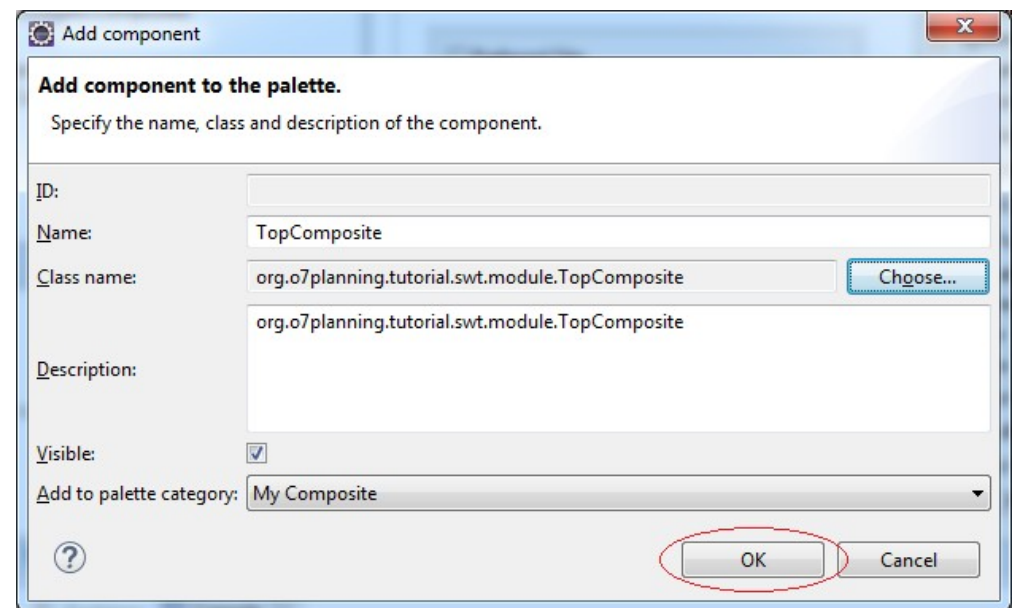
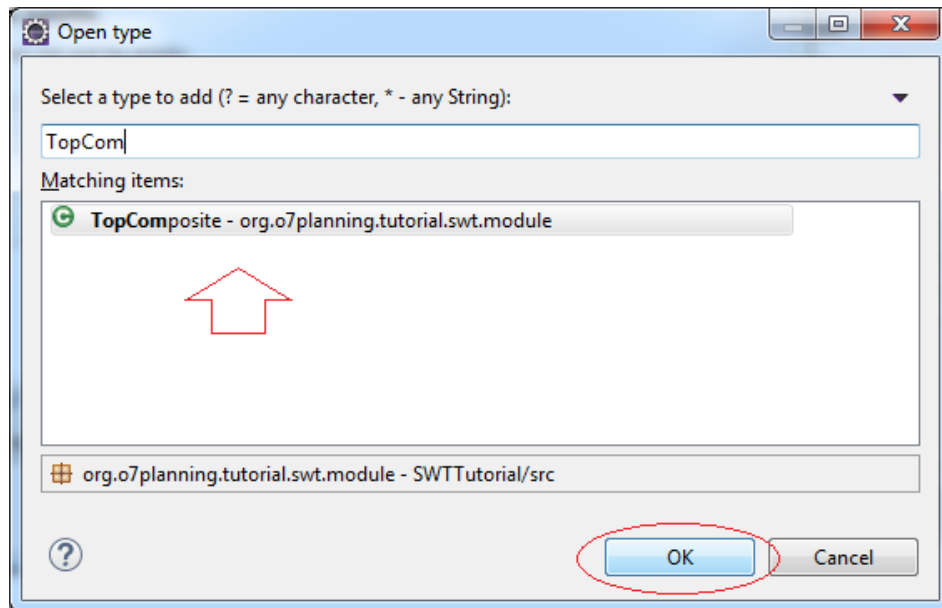
Faire un clic-droit sur **My Composite** et ajouter **TopComposite** et **BottomComposite**



6

Modular

- Interface avec composants modulaire

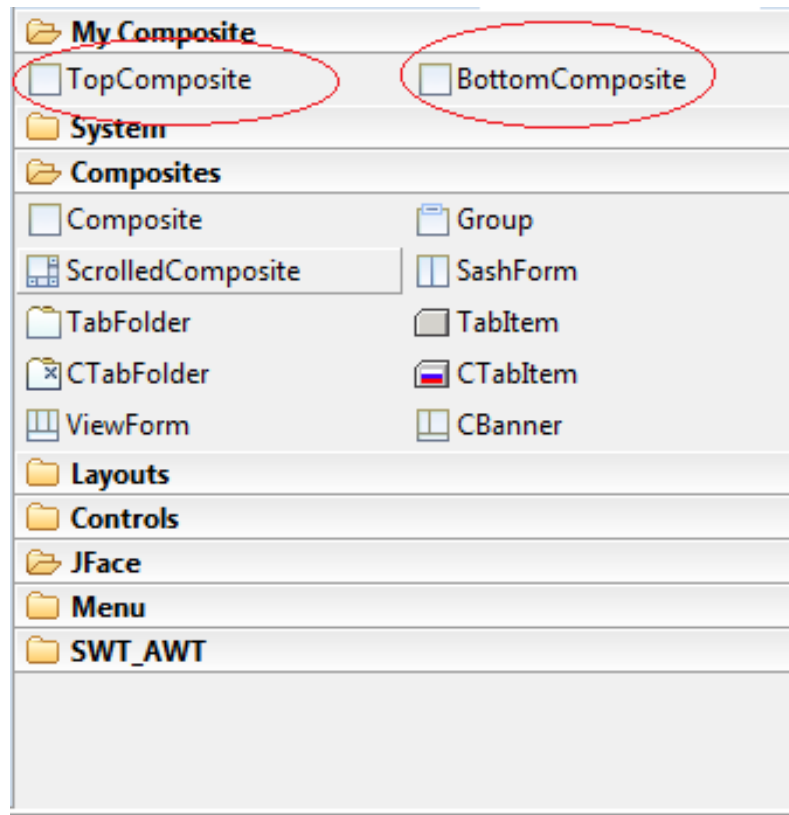


6

Modular

- **Interface avec composants modulaire**

Même chose pour BottomComposite. Au final, on a :

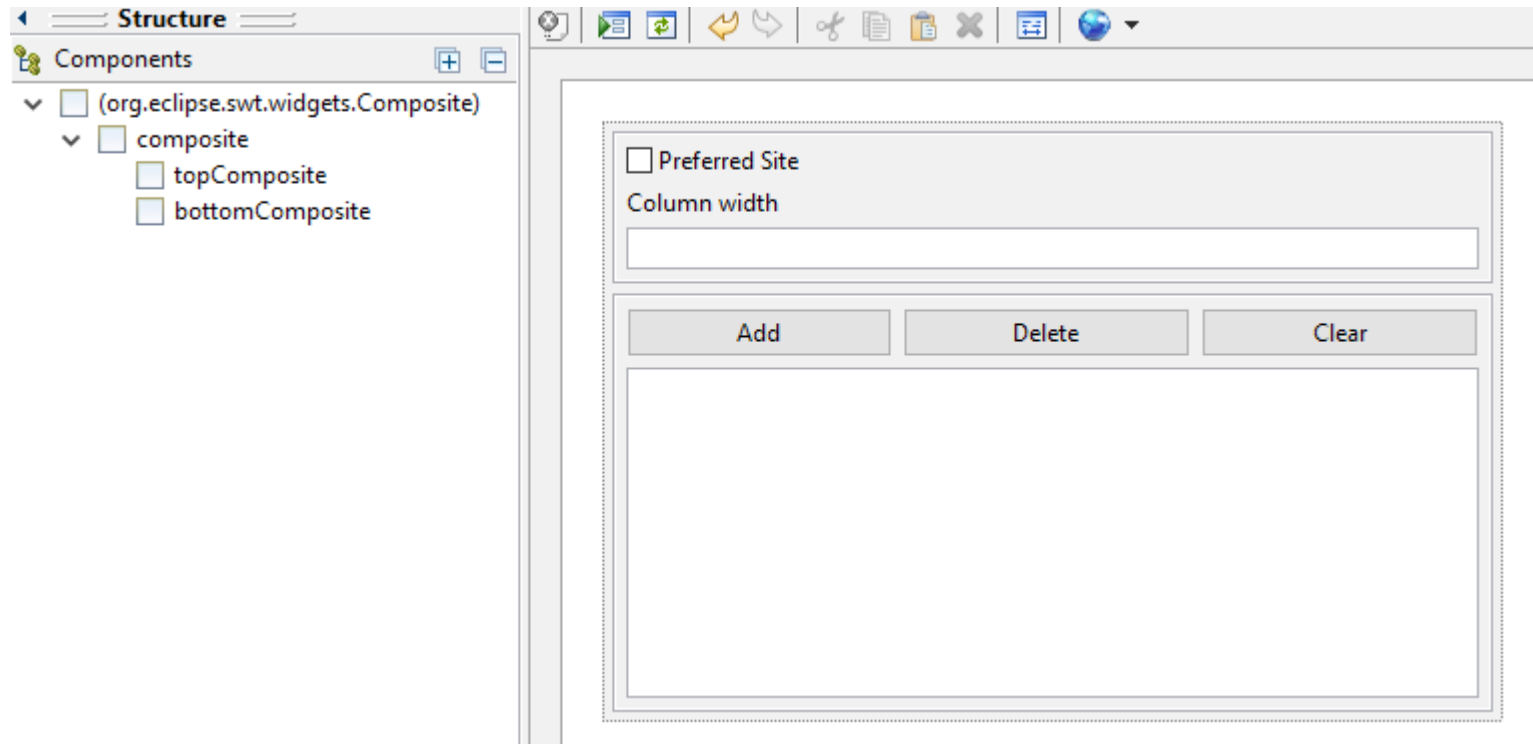


6

Modular

- **Interface avec composants modulaire**

Il suffit maintenant de glisser les composants pour les positionner dans l'interface.



6

Modular

- Interface avec composants modulaire

Voici le code :

```
public class MainComposite extends Composite {  
  
    /**  
     * Create the composite.  
     * @param parent  
     * @param style  
     */  
    public MainComposite(Composite parent, int style) {  
        super(parent, style);  
        setLayout(new FillLayout(SWT.HORIZONTAL));  
  
        Composite composite = new Composite(this, SWT.NONE);  
        composite.setLayout(new GridLayout(1, false));  
  
        TopComposite topComposite = new TopComposite(composite, SWT.BORDER);  
        topComposite.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));  
  
        BottomComposite bottomComposite = new BottomComposite(composite, SWT.BORDER);  
        bottomComposite.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1));  
  
    }  
  
    @Override  
    protected void checkSubclass() {  
    }  
}
```



Fin du tutoriel Java
