



## MENU



All Tutorials

Java

Maven

Gradle

Servlet/Jsp

Thymeleaf

Spring

Struts2

Hibernate

Java Web Service

JavaFX

SWT

Oracle ADF

Android




iOS

Python

Swift

C#



C/C++	
Ruby	
Dart	
Batch	
Database	
Oracle APEX	
Report	
Client	
ECMAScript / Javascript	
TypeScript	
NodeJS	
ReactJS	
Flutter	
AngularJS	
HTML	
CSS	
Bootstrap	
OS	
Git	
SAP	
Amazon AWS	 

Others

## Programming Java Desktop Application Using SWT

### View more Tutorials:

Eclipse Technology

Eclipse RCP

Java SWT Tutorials

1. Introduction
2. RCP (Rich Client Platform)
3. The settings required before starting
4. Some concepts of SWT.
  1. Display & Shell
  2. SWT Widgets
5. Create RCP Plugin Project
6. First Example
7. Using WindowBuilder
8. SWT Widget
  1. Overview



2. Widgets can contain other widgets (Container)
3. Controls
9. SWT Layout
  1. What is Layout?
  2. Online Example
  3. FillLayout
  4. RowLayout
  5. GridLayout
  6. StackLayout
  7. Combine Layout
10. Write the class extending from the SWT widget
11. Modular component interfaces
12. Using event handlers
13. JFace



Websites to learn foreign languages for free:

- English
- Vietnamese
- Other Languages



Follow us on our fanpages to receive notifications every time there are new articles. [Facebook](#) [Twitter](#)

## 1- Introduction

This document is based on:



- **Eclipse 4.6, 47 (NEON, OXYGEN)**

In this document, I will introduce you to the Desktop application programming with SWT.

//

See more:

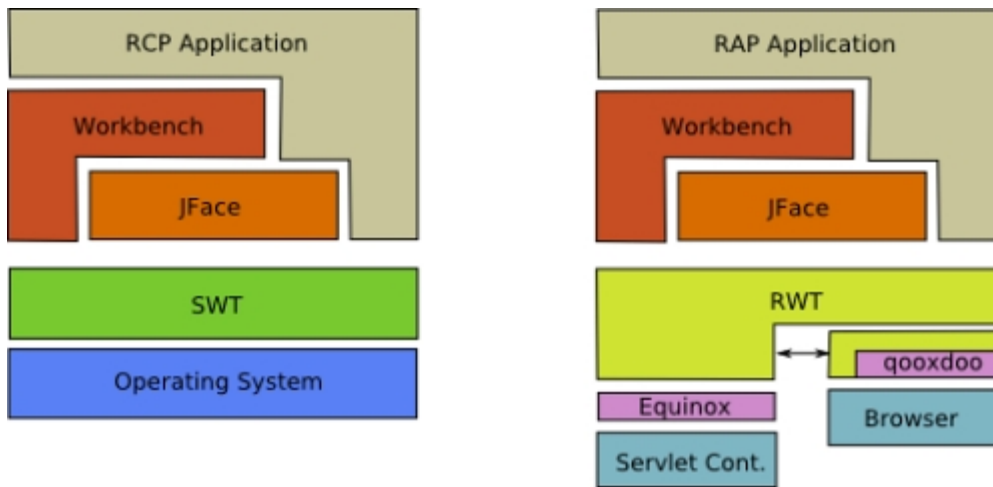
- *Which Platform Should You Choose for Developing Java Desktop Applications?*

## 2- RCP (Rich Client Platform)

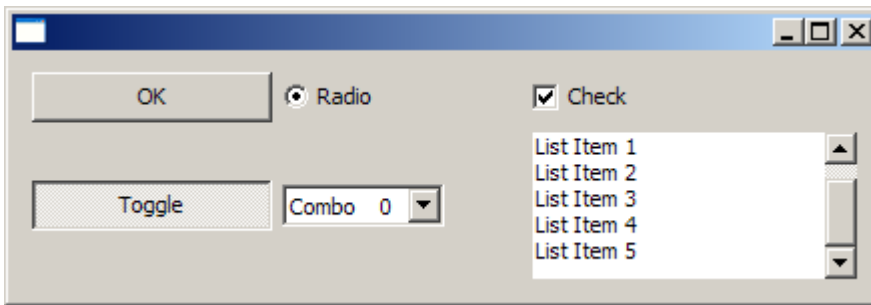
RCP (Rich Client Platform) - As a platform based on SWT, used for programming Desktop applications, so far it has built a platform that allows you to develop desktop-style applications Workbench, like Eclipse IDE, or programmers can integrate the plugin to Eclipse IDE.

But even if you want to use SWT to programming, and do not need to use those provided by the RCP you should also create an RCP application.



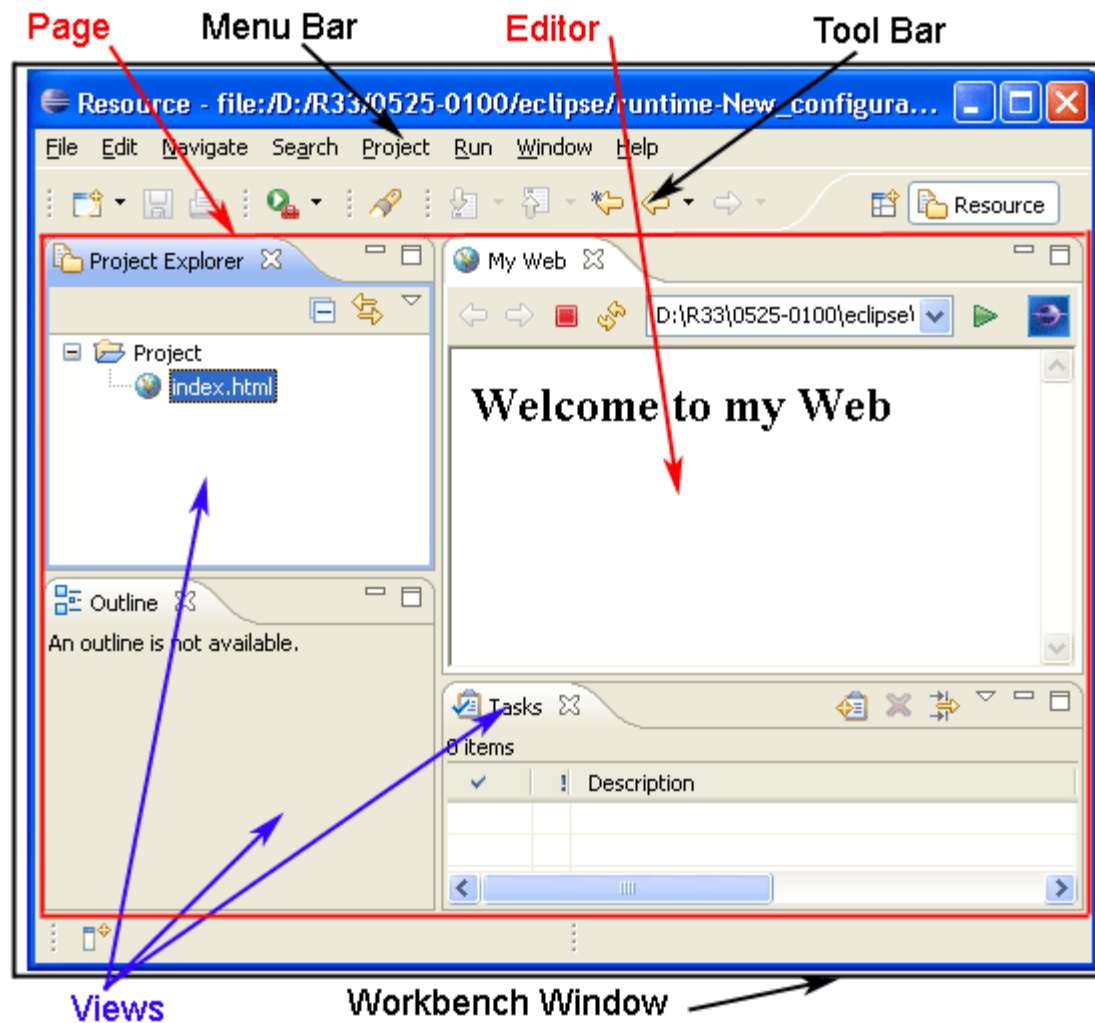


SWT:



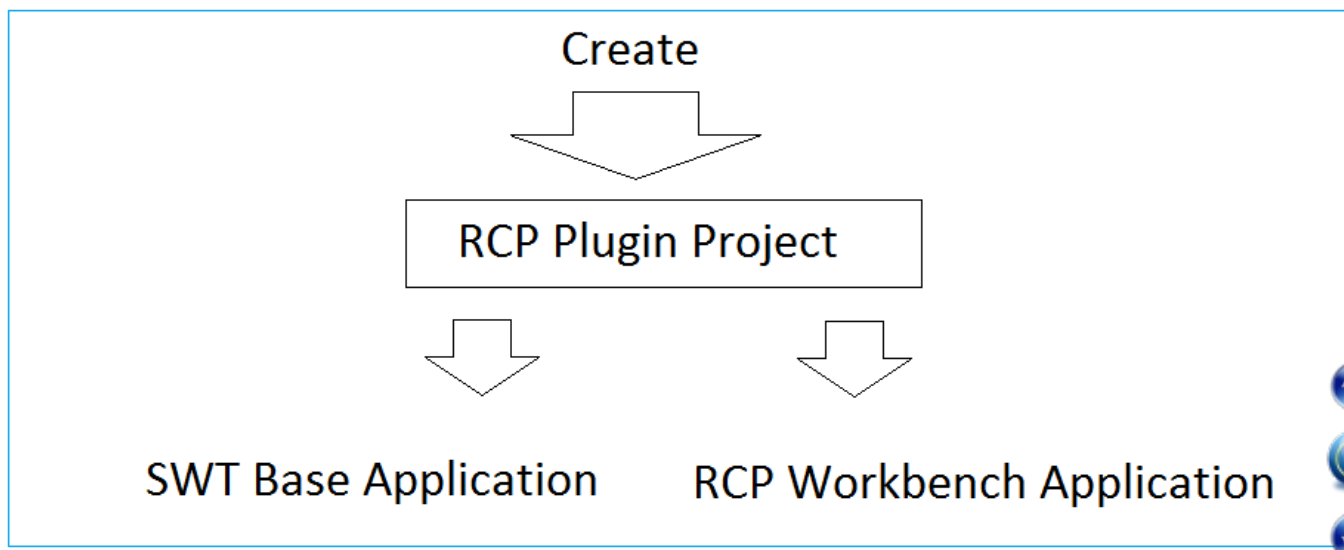
Workbench Application:





To create a desktop application using SWT. On the Eclipse, we will create an RCP Plugin Project. You have 2 options.

- Only use the features of the SWT
- Using the platform provided by the RCP to RCP Application Workbench programming



In this document, I will guide you to become familiar with basic programming SWT, using WindowBuilder to drag and drop components into the interface.

### 3- The settings required before starting

Some required settings before you start:

You need the latest version of **Eclipse**. There currently is **Eclipse 4.7** (Codes **OXYGEN**).

- <http://eclipse.org/downloads/>

In my opinion you to download package: "**Eclipse IDE for Java EE Developers**". The only different is number of Plugins, for the purpose of different programming. You can install additional plugins for other purposes if desired.

Package Solutions

Filter Packages ▼

	<b>Eclipse IDE for Java EE Developers</b> , 259 MB Downloaded 1,142,884 Times Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...		Windows 32 Bit Windows 64 Bit
	<b>Eclipse IDE for Java Developers</b> , 155 MB Downloaded 508,990 Times The essential tools for any Java developer, including a Java IDE, a CVS client, Git client, XML Editor, Mylyn, Maven integration...		Windows 32 Bit Windows 64 Bit

**WindowBuilder**, this is a plugin that allows you to design **SWT** GUI applications using drag and drop convenience.

See installation instructions at:

- [Install WindowBuilder for Eclipse](#)



### 4- Some concepts of SWT.





## 4.1- Display & Shell

The Display and Shell classes are key components of SWT applications.

- **org.eclipse.swt.widgets.Shell** class represents a window.
- **org.eclipse.swt.widgets.Display** class is responsible for managing event loops, fonts, colors and for controlling the communication between the UI thread and other threads. Display is the base for all SWT capabilities.

Every SWT application requires at least one Display and one or more Shell instances. The main Shell gets, as a default parameter, a Display as a constructor argument. Each Shell is constructed with a Display and if none is provided during construction it will use either the Display which is currently used or a default one.

Example:

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

// run the event loop as long as the window is open
while (!shell.isDisposed()) {
    // read the next OS event queue and transfer it to a SWT event
    if (!display.readAndDispatch())
    {
        // if there are currently no other OS event to process
        // sleep until the next OS event is available
        display.sleep();
    }
}

// disposes all associated windows and their components
display.dispose();
```

## 4.2- SWT Widgets

SWT widgets are located in the packages **org.eclipse.swt.widgets** and **org.eclipse.swt.custom**. Widgets extend either the Widget or the Control class. Several of these widgets are depicted in the



following graphic.

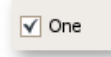
This graphic is a screenshot of the SWT widget homepage.



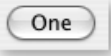
**Browser**  
[javadoc - snippets](#)



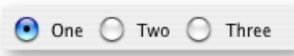
**Button (SWT.ARROW)**  
[javadoc - snippets](#)



**Button (SWT.CHECK)**  
[javadoc - snippets](#)



**Button (SWT.PUSH)**  
[javadoc - snippets](#)



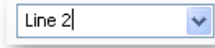
**Button (SWT.RADIO)**  
[javadoc - snippets](#)



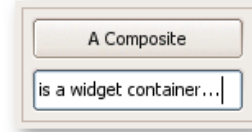
**Button (SWT.TOGGLE)**  
[javadoc - snippets](#)



**Canvas**  
[javadoc - snippets](#)



**Combo**  
[javadoc - snippets](#)



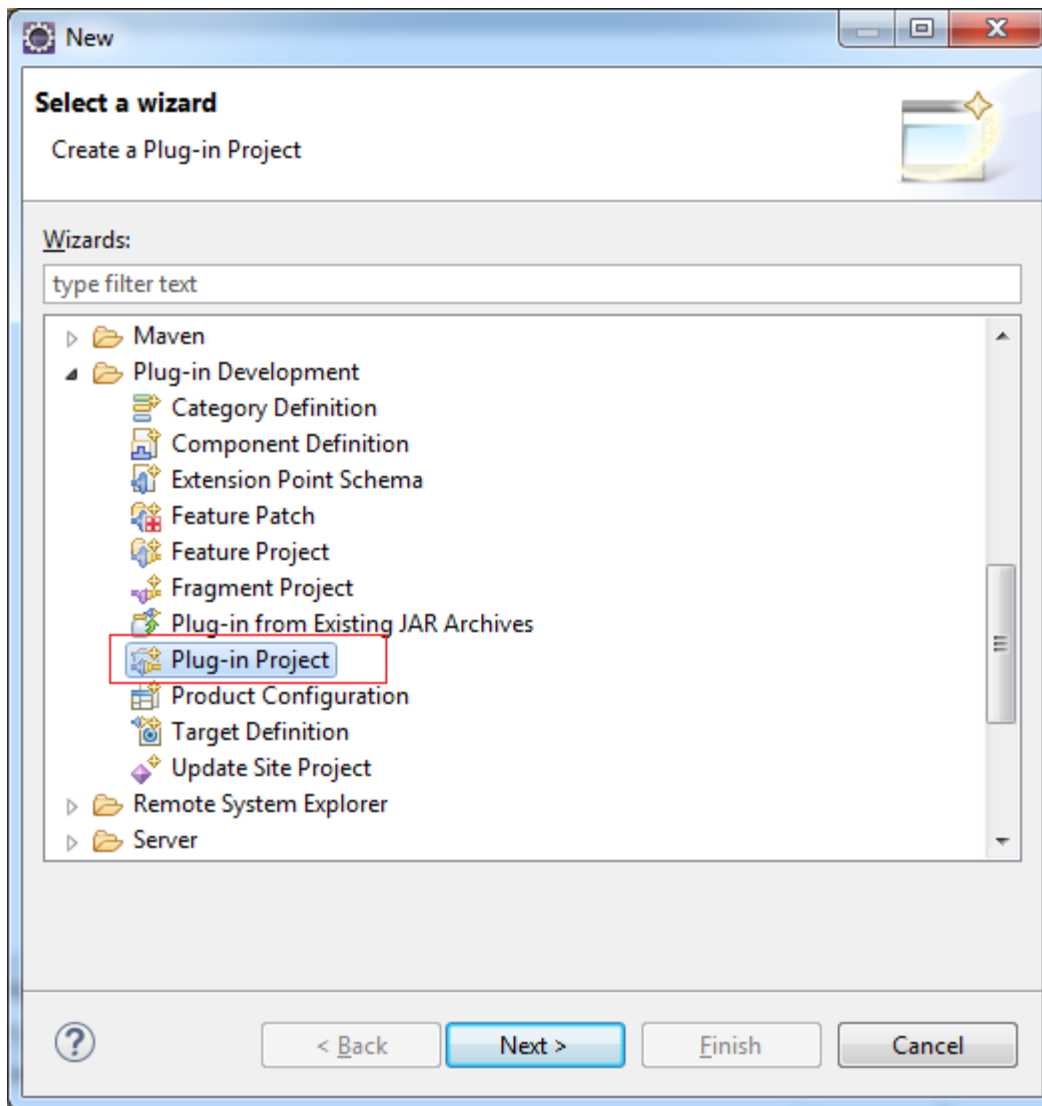
**Composite**  
[javadoc - snippets](#)

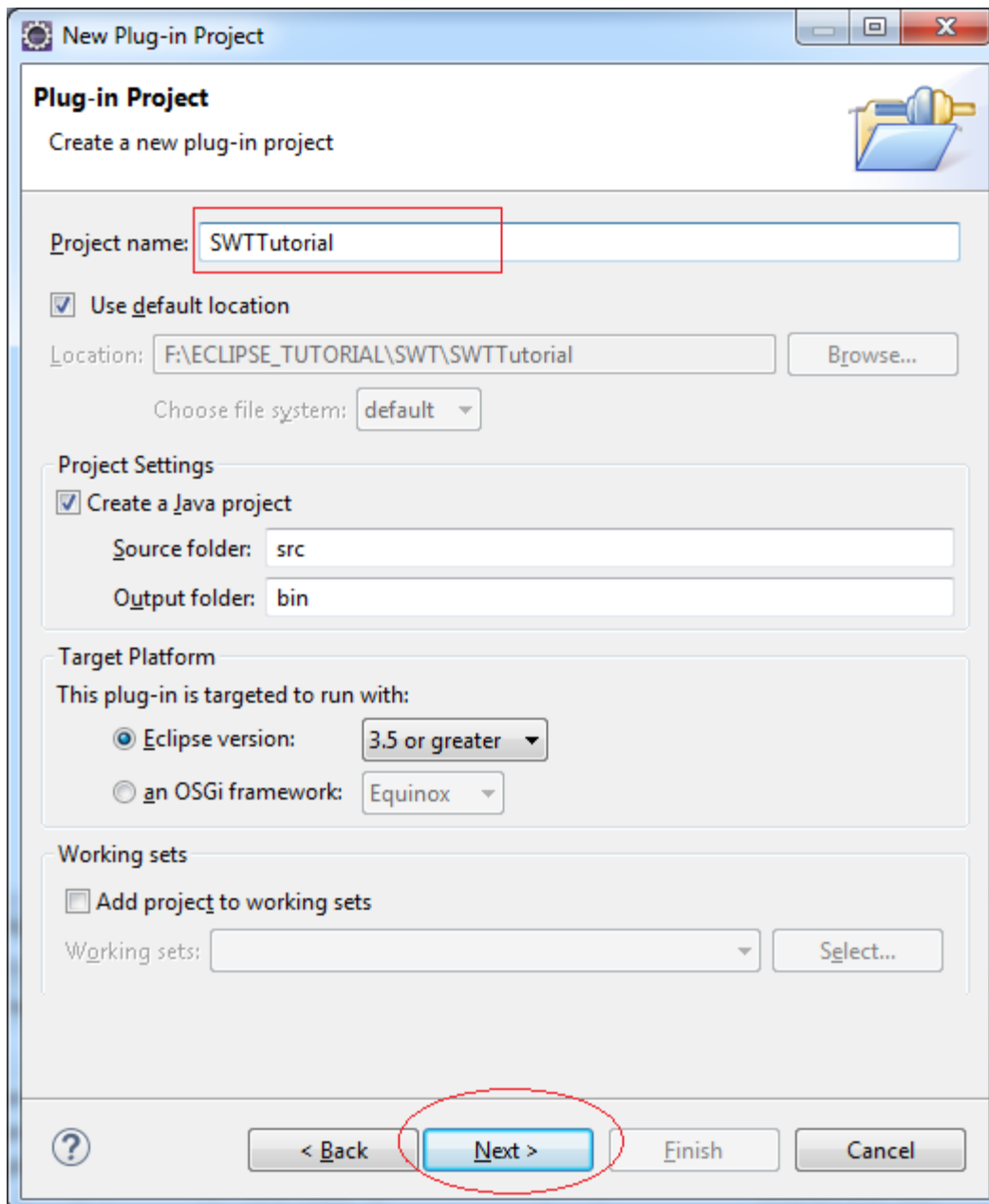
## 5- Create RCP Plugin Project

In Eclipse select:

- File/New/Other...







- So there is no need to create a Workbench application so we can not check in (1) as shown below.
- Select 'Yes' in the region (2) to Eclipse RCP Application created (Run on the Desktop), otherwise it will create RAP Application (Running on Web).



**New Plug-in Project**

**Content**  
Enter the data required to generate the plug-in.

**Properties**

ID:

Version:

Name:

Vendor:

Execution Environment:

**Options**

☐ Generate an activator, a Java class that controls the plug-in's life cycle

1 Activator:

☐ This plug-in will make contributions to the UI

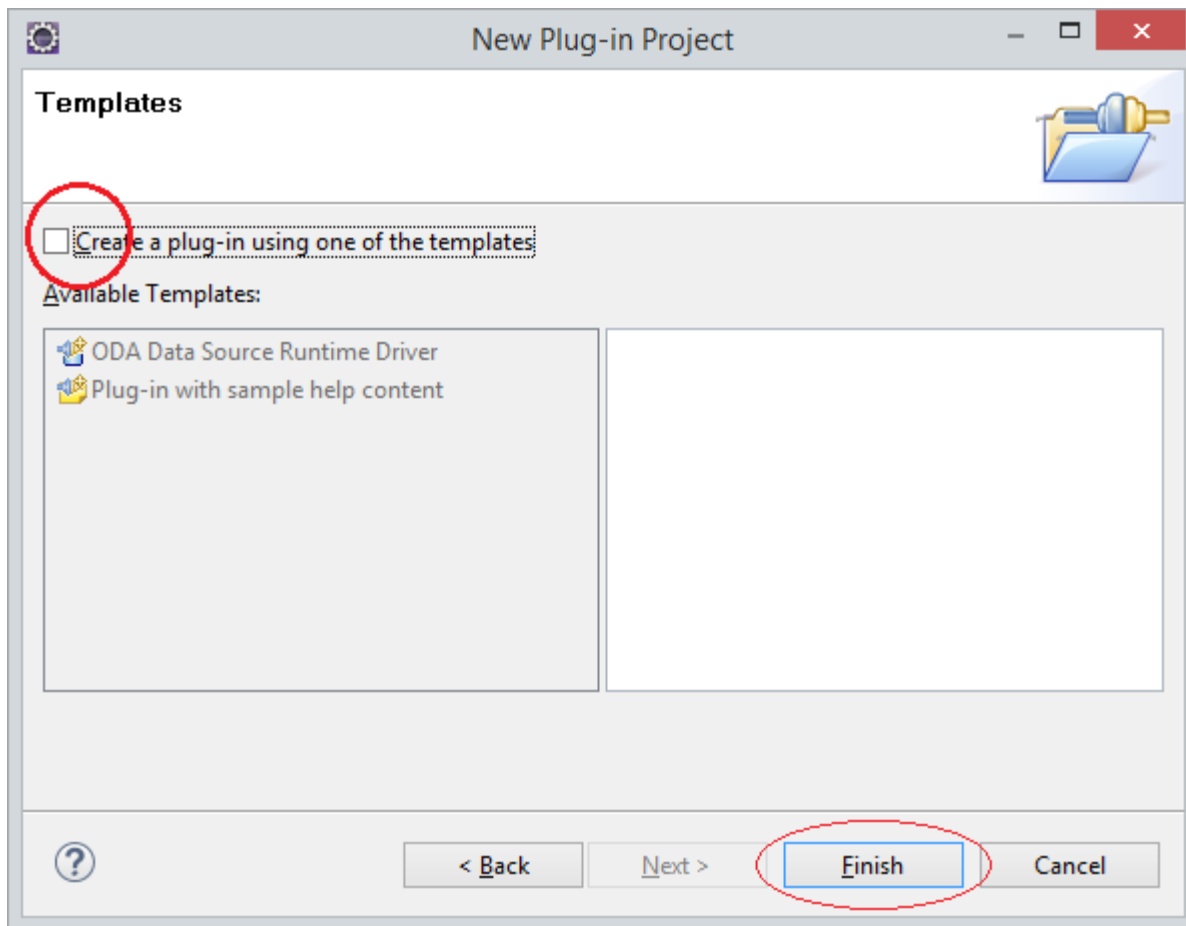
☐ Enable API analysis

**Rich Client Application**

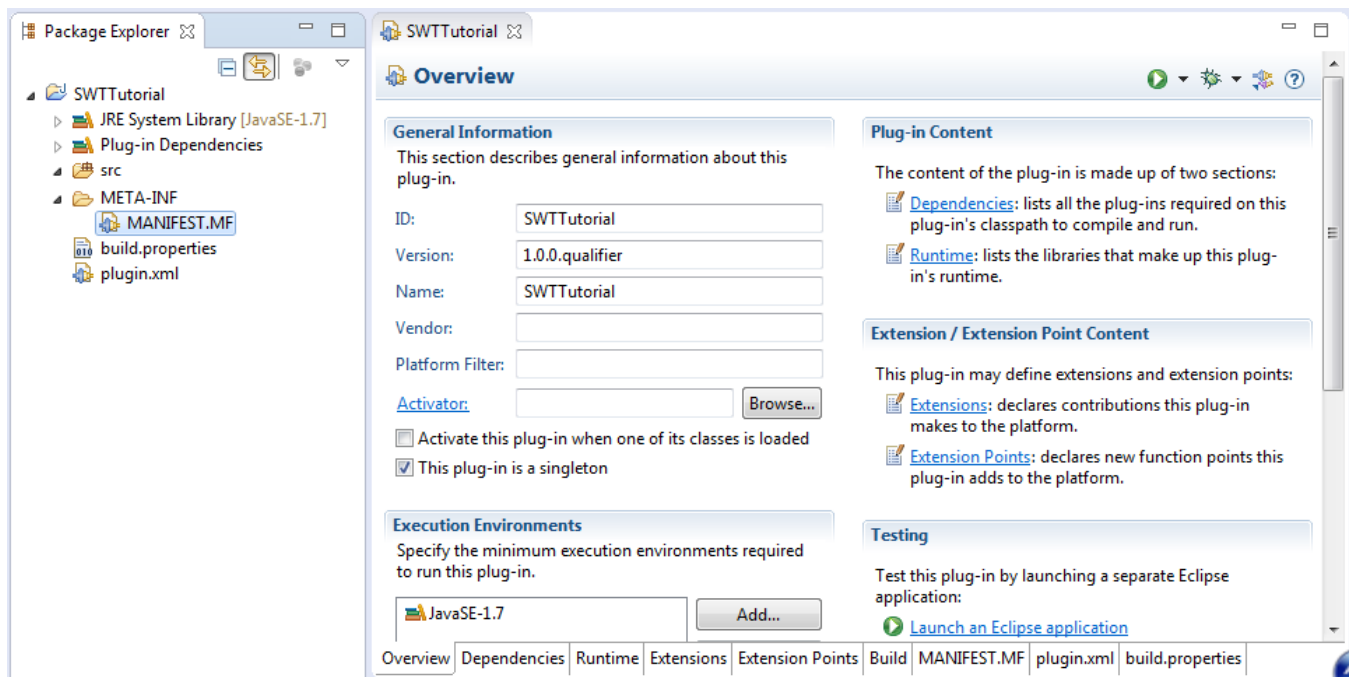
Would you like to create a 3.x rich client application? ☒ Yes ☐ No

2





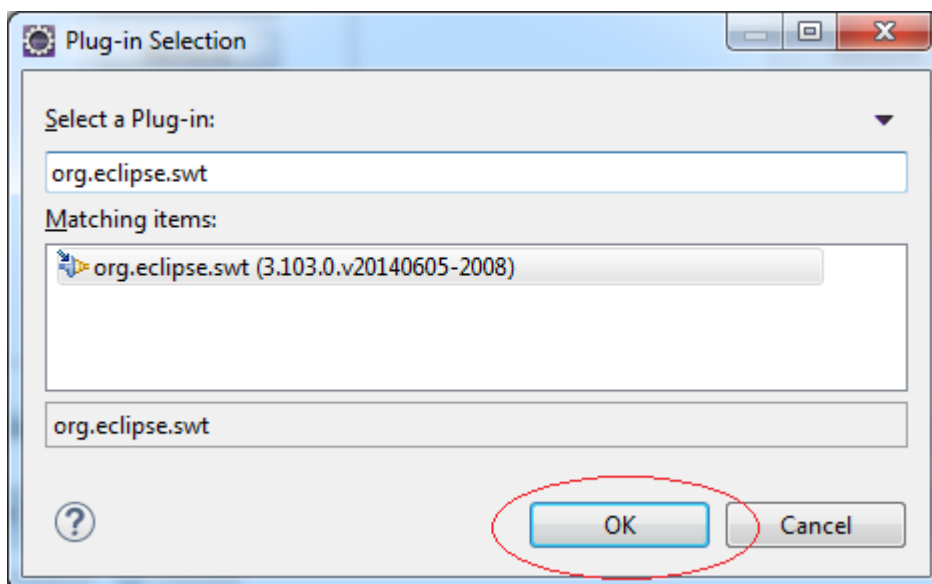
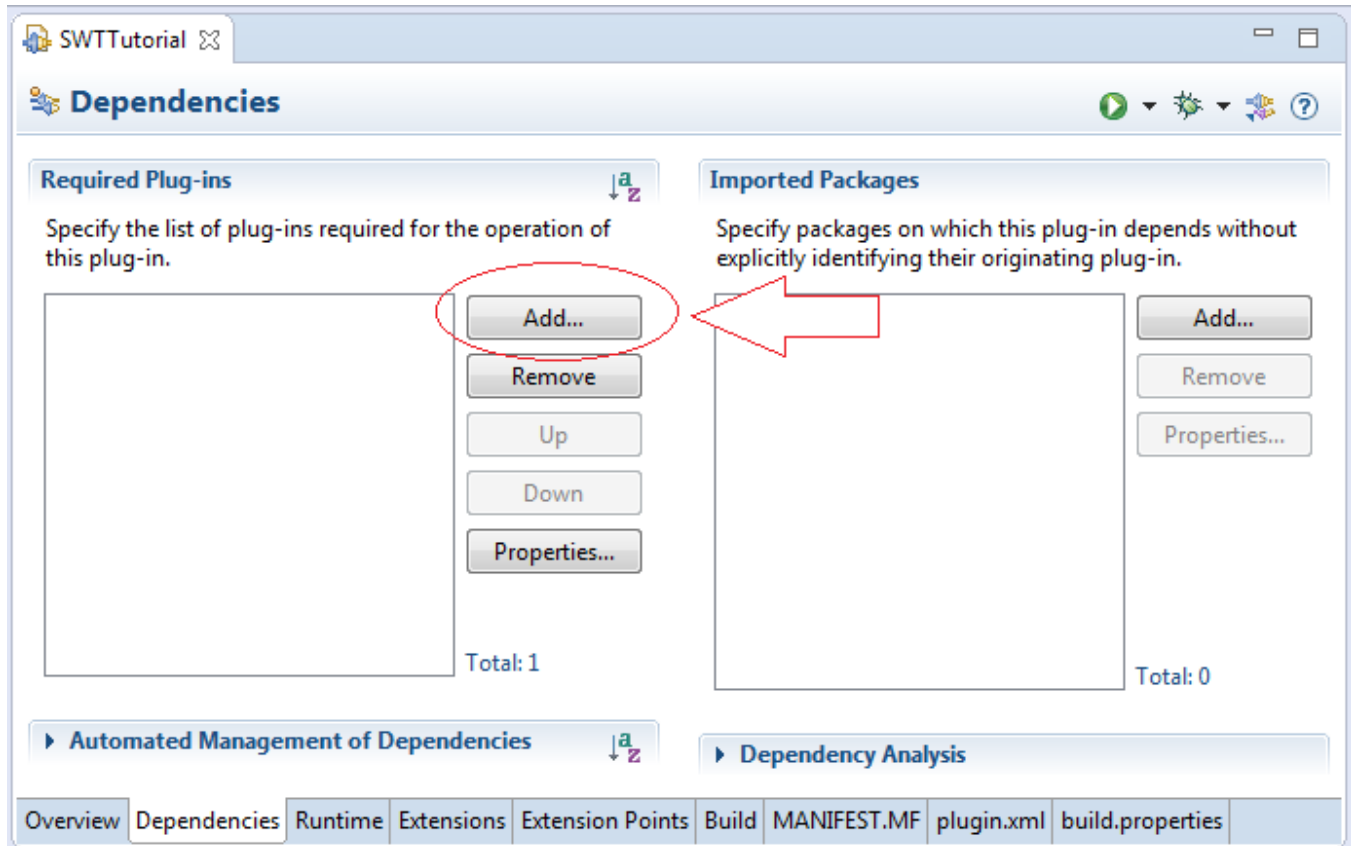
Created Project:

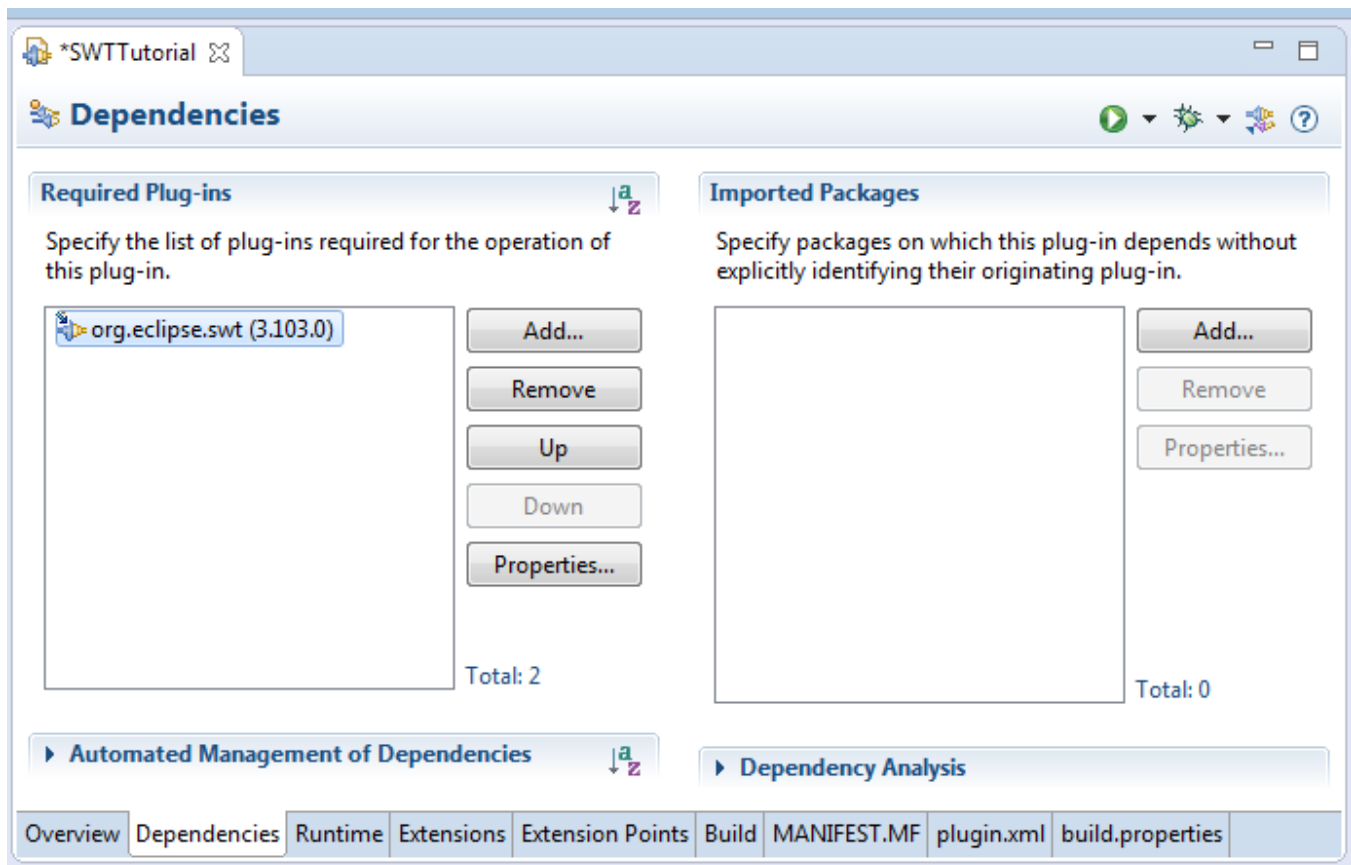


Add swt library: org.eclipse.swt

/

If you develop RAP, using `org.eclipse.rap.rwt`





## 6- First Example

This is a simple example, not using the drag and drop tools WindowBuilder.

HelloSWT.java

```
package org.o7planning.tutorial.swt.helloswt;
```





```
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

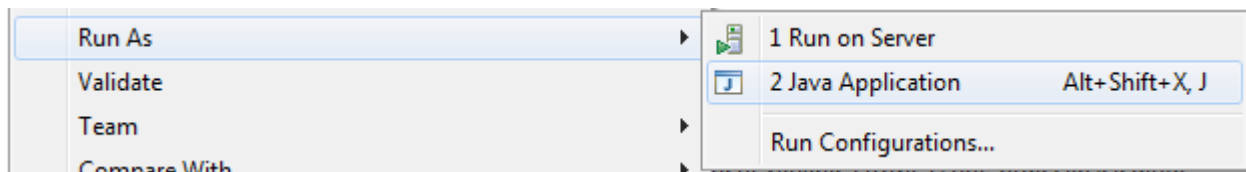
public class HelloSWT {

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

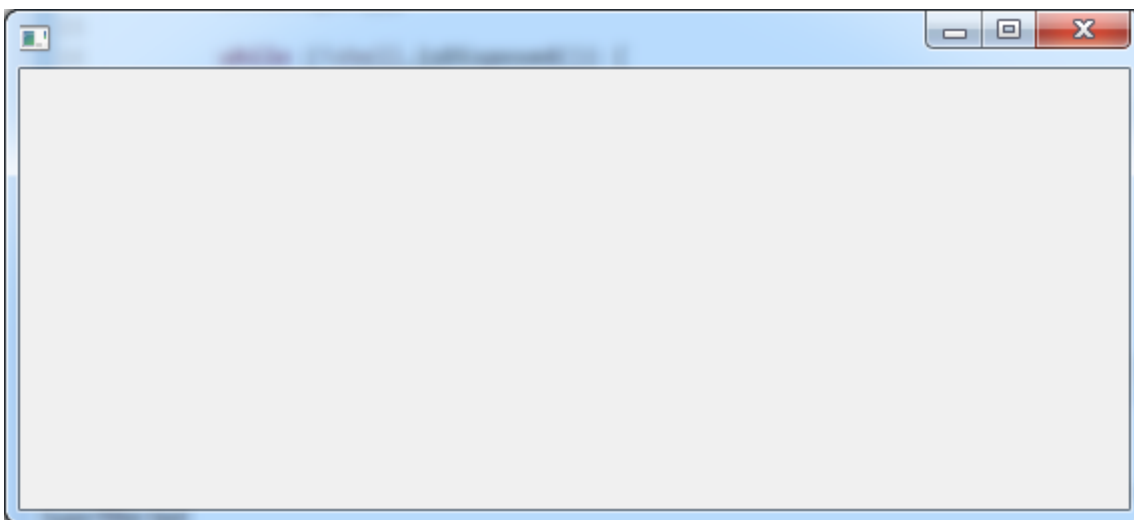
        shell.open();

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

Right-click on the class **HelloSWT**, and select **Run As/Java Application**.



Result:

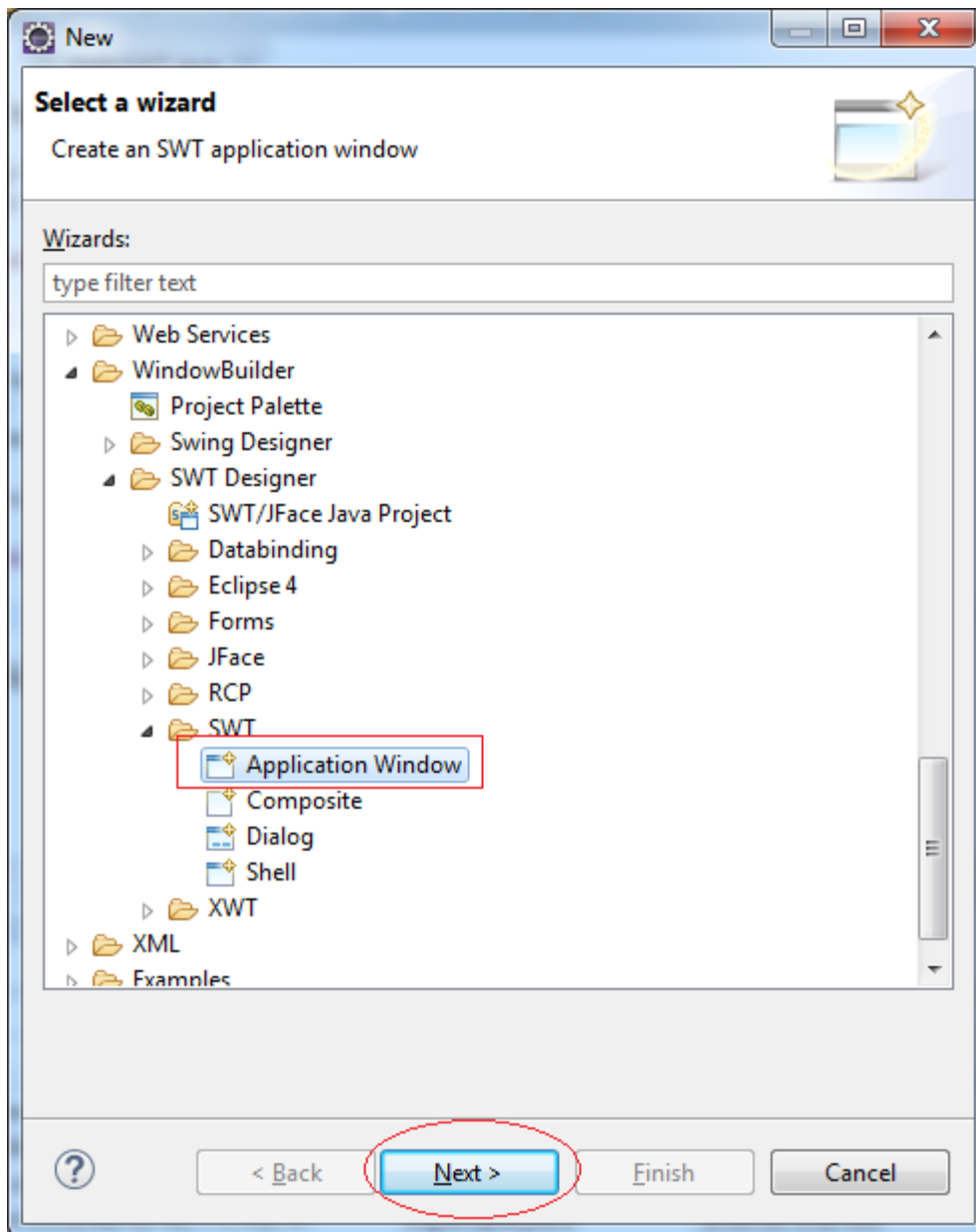


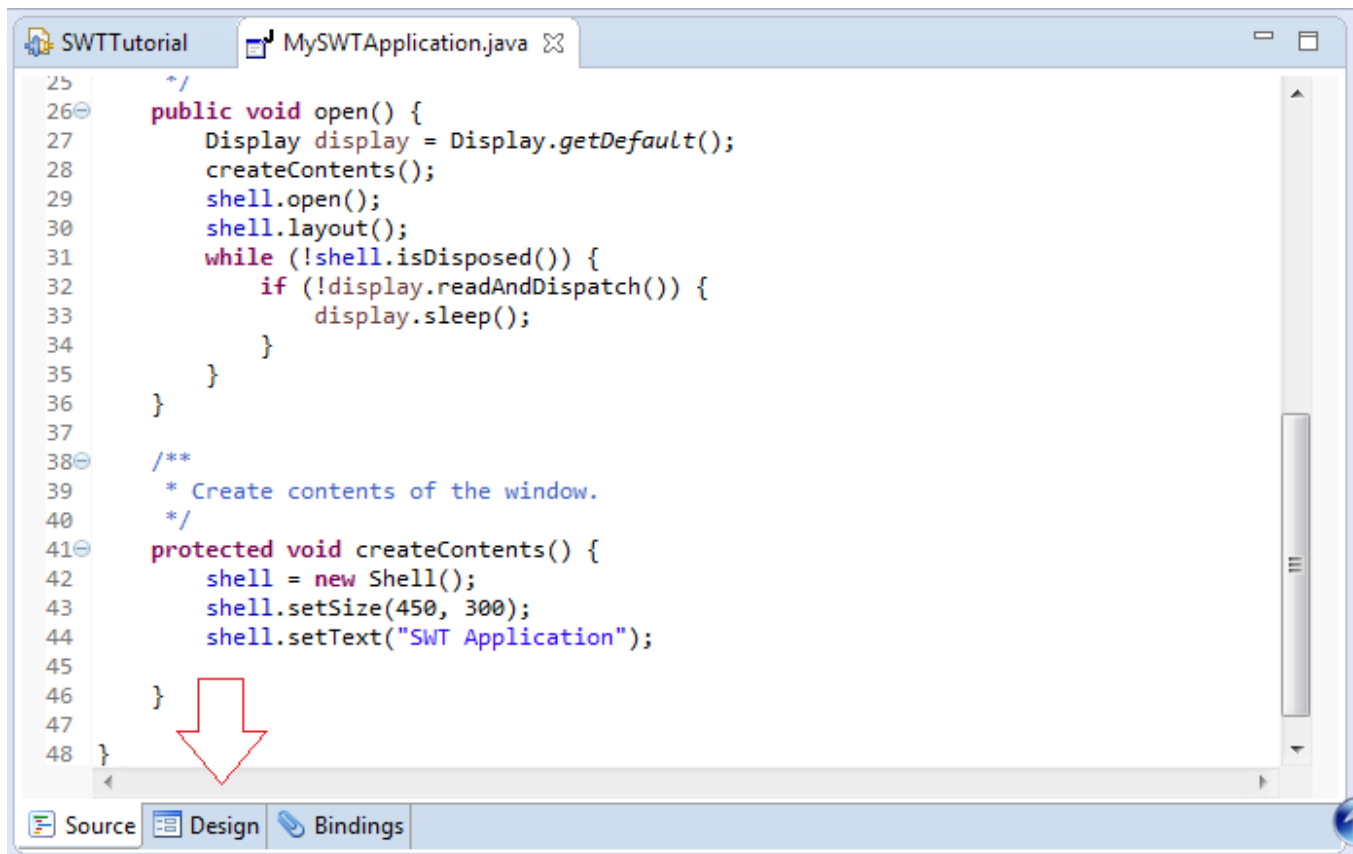
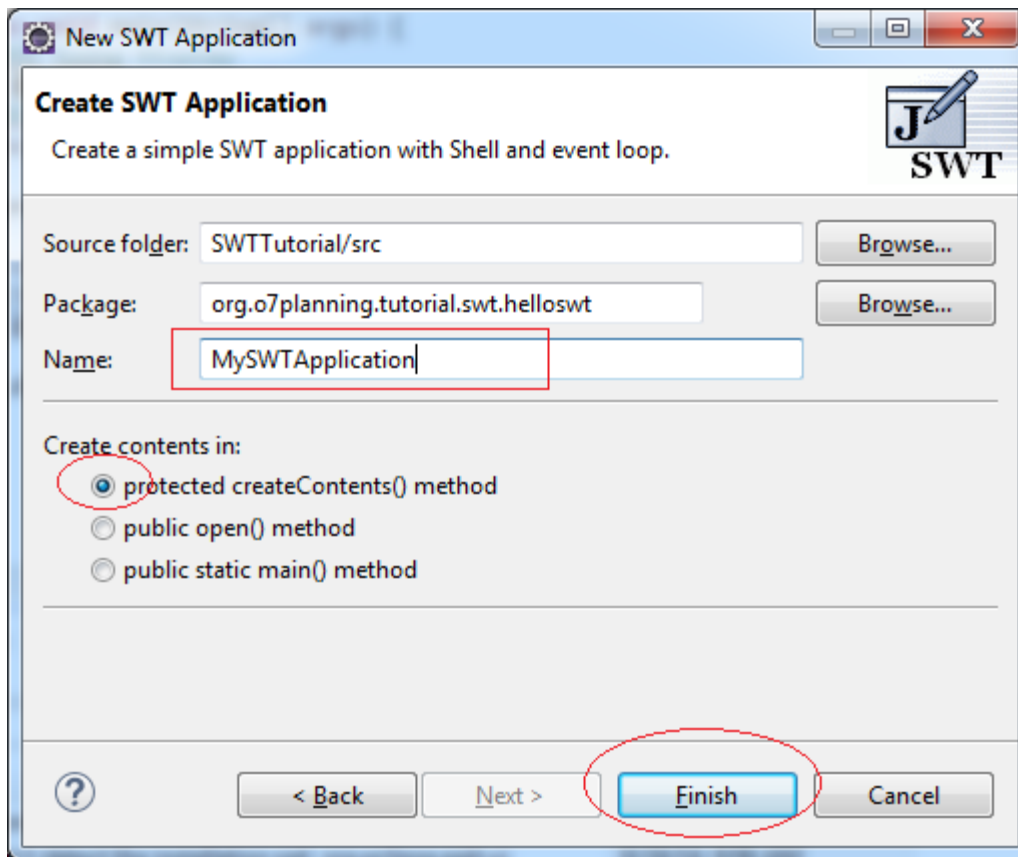
## 7- Using WindowBuilder

Next we will create an example for drag and drop with **WindowBuilder**.

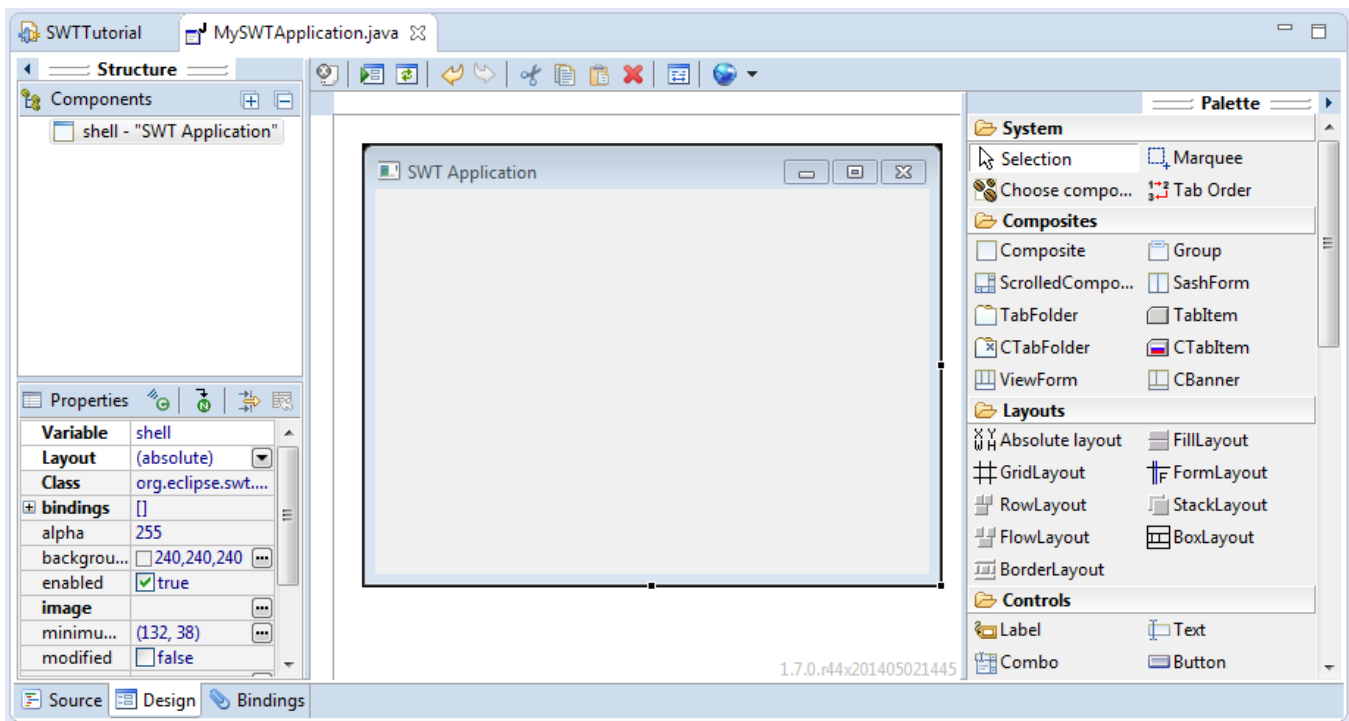
- File/New/Other ..







This is the window of WindowBuilder design. It allows you to drag and drop the widgets easily.



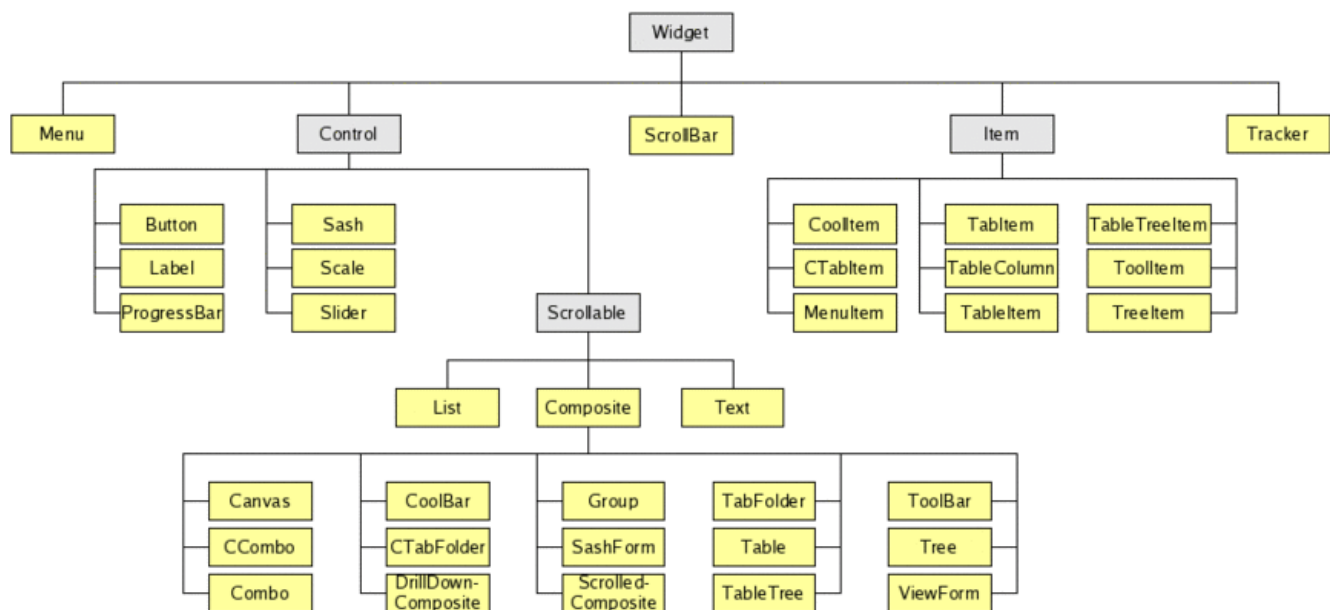
You can watch the video below:



## 8- SWT Widget

### 8.1- Overview

This is the hierarchy of widgets in SWT.



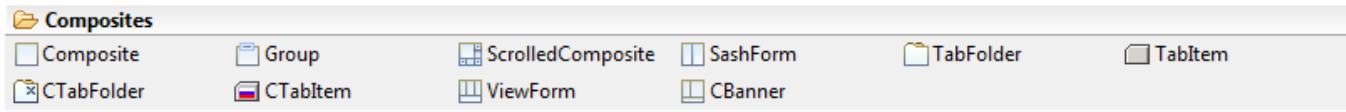
You can view the demo of the Control at the link below, it's RWT Control, but they are essentially the same as SWT control.

#### Demo:

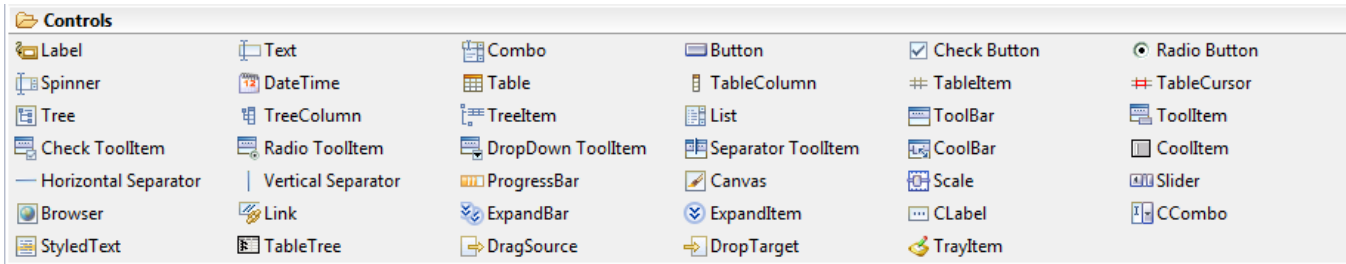
- <http://rap.eclipsesource.com/demo/release/controls/>



## 8.2- Widgets can contain other widgets (Container)



## 8.3- Controls

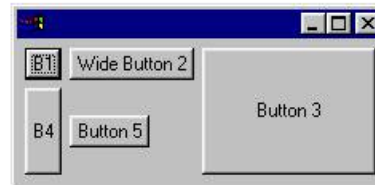
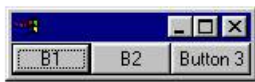


## 9- SWT Layout

### 9.1- What is Layout?

Put simply, Layout is how to arrange the components on the interface.



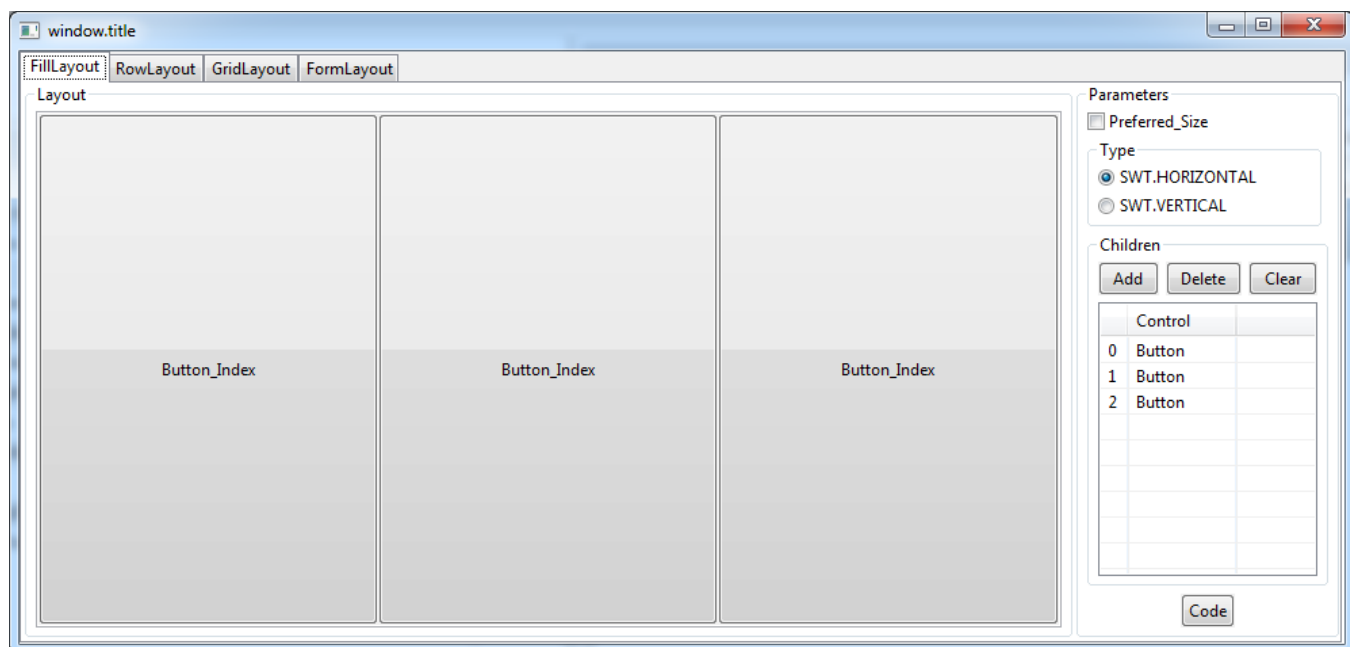


The standard layout classes in the SWT library are:

- **FillLayout** – lays out equal-sized widgets in a single row or column
- **RowLayout** – lays out widgets in a row or rows, with fill, wrap, and spacing options
- **GridLayout** – lays out widgets in a grid

## 9.2- Online Example

This is an example online, allowing you to see how the exercise of the Layout.



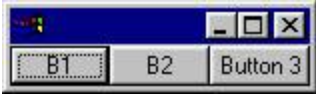



- TODO Link?

## 9.3- FillLayout

**FillLayout** is the simplest layout class. It lays out widgets in a single row or column, forcing them to be the same size. Initially, the widgets will all be as tall as the tallest widget, and as wide as the widest. FillLayout does not wrap, and you cannot specify margins or spacing.



```
FillLayout fillLayout = new FillLayout();  
fillLayout.type = SWT.VERTICAL;  
shell.setLayout(fillLayout);
```

	Initial	After resize
<code>fillLayout.type = SWT.HORIZONTAL</code> (default)		
<code>fillLayout.type = SWT.VERTICAL</code>		

Video:



## FillLayoutExample.java

```
package org.o7planning.tutorial.swt.layout;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.layout.RowLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class FillLayoutExample {

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

        //
        Composite parent = new Composite(shell, SWT.NONE);

        FillLayout fillLayout= new FillLayout();
        fillLayout.type= SWT.VERTICAL;

        parent.setLayout(fillLayout);

        Button b1 = new Button(parent, SWT.NONE);
        b1.setText("B1");

        Button b2 = new Button(parent, SWT.NONE);
        b2.setText("B2");

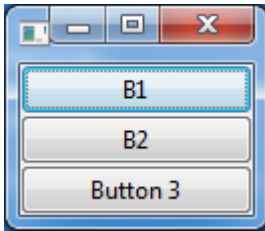
        Button button3 = new Button(parent, SWT.NONE);
        button3.setText("Button 3");

        // Windows back to natural size.
        shell.pack();
        //
        shell.open();
    }
}
```



```
while (!shell.isDisposed()) {  
    if (!display.readAndDispatch())  
        display.sleep();  
}  
// tear down the SWT window  
display.dispose();  
}  
}
```

Run result:



## 9.4- RowLayout





**RowLayout** is more commonly used than **FillLayout** because of its ability to wrap, and because it provides configurable margins and spacing. **RowLayout** has a number of configuration fields. In addition, the height and width of each widget in a **RowLayout** can be specified by setting a **RowData** object into the widget using **setLayoutData**.

### The field configuration:

Wrap, Pack, Justify:

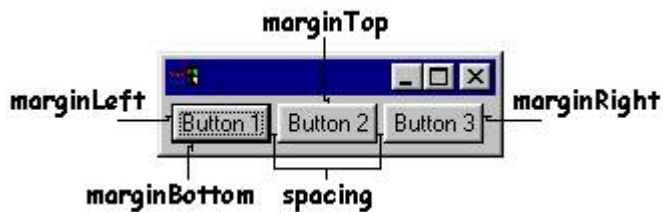
	Initial	After resize
<b>wrap = true</b> <b>pack = true</b> <b>justify = false</b>  (defaults)		
<b>wrap = false</b>  (clips if not enough space)		



<p><code>pack = false</code></p> <p>(all widgets are the same size)</p>		
<p><code>justify = true</code></p> <p>(widgets are spread across the available space)</p>		

## MarginLeft, MarginTop, MarginRight, MarginBottom, Spacing:

These fields control the number of pixels between widgets (**spacing**) and the number of pixels between a widget and the side of the parent **Composite** (**margin**). By default, **RowLayouts** leave 3 pixels for margin and spacing. The margin and spacing fields are shown in the following diagram.



Video:



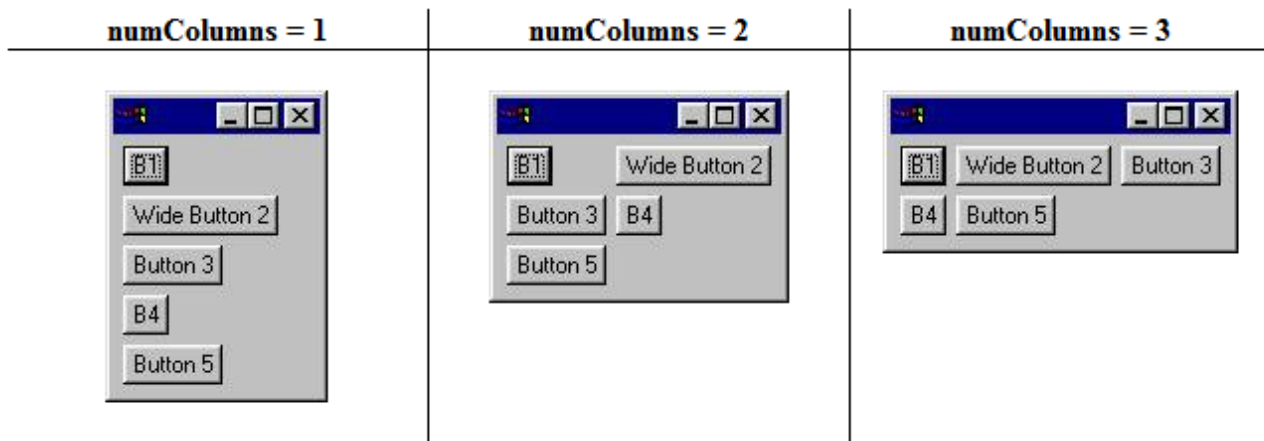
## 9.5- GridLayout

**GridLayout** is the most useful and powerful of the standard layouts, but it is also the most complicated. With a **GridLayout**, the widget children of a **Composite** are laid out in a grid. **GridLayout** has a number of configuration fields, and, like **RowLayout**, the widgets it lays out can have an associated layout data object, called **GridData**. The power of **GridLayout** lies in the ability to configure **GridData** for each widget controlled by the **GridLayout**.

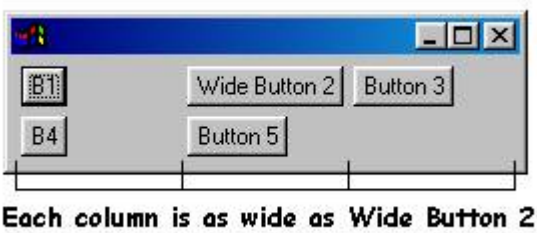
### The configuration of the GridLayout:

- NumColumns





- MakeColumnsEqualWidth



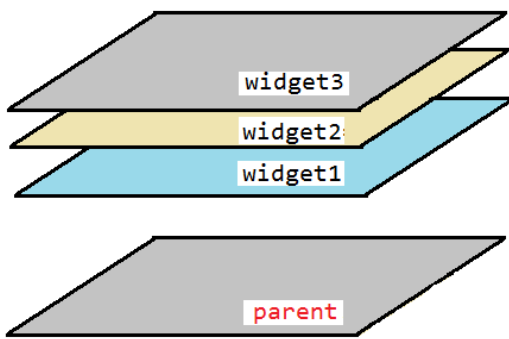
Video GridLayout:



## 9.6- StackLayout

This Layout stacks all the controls one on top of the other and resizes all controls to have the same size and location. The control specified in **topControl** is visible and all other controls are not visible. Users must set the **topControl** value to flip between the visible items and then call **layout()** method on the composite which has the **StackLayout**.





```
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();
```

Video StackLayout:

## 9.7- Combine Layout





Above, we have become familiar with the standard layout. Combining different layout, and the other container (**Composite**, **TabFolder**, **SashForm**, ..) will create the desired interface.

- TODO

## 10- Write the class extending from the SWT widget

Sometimes you need to write a class extending from available widget class of SWT. It is perfectly normal, but there is a small note, you need to override the method `checkSubclass()` without having to do anything in that method.

### MyButton.java

```
package org.o7planning.tutorial.swt.swtsubclass;

import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;

public class MyButton extends Button {

    public MyButton(Composite parent, int style) {
        super(parent, style);
    }

    // You have to override this method.
    @Override
    protected void checkSubclass() {
        // No need to do anything.
    }

}
```

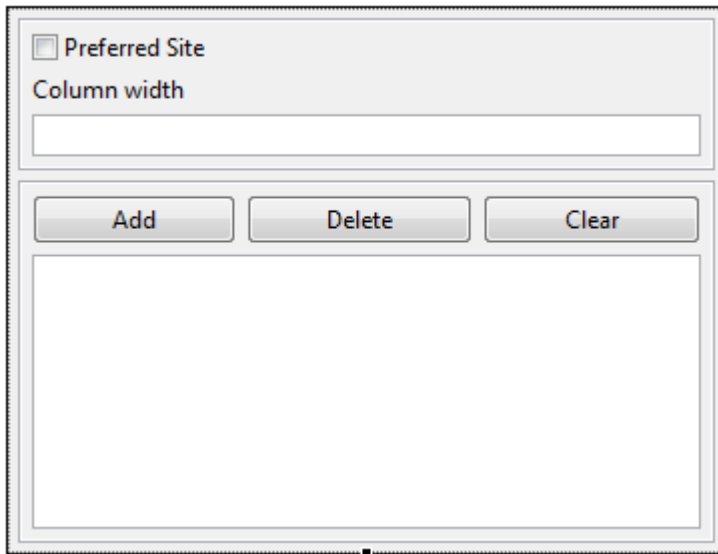
## 11- Modular component interfaces



In case you have designed a complex interface. The splitting of designs is necessary and then put together, this will be easier if designing on **Window Builder**.

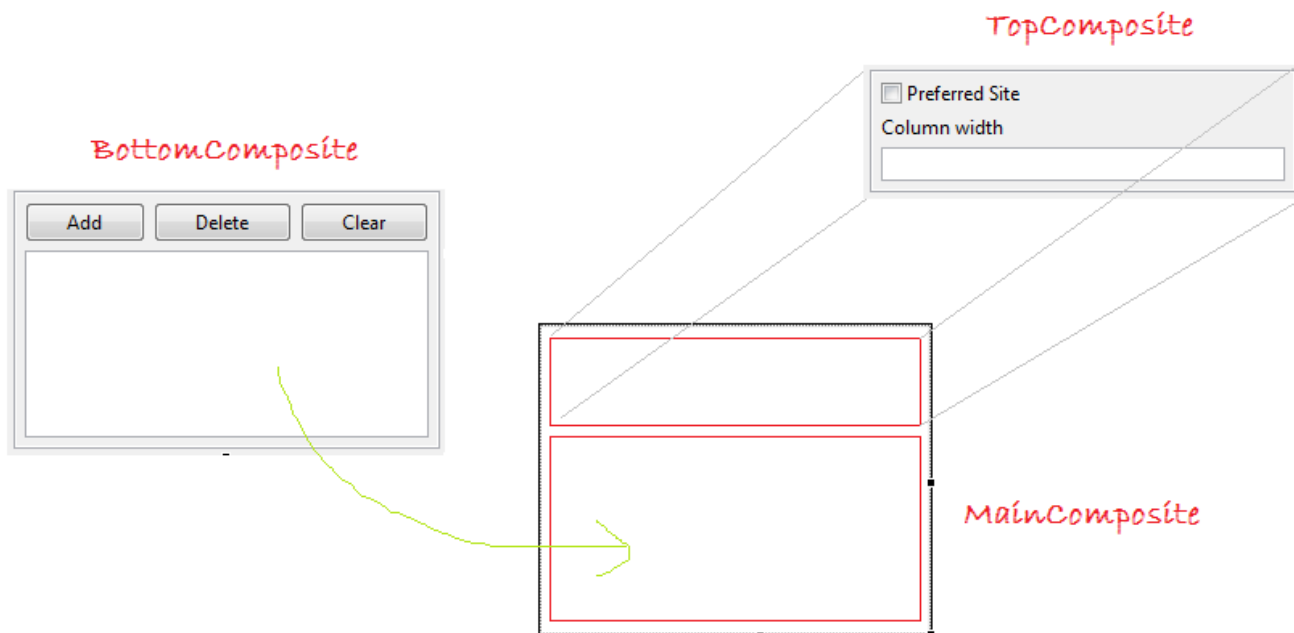
Please see the following interface, and we will try to split it up.

Suppose that you want to design an interface like the illustration below. (It is not complicated to split the design, but this is an example to illustrate how to split interface design)



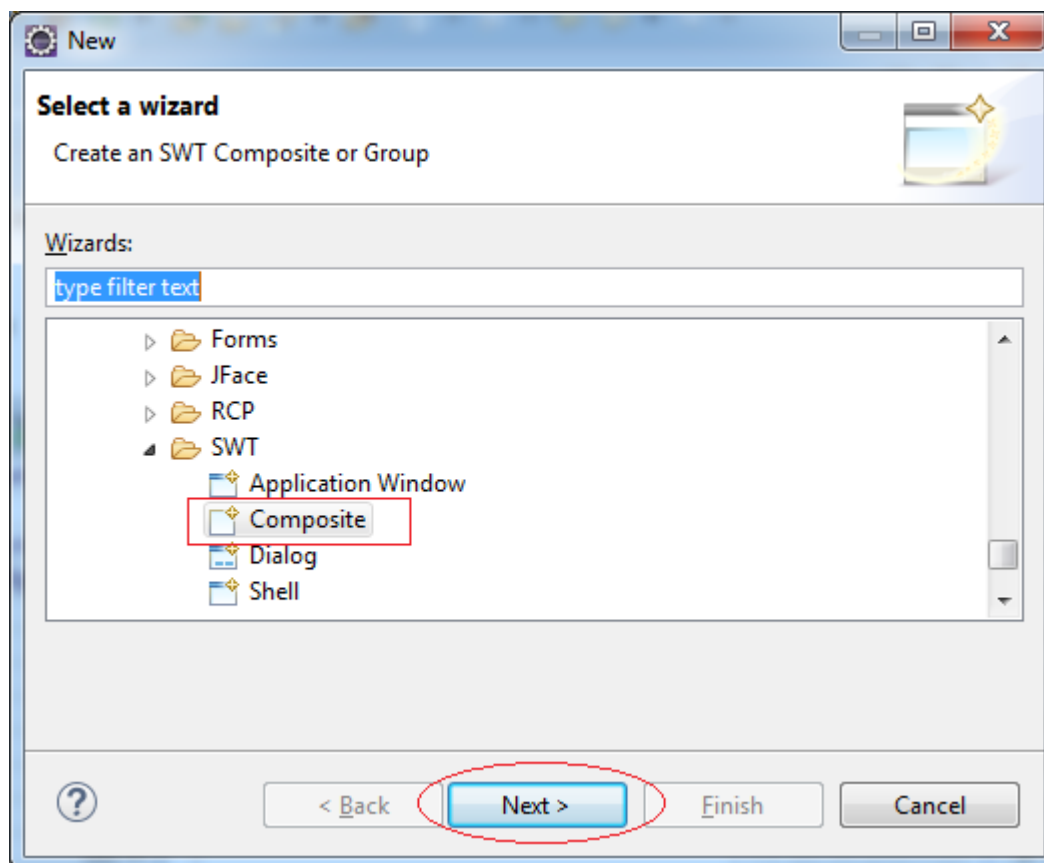
We can design two separate Composite and graft it on MainComposite.

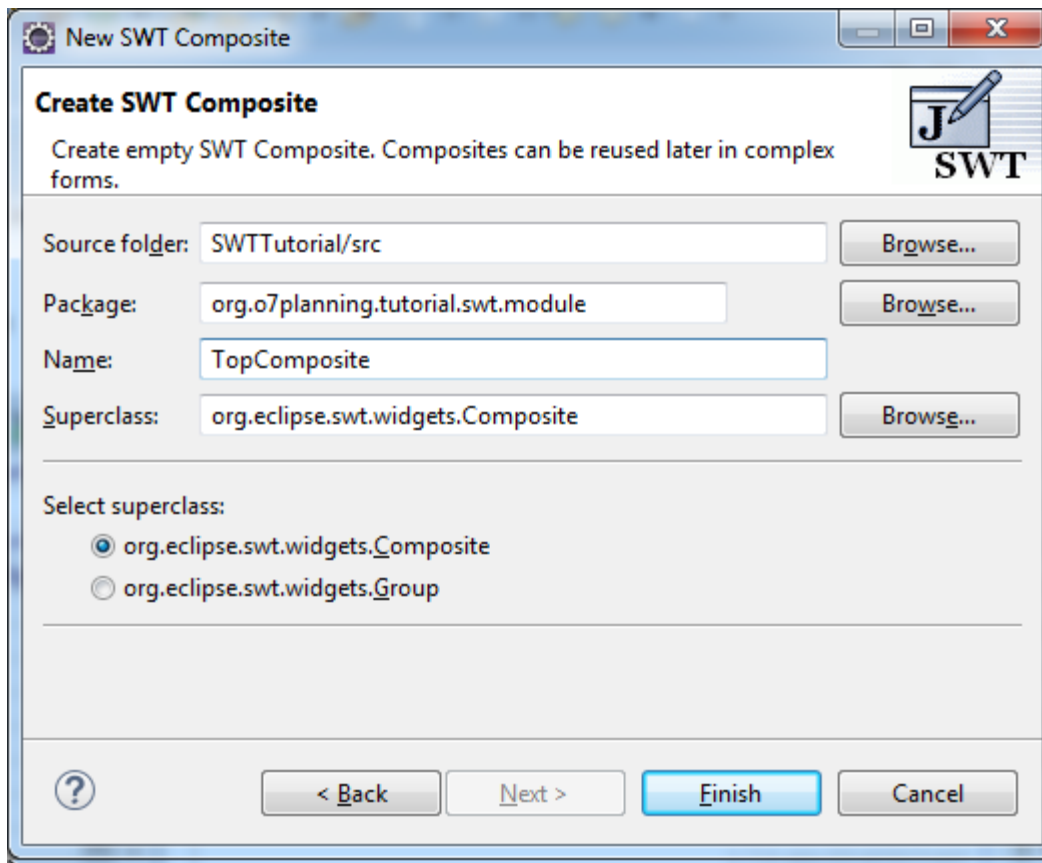




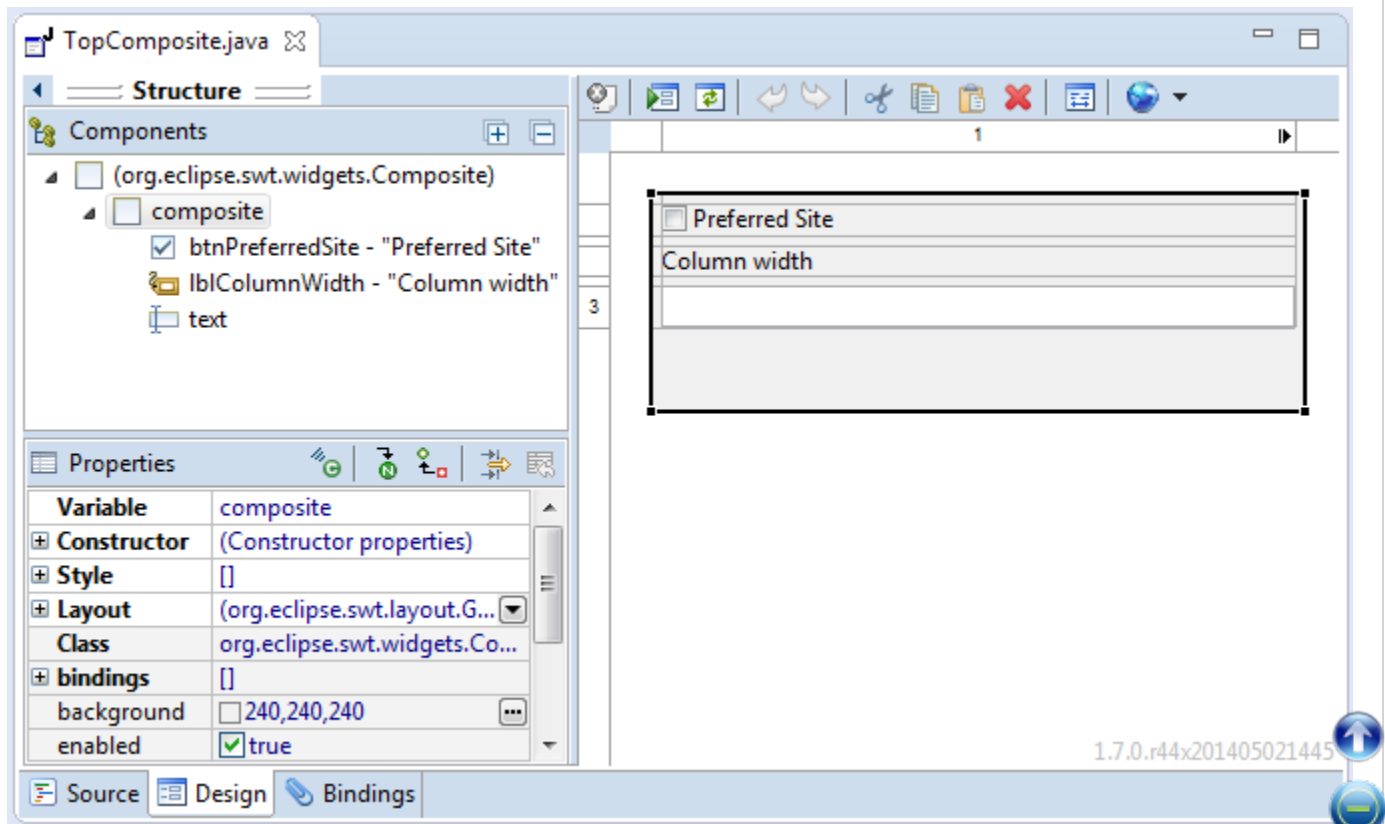
## TopComposite

- File/New/Other...





Interface Design for TopComposite.



TopComposite.java

```
package org.o7planning.tutorial.swt.module;

import org.eclipse.swt.widgets.Composite;

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));

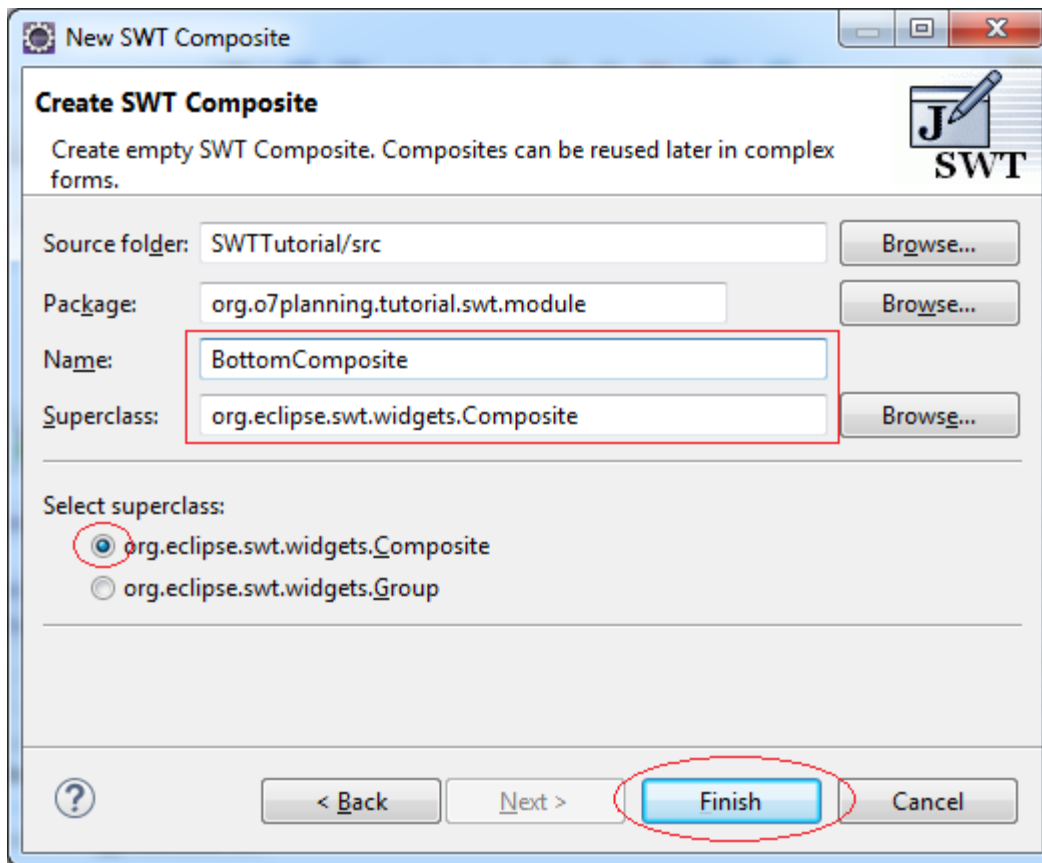
    }

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

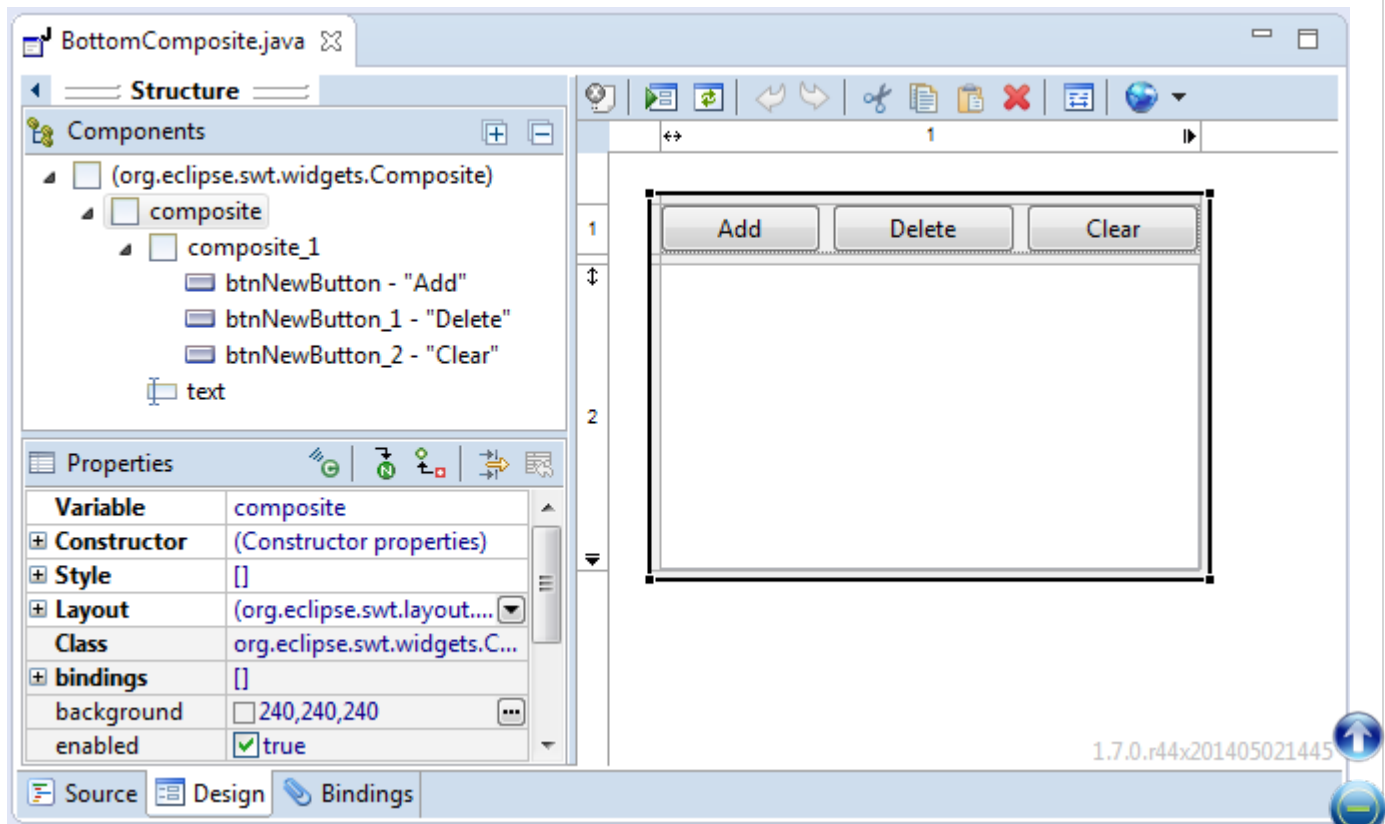
## BottomComposite

Similarly, create class BottomComposite





Interface Design for **BottomComposite**:



BottomComposite.java

```
package org.o7planning.tutorial.swt.module;

import org.eclipse.swt.SWT;

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() {
```

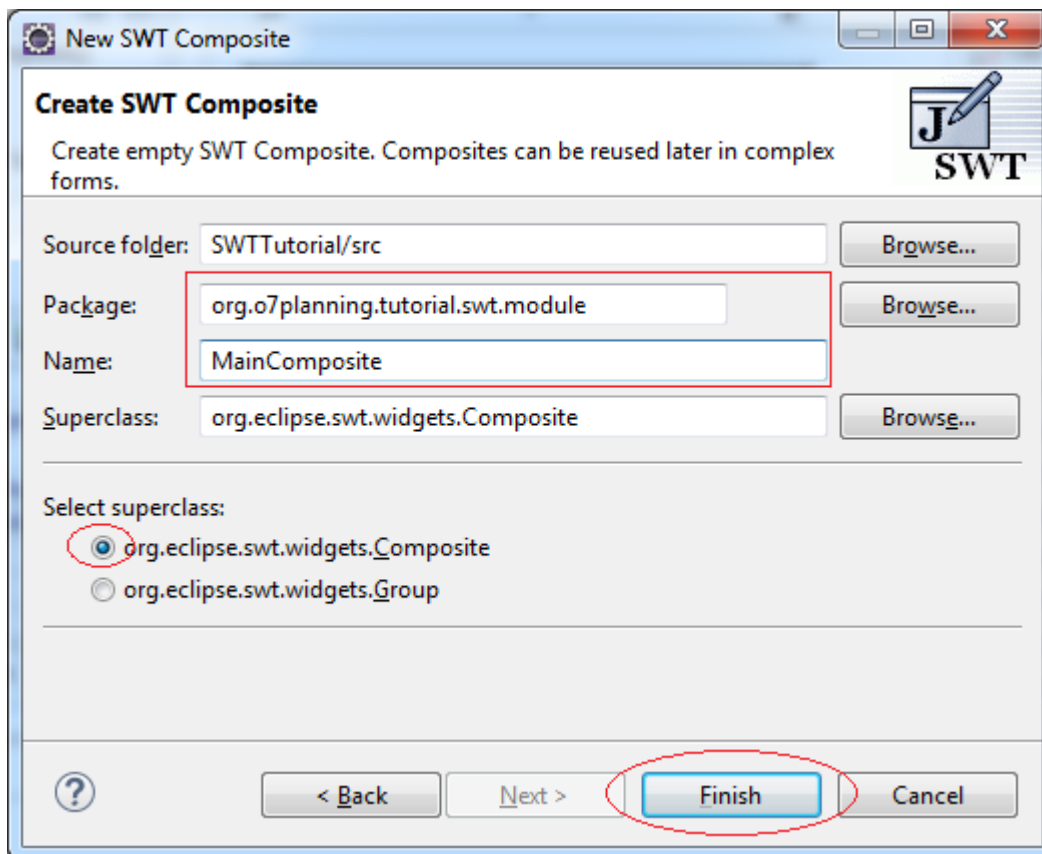
```
// Disable the check that prevents subclassing of SWT components
```

```
}
```

```
}
```

## MainComposite

Similarly create class MainComposite:

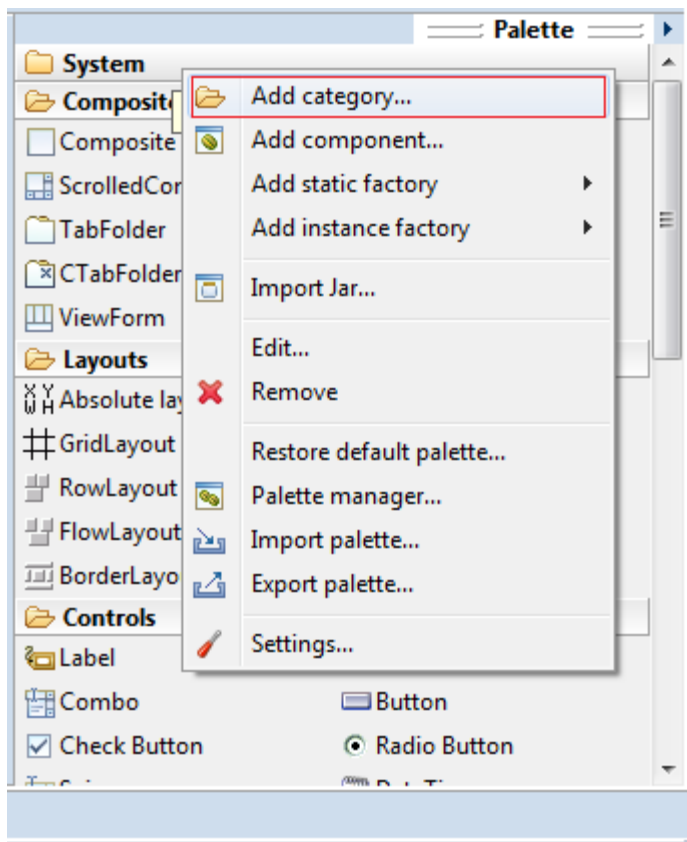


## Register TopComposite & BottomComposite into Palette

Right-click the Palette and choose Add category ...

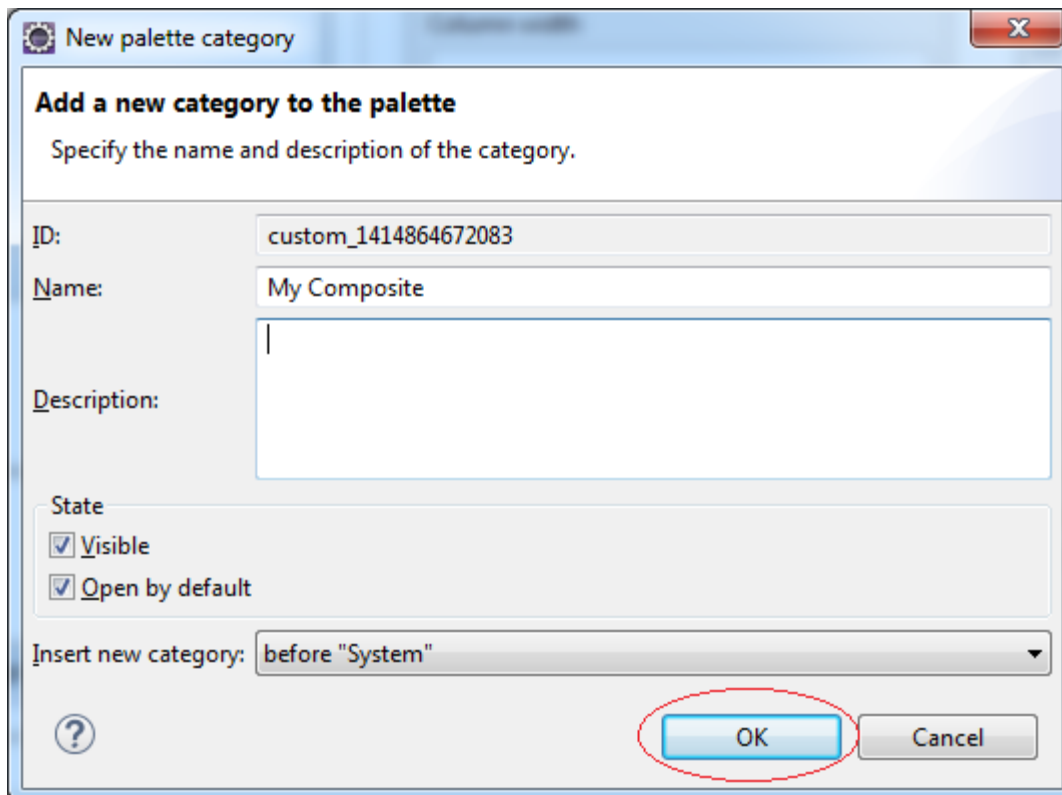


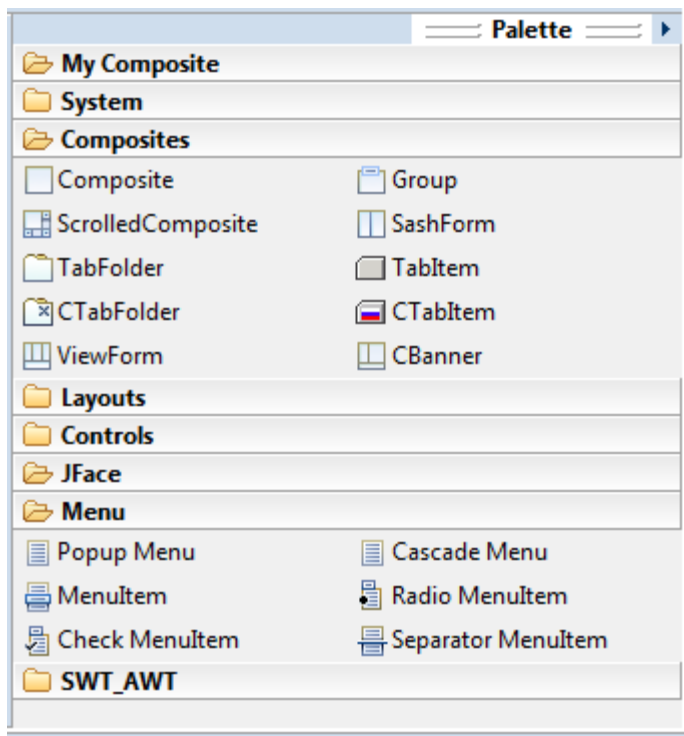




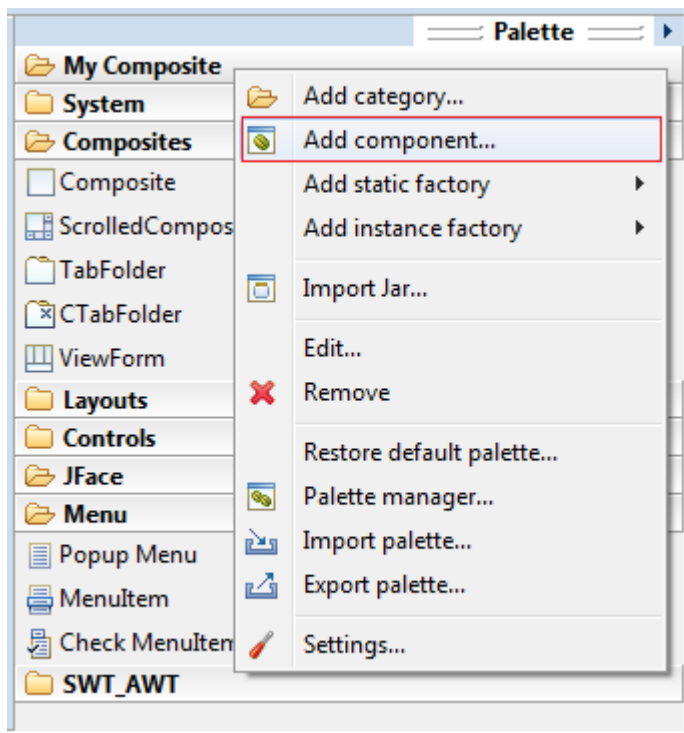
Named Pallete Category:

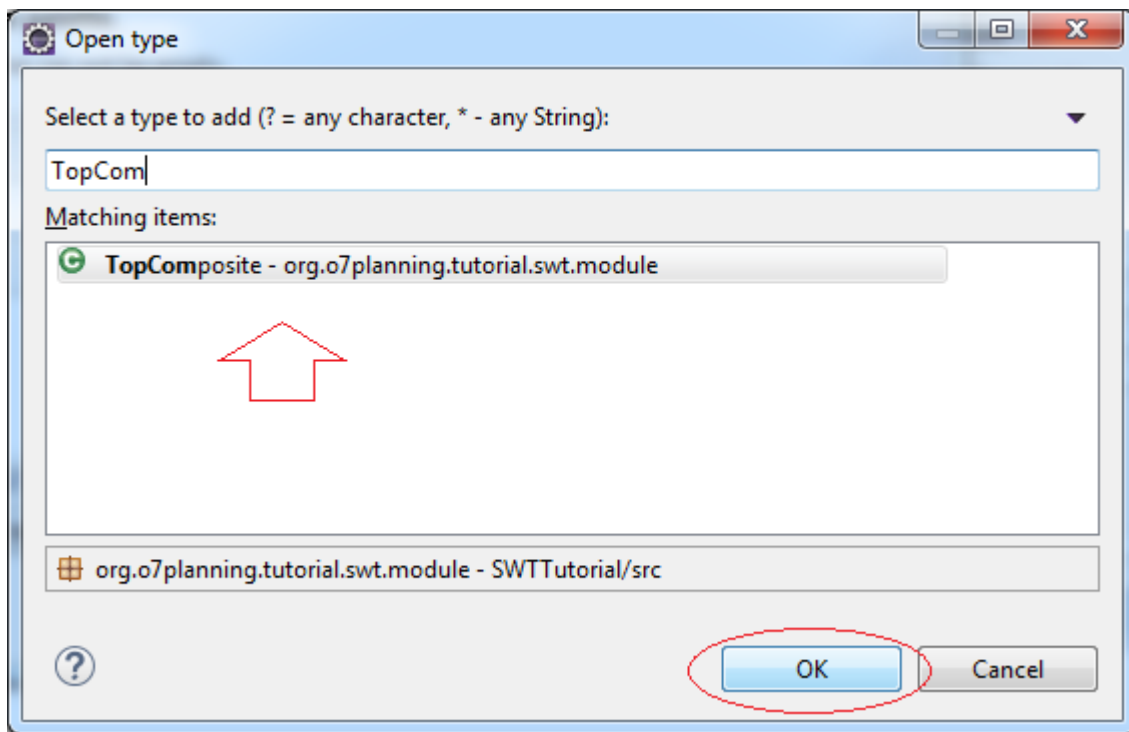
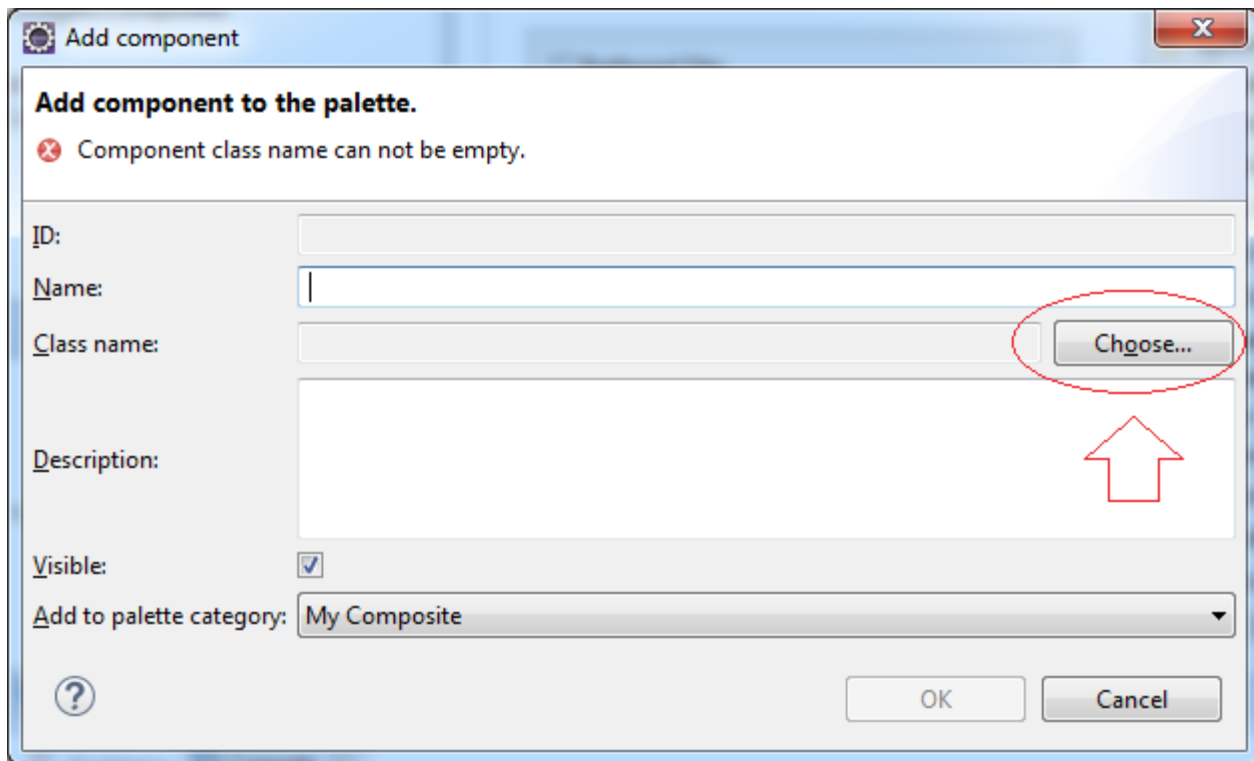
- My Composite

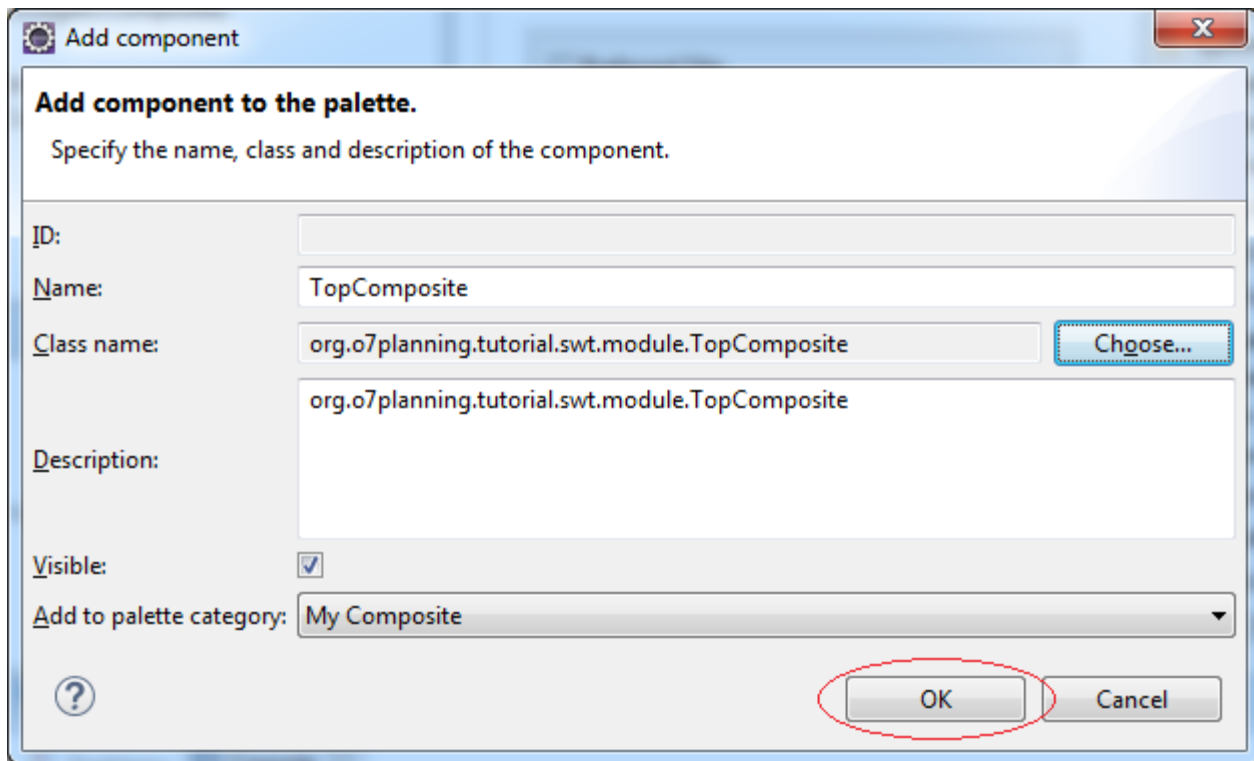




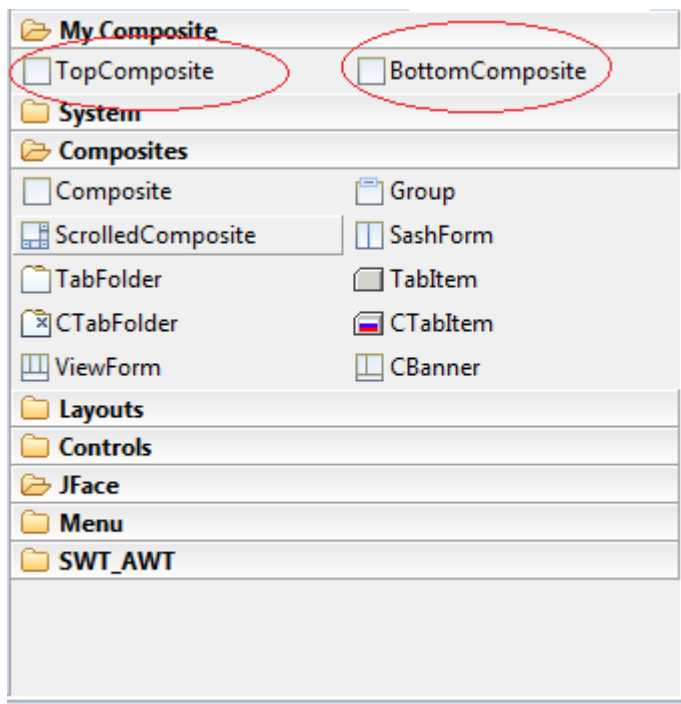
Right-click on the Category Palette "My Composite" to add TopComposite & BottomComposite.







Similarly add **BottomComposite** to "*My Composite*" catalog.



Now **TopComposite** & **BottomComposite** are easily drag and drop on the other Composite.

**MainComposite.java**

```
package org.o7planning.tutorial.swt.module;
```



```
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;

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() {
    }
}
```

## 12- Using event handlers



SWT event handling is very simple with the support of **WindowBuilder**.

### Examples:

- File/New/Other...



o7planning.org

### Fanpages



Facebook



Twitter

### Websites



o7planning.org



devstory.net



codestory.de



betacode.net



openplanning.net

### About Us

The website was created in March 2014 by a group of programmers and authors from Vietnam. Currently, the project supports 5 languages, including English, French, German, Russian and Vietnamese.

