



All Tutorials

Document Categories ▼

Java Desktop Application Programming using SWT


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

Quick Link



- 1- Introduction
- 2- RCP (Rich Client Platform)
- 3- The settings required before starting
- 4- Some concepts of SWT.
 - 4.1- Display & Shell
 - 4.2- SWT Widgets
- 5- Create RCP Plugin Project
- 6- First Example
- 7- Using WindowBuilder
- 8- SWT Widget
 - 8.1- Overview
 - 8.2- Widgets can contain other widgets (Container)
 - 8.3- Controls
- 9- SWT Layout
 - 9.1- What is Layout?
 - 9.2- Online Example

- What is needed to get started with Java?
- Guide to Installing and Configuring

8.2-  Widgets can contain other widgets (Container)

8.3-  Controls

9-  SWT Layout

9.1-  What is Layout?

9.2-  Online Example

9.3-  FillLayout

9.4-  RowLayout

9.5-  GridLayout

9.6-  StackLayout

9.7-  Combine Layout

10-  Write the class extending from the SWT widget

11-  Modular component interfaces

12-  JFace

1- Introduction

This document is based on:

- ***Eclipse 4.4 (LUNA)***

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

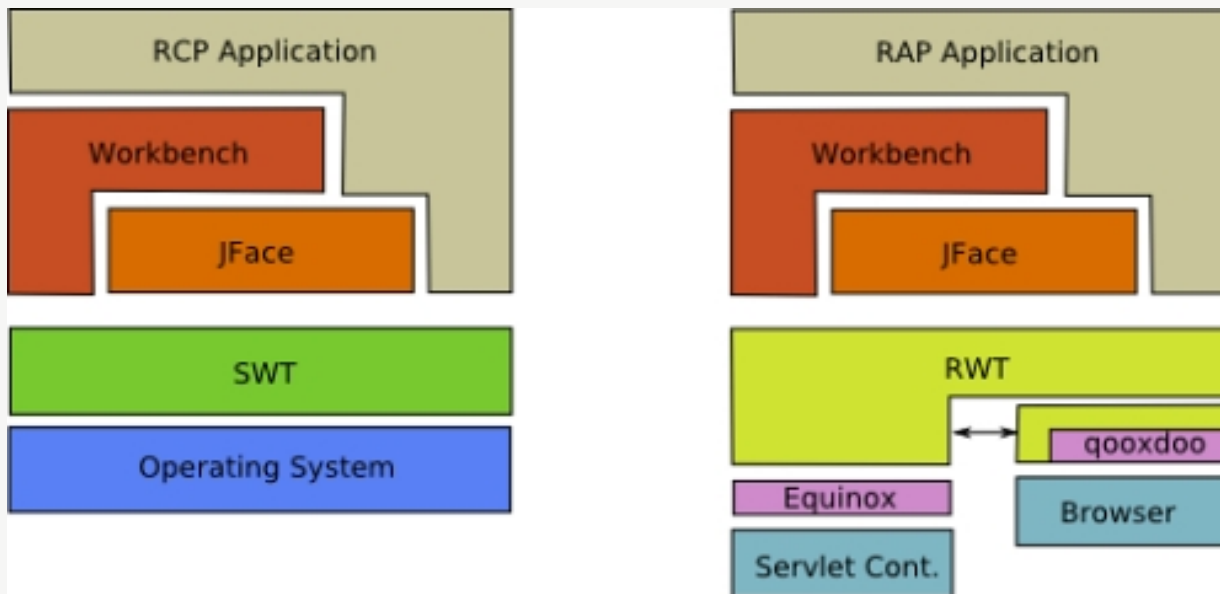
Java

- Guide to Installing and Configuring Eclipse
- Quick Learning Java for beginners
- JDK Javadoc in CHM format
- Inheritance and polymorphism in Java
- Access modifiers in Java
- Java Enum Tutorial
- Java Annotation Tutorial
- Comparison and sorting in Java
- Java String, StringBuffer and StringBuilder Tutorial
- Java Exception Handling Tutorial
- Java Generics Tutorial
- Java Collection Framework Tutorial
- Java IO Tutorial - Binary Streams
- Java IO Tutorial - Character Streams
- Java Date Time Tutorial
- Syntax and new features in Java 8
- Java Regular expression Tutorial
- Java Multithreading Programming Tutorial
- JDBC Driver Libraries for different types of database in Java
- Java JDBC tutorial
- Java Compression Tutorial

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.

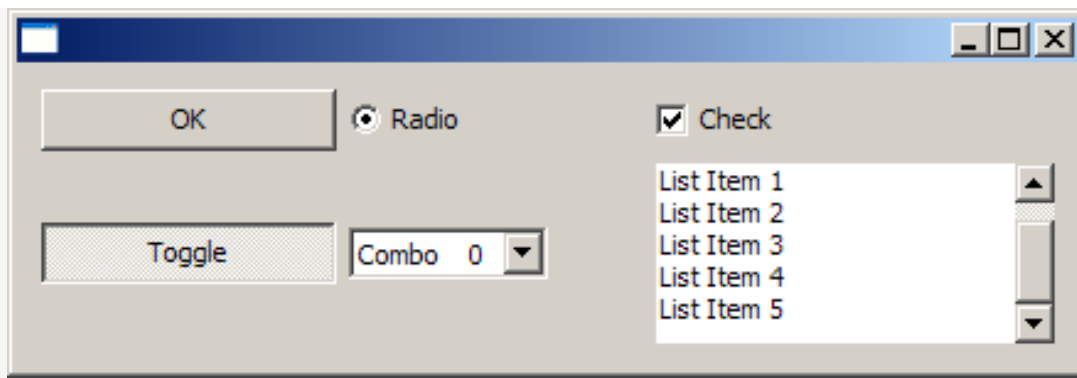


SWT:

- [Java reflection Tutorial](#)
- [Java remote method invocation - Java RMI Tutorial](#)
- [Java Socket Programming Tutorial](#)
- [Java Desktop Application Programming using SWT](#)
- [Eclipse JFace Tutorial](#)
- [Commons IO Tutorial](#)
- [Commons Logging Tutorial](#)

Advanced Java

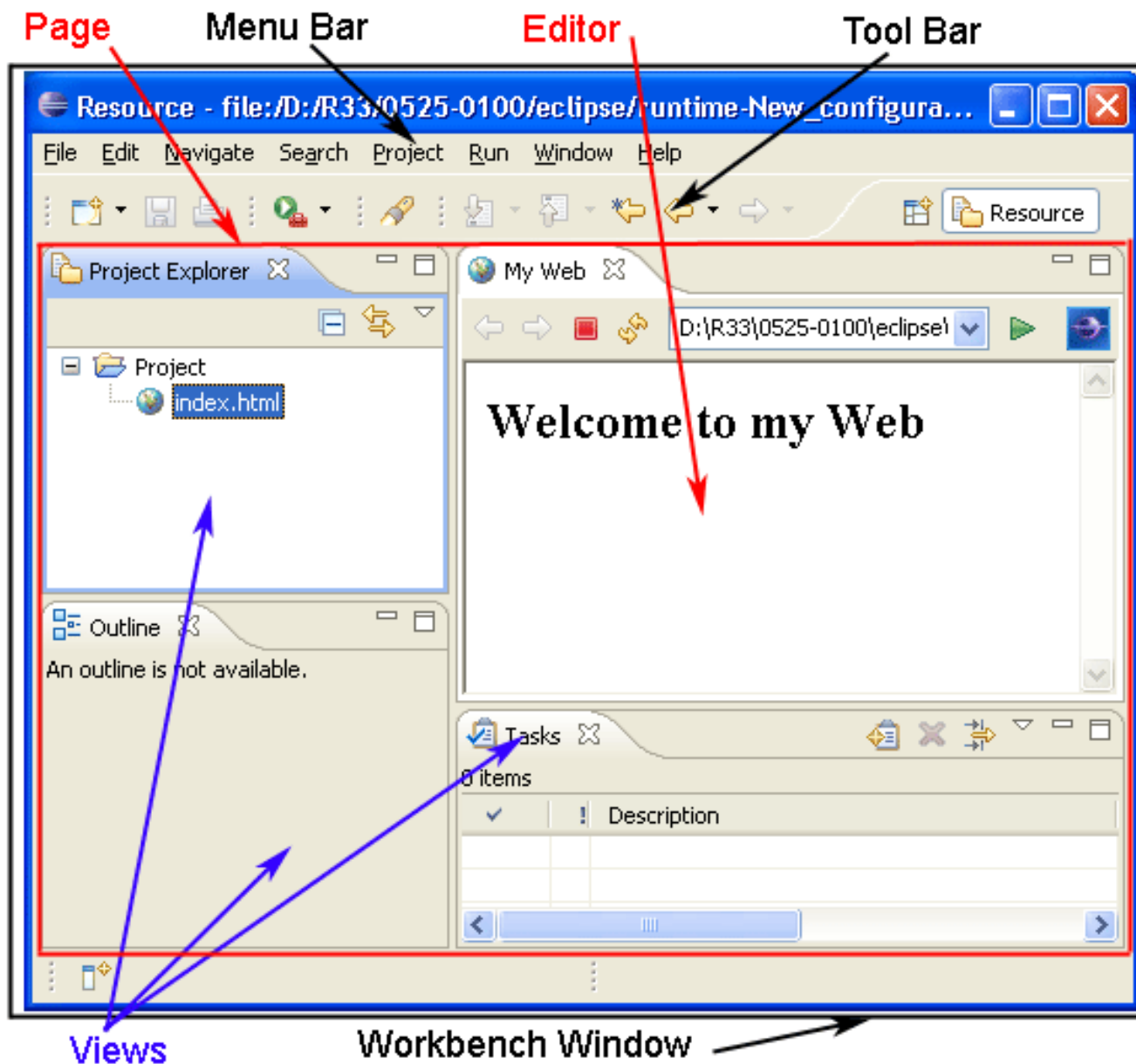
- [Java Programming for team using Eclipse and SVN](#)
- [Customize java compiler processing your Annotation \(Annotation Processing Tool\)](#)
- [Java Aspect Oriented Programming Tutorial with AspectJ \(AOP\)](#)
- [Install Maven into Eclipse](#)
- [Maven Tutorial for Beginners](#)
- [Maven Manage Dependencies](#)



Workbench Application:

- Setup a Multiple Module Project using Maven
- Install JBoss Tools into Eclipse
- Java Hibernate Tutorial for Beginners
- Using Hibernate Tools generate entity classes from Tables

Server-side Programming



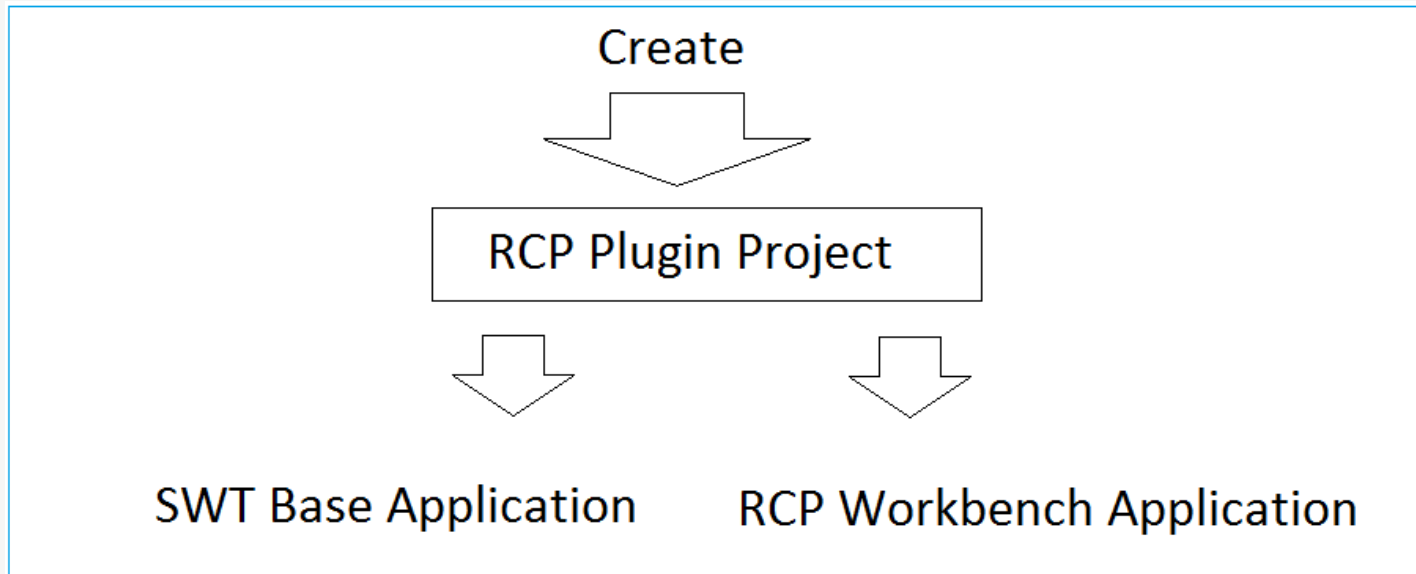
- Configuring Tomcat Server in Eclipse
- Guide to installing and configuring Glassfish Web Server
- Java Servlet Tutorial for Beginners
- Java Servlet Filter Tutorial
- Java JSP Tutorial for Beginners
- Install Web Tools Platform into Eclipse
- Create a Simple Web Application Using Servlet, JSP and JDBC
- Using Google reCAPTCHA with Java Web Application
- Run Maven Web Application in Tomcat Maven Plugin
- Run Maven Web Application in Jetty Maven Plugin
- Struts2 Tutorial for Beginners
- Spring MVC Tutorial for Beginners - Hello Spring 4 MVC
- Simple Login Web Application using Spring MVC, Spring Security and Spring JDBC
- Spring MVC and Hibernate Transaction Tutorial

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

Struts2 Framework

- 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 4.4 (Codes **LUNA**).

- <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.

- Struts2 Tutorial for Beginners
- Struts2 Tutorial for Beginners (Annotation Configuration)

Spring Framework

- Spring Tutorial for Beginners
- Spring MVC Tutorial for Beginners - Hello Spring 4 MVC
- Install Spring Tool Suite into Eclipse
- Configuring Static Resource and Resource Bundle in Spring MVC Tutorial
- Spring MVC File Upload Tutorial
- Spring JDBC Tutorial
- Simple Login Web Application using Spring MVC, Spring Security and Spring JDBC
- Spring MVC and Velocity Tutorial
- Using Template in Spring MVC with Apache Tiles
- Spring MVC and Spring JDBC Transaction Tutorial
- Spring MVC and Hibernate



Eclipse IDE for Java EE Developers, 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



Eclipse IDE for Java Developers, 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:

- <http://o7planning.org/web/fe/default/en/document/5699/install-windowbuilder-into-eclipse>



4- Some concepts of SWT. ⚓

4.1- Display & Shell ⚓

Eclipse Technology

- Install AspectJ development tools into Eclipse
- Java Aspect Oriented Programming Tutorial with AspectJ (AOP)
- How to get the open source Java libraries as OSGi(s)
- Install Tycho into Eclipse
- Java OSGi Tutorial for Beginners
- OSGi and AspectJ integration
- Building OSGi project with Maven and tycho
- Install WindowBuilder into Eclipse
- Java Desktop Application Programming using SWT
- Eclipse JFace Tutorial
- Install e4 Tools Developer Resources into Eclipse
- Eclipse RCP 4 Tutorial for Beginners - E4 Workbench Application
- Guide to Packing and Deploying Desktop Application SWT/RCP

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

A `org.eclipse.swt.widgets.Shell` class represents a window.

The `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:

```
1  Display display = new Display();
2  Shell shell = new Shell(display);
3  shell.open();
4
5  // run the event loop as long as the window is open
6  while (!shell.isDisposed()) {
7      // read the next OS event queue and transfer it to a SWT event
8      if (!display.readAndDispatch())
9      {
10         // if there are currently no other OS event to process
11         // sleep until the next OS event is available
12         display.sleep();
13     }
14 }
15
16 // disposes all associated windows and their components
17 display.dispose();
```

- [Install RAP tools into Eclipse](#)
- [Install RAP e4 Tooling into Eclipse](#)
- [Install Eclipse RAP Target Platform](#)
- [Eclipse RAP Tutorial for Beginners - Basic Application](#)
- [Eclipse RAP Tutorial for Beginners - e4 Workbench Application](#)
- [Create Eclipse RAP Widget from ClientScripting-based widget](#)
- [Guide to packaging and deploying Eclipse RAP application](#)
- [Install GEF into Eclipse](#)
- [Eclipse RAP Tutorial for Beginners - Workbench Application \(Earlier e4\)](#)
- [Eclipse RCP 3 Tutorial for Beginners - Workbench Application](#)
- [Simple Eclipse RCP 3 Application - View and Editor integration](#)

Java API for HTML & XML

- [Parsing an XML File Using SAX](#)
- [JDOM2 Tutorial](#)
- [JAXB Tutorial](#)
- [Jsoup Java Html Parser Tutorial](#)

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.



5- Create RCP Plugin Project

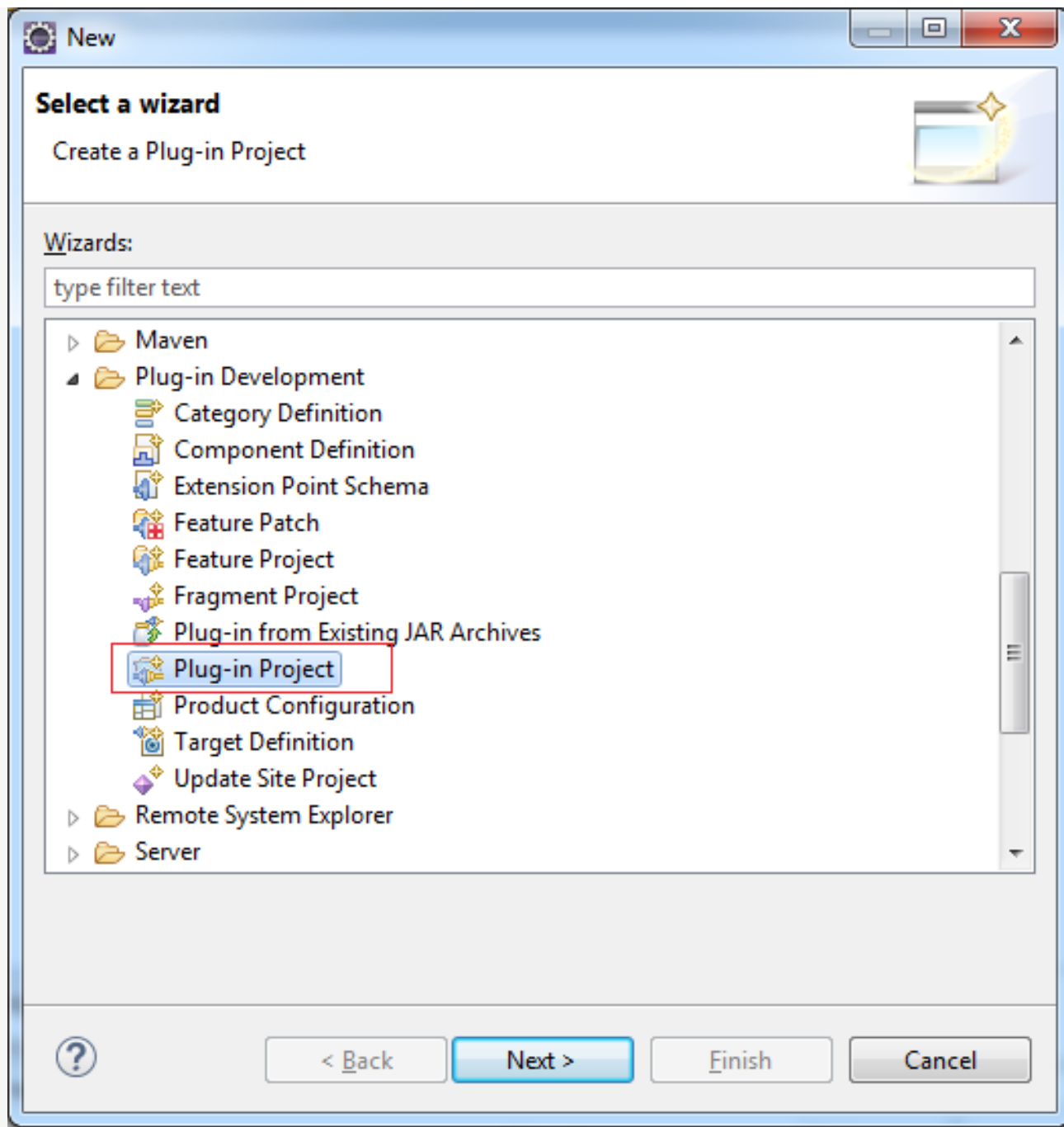
In Eclipse select **File/New/Other...**

Java Open source libraries

- [Skype Java API Tutorial](#)
- [Using Scribe OAuth Java API with Google OAuth 2 Tutorial](#)
- [JDOM2 Tutorial](#)
- [Jsoup Java Html Parser Tutorial](#)

Newest Documents

- [Configuring Android Simulator on Android Studio](#)
- [Android Tutorial for Beginners - Hello Android](#)
- [Maven Manage Dependencies](#)
- [Oracle APEX Tutorial, Tabular Form](#)
- [What is needed to get started with Android?](#)
- [Guide to Installing and Configuring Android Studio](#)



- [Java Generics Tutorial](#)
- [Using Template in Spring MVC with Apache Tiles](#)
- [Jsoup Java Html Parser Tutorial](#)
- [Using Google reCAPTCHA with Java Web Application](#)



Create a new plug-in project

Project name:

☒ Use default location

Location:

[Browse...](#)

Choose file system:

Project Settings

☒ Create a Java project

Source folder:

Output folder:

Target Platform

This plug-in is targeted to run with:

☒ Eclipse version:

☐ an OSGi framework:

Working sets

☐ Add project to working sets

Working sets:

[Select...](#)




< Back

Next >

Finish


Cancel

- 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


☐ This plug-in will make contributions to the UI

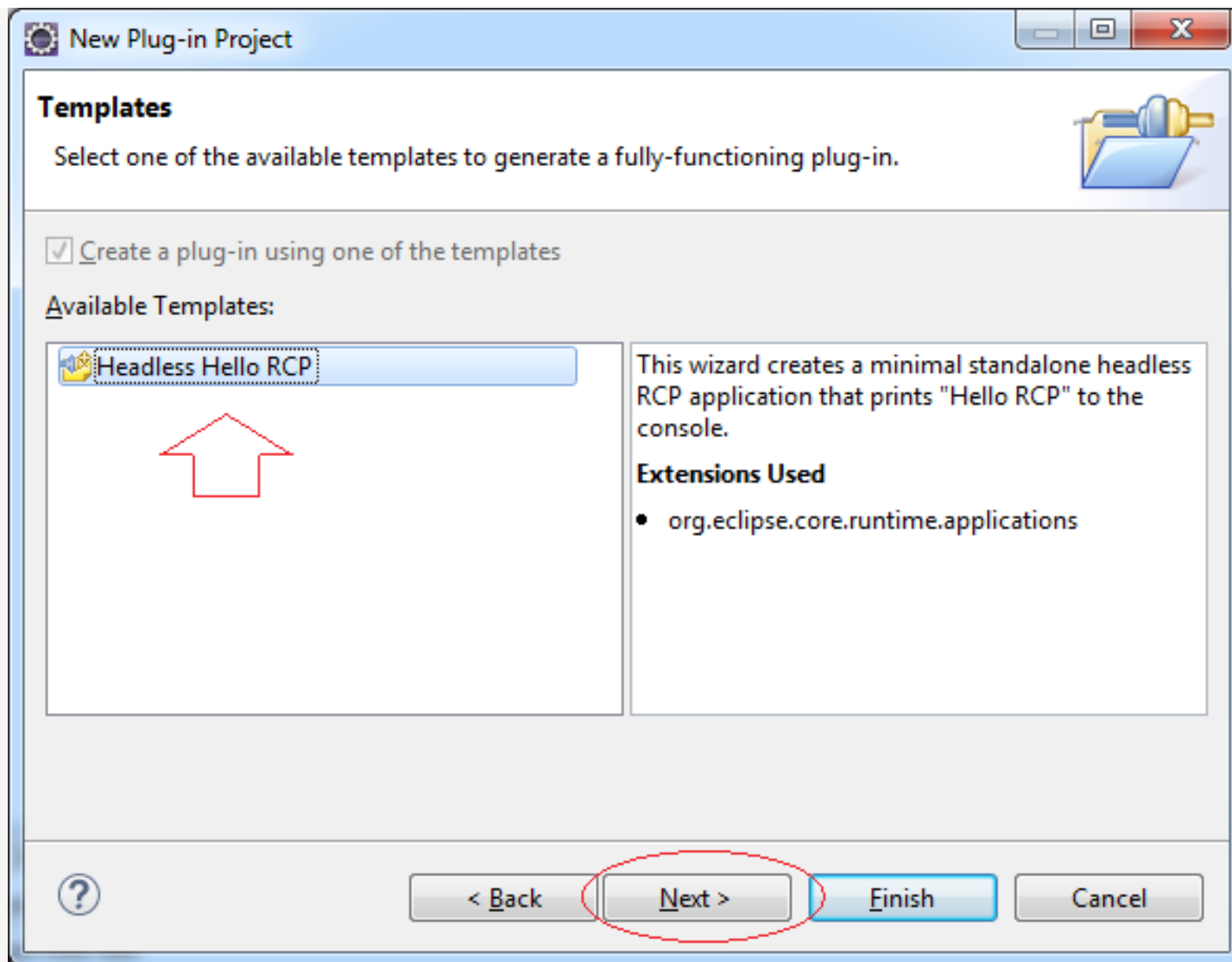
☐ Enable API analysis

Rich Client Application

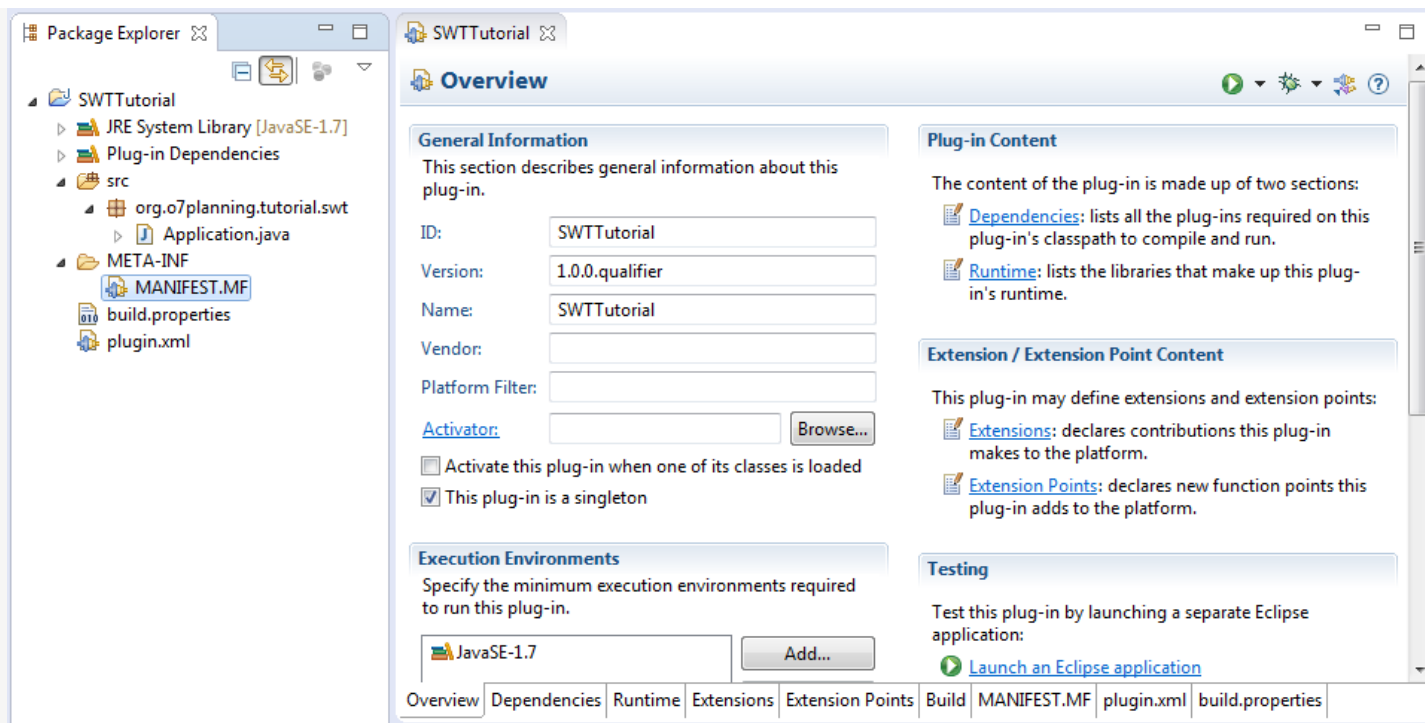
Would you like to create a 3.x rich client application? ☒ Yes ☐ No

2 





Created Project:



Add library swt: **org.eclipse.swt**

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

SWTTutorial

Dependencies

Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

org.eclipse.core.runtime

Add...

Remove

Up

Down

Properties...

Total: 1

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

Add...

Remove

Properties...

Total: 0

Automated Management of Dependencies

Overview

Dependencies

Runtime

Extensions

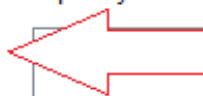
Extension Points

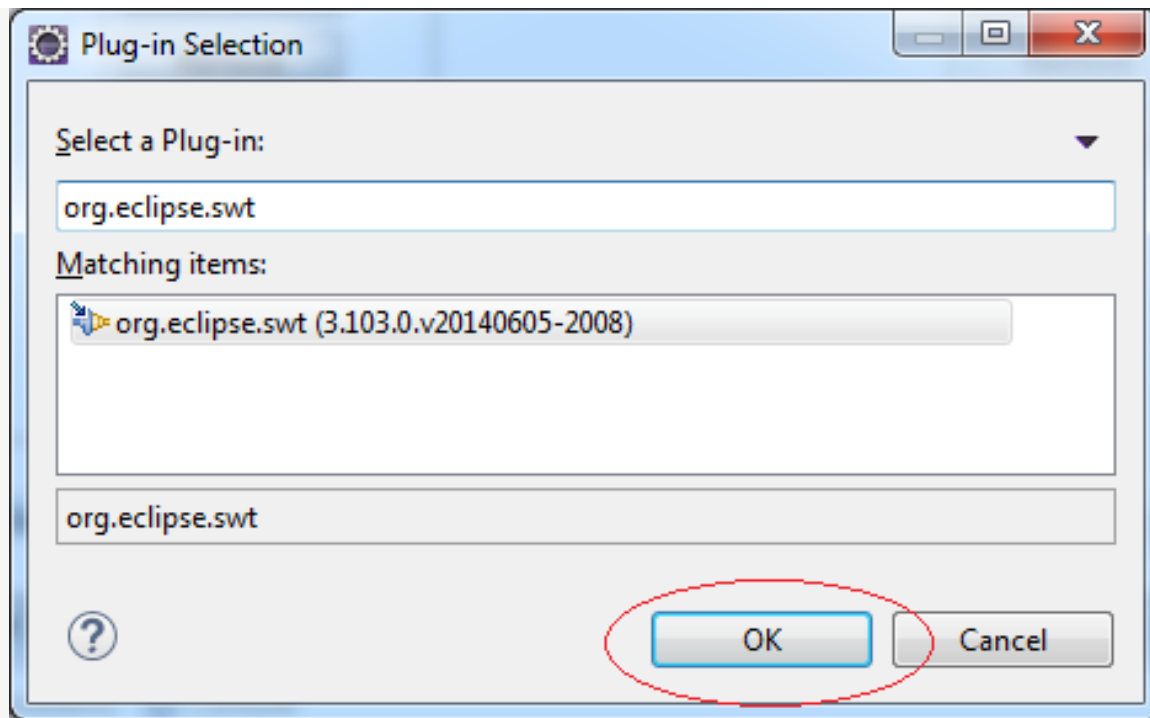
Build

MANIFEST.MF

plugin.xml

build.properties





6- First Example

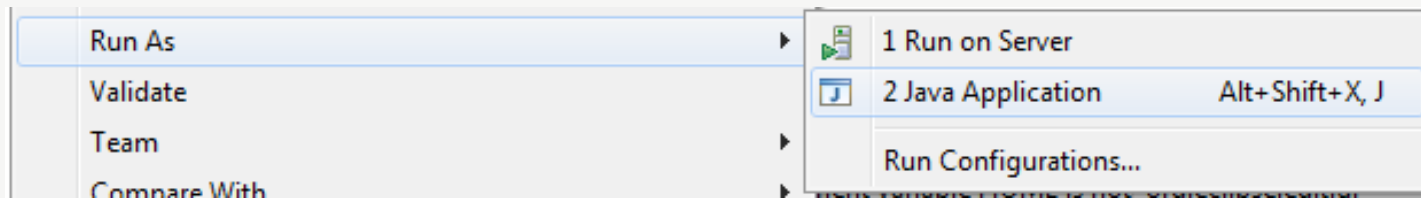
This is a simple example, not using the drag and drop tools WindowBuilder.

- **HelloSWT.java**

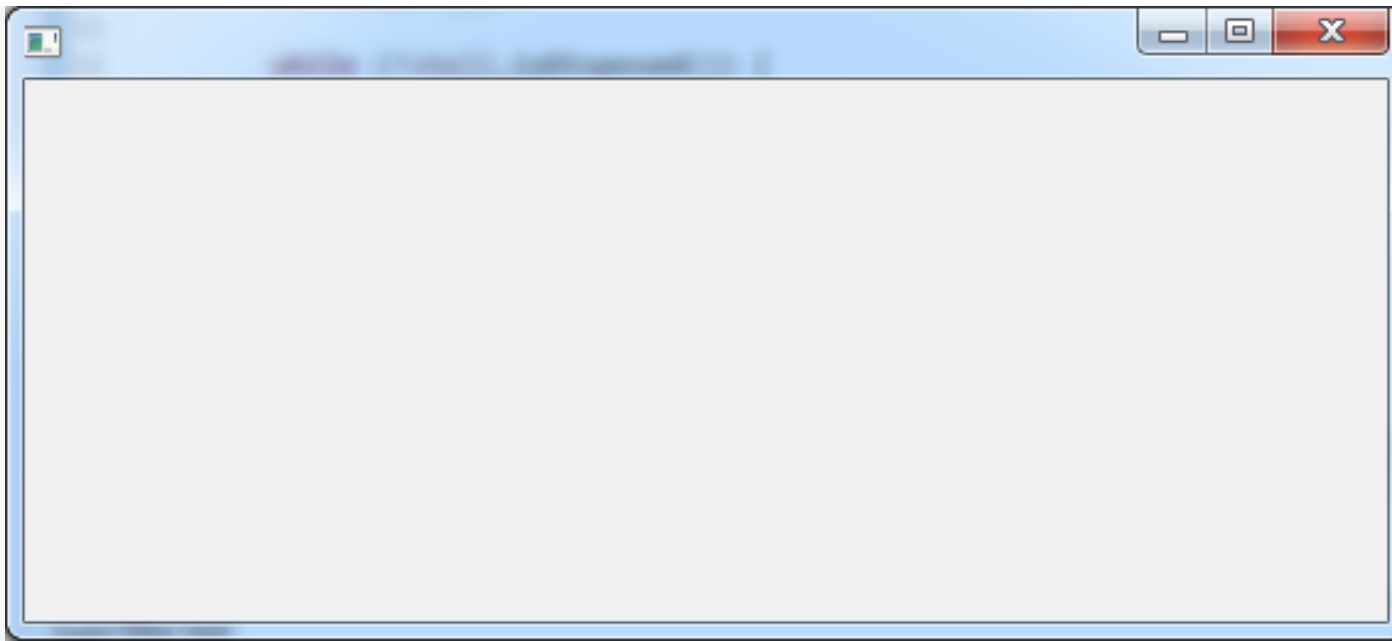
```
1 package org.o7planning.tutorial.swt.helloswt;
2
3 import org.eclipse.swt.widgets.Display;
4 import org.eclipse.swt.widgets.Shell;
5
6 public class HelloSWT {
7
8     public static void main(String[] args) {
```

```
9      // Create Display
10     Display display = new Display();
11     // Create Shell (Window) from display
12     Shell shell = new Shell(display);
13
14     shell.open();
15
16     while (!shell.isDisposed()) {
17         if (!display.readAndDispatch())
18             display.sleep();
19     }
20     display.dispose();
21 }
22 }
```

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



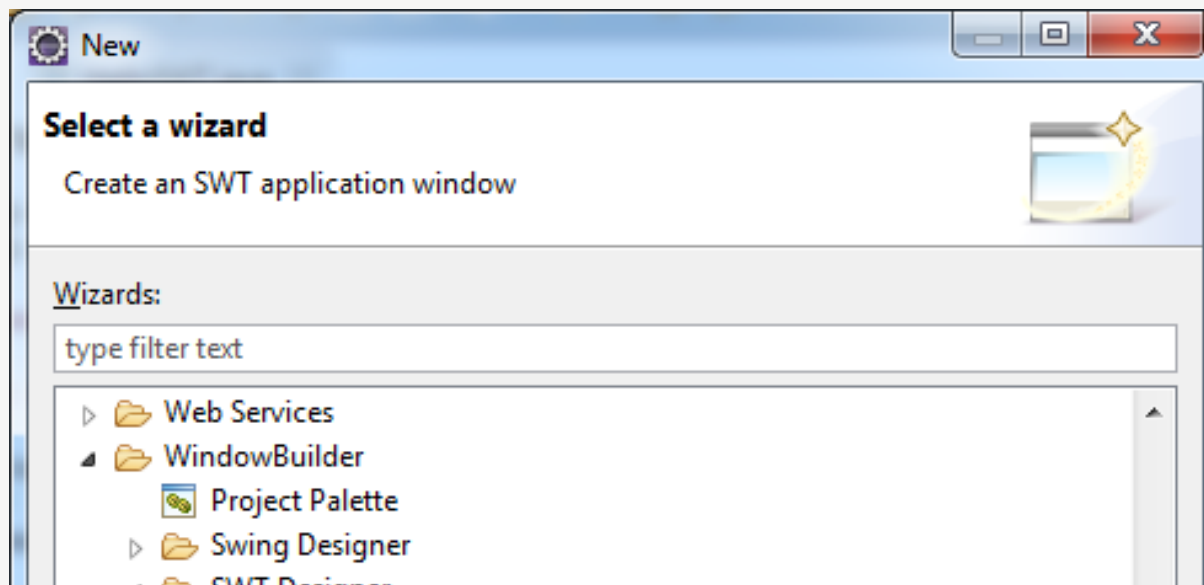
Result:

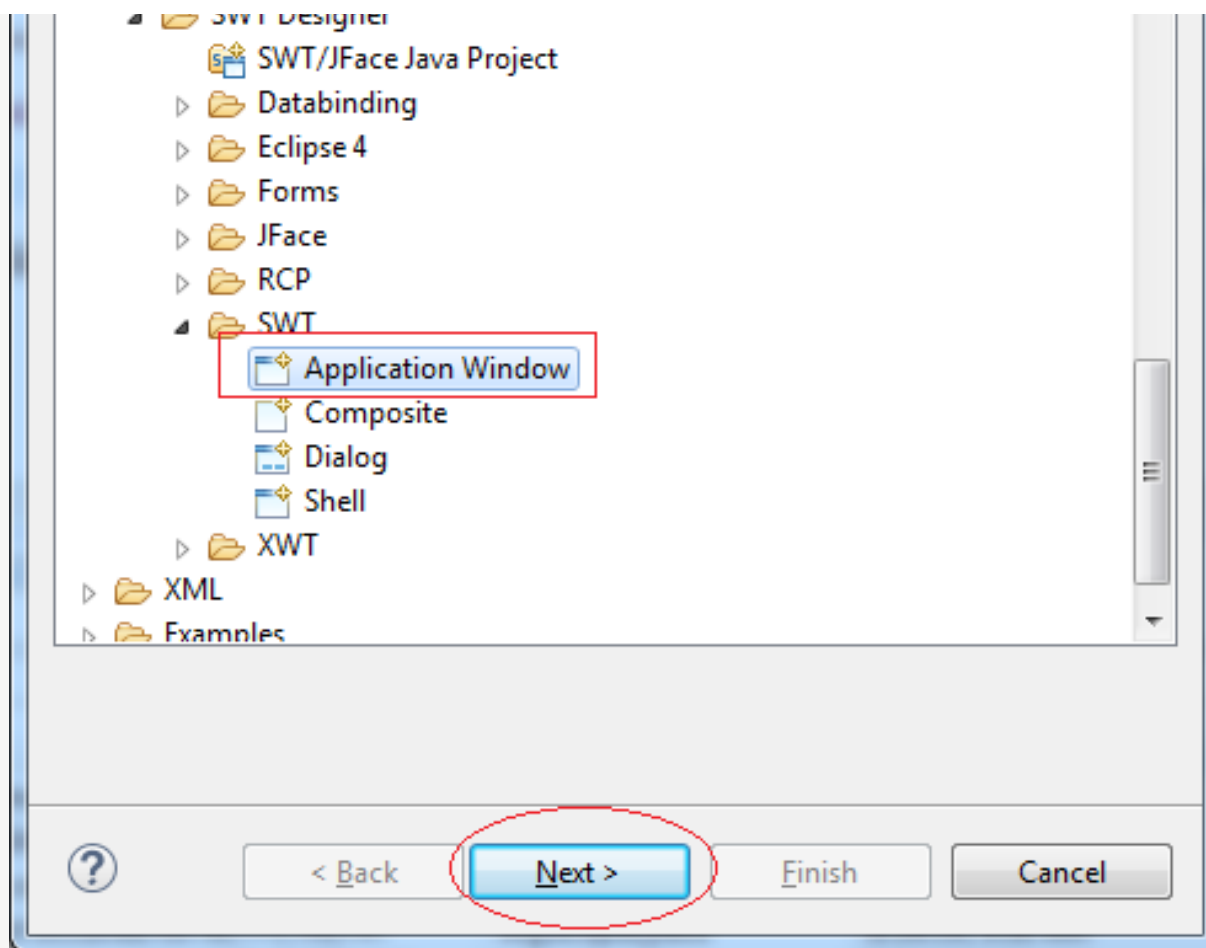



7- Using WindowBuilder

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

File/New/Other ..






 New SWT Application

Create SWT Application

Create a simple SWT application with Shell and event loop.




Source folder:

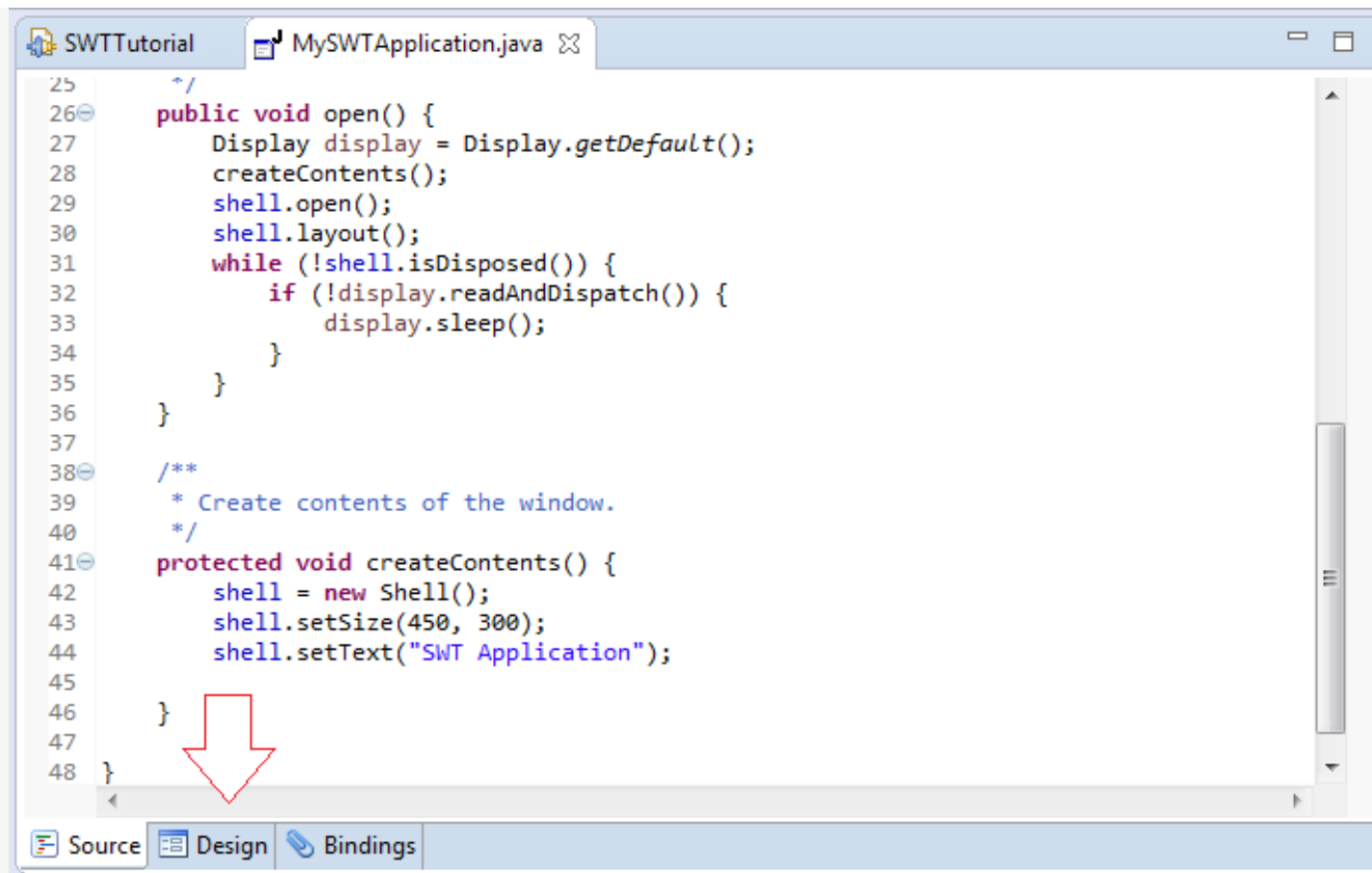
Package:

Name:

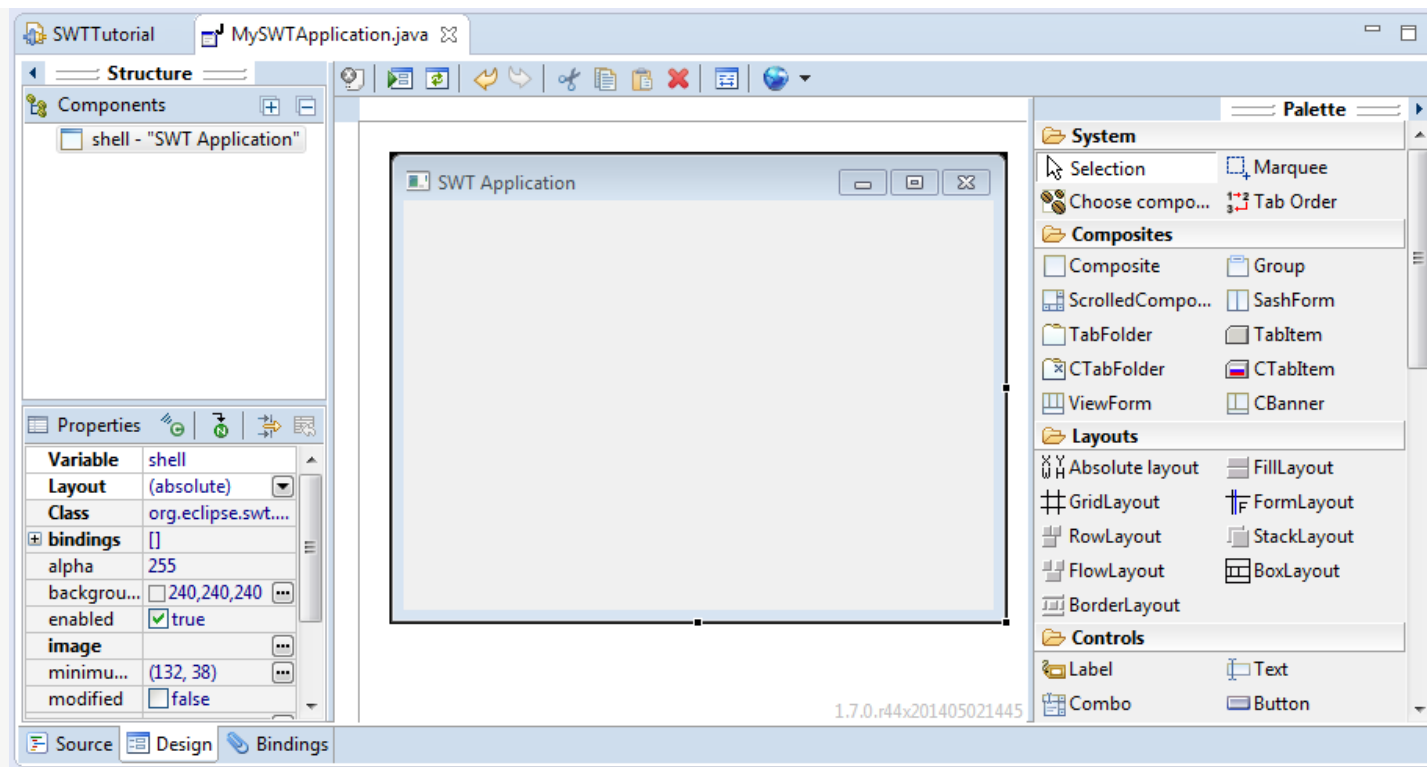
Create contents in:

- ☒ protected createContents() method
- ☐ public open() method
- ☐ public static main() method





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

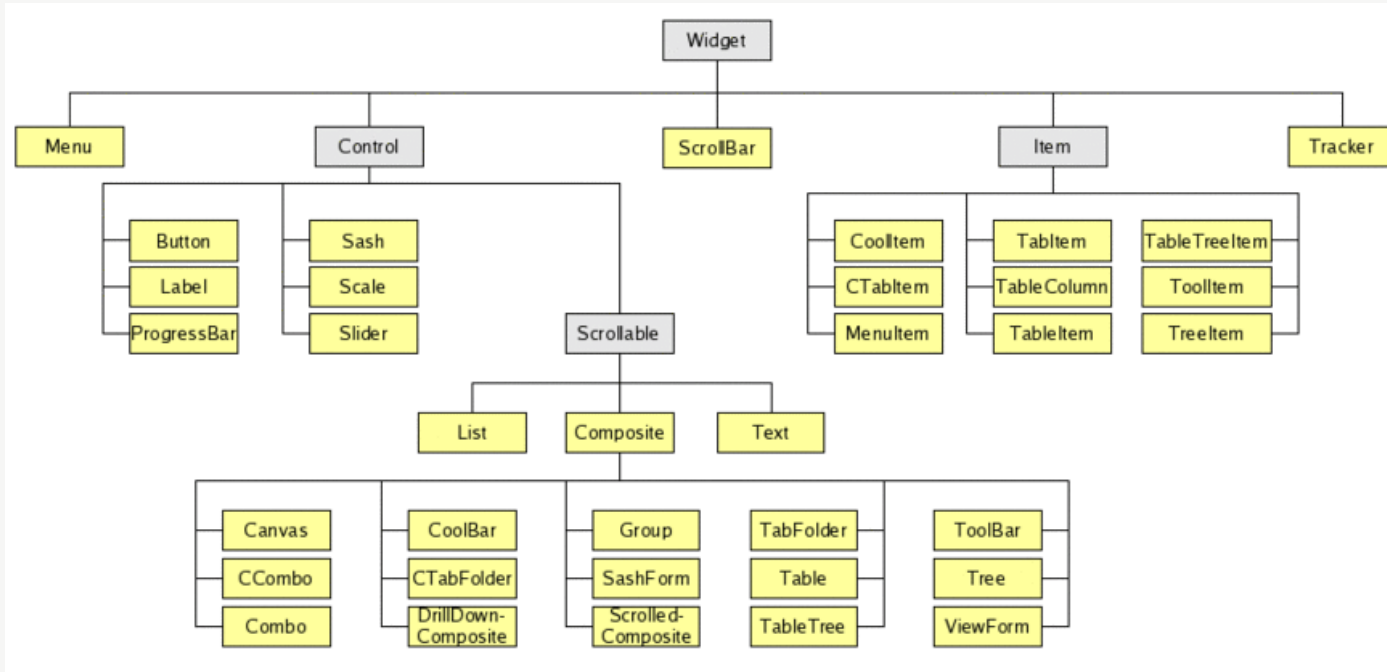


You can watch the video below:

8- SWT Widget

8.1- Overview

This is the hierarchy of widgets in SWT.

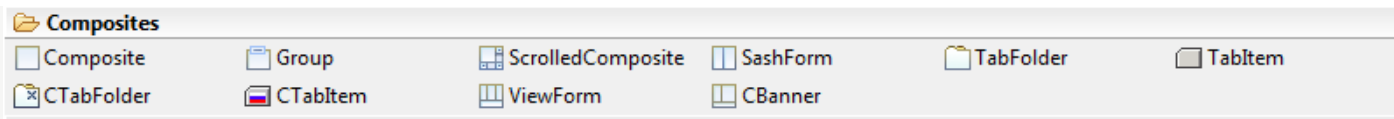


You can view the demo of the Control at the link below, it's RWT Control, but they are essentially the same as SWT control.

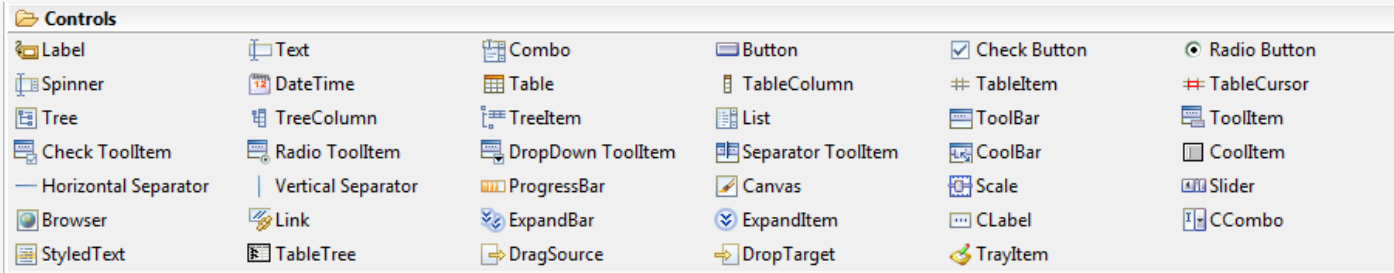
Demo:

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

8.2- Widgets can contain other widgets (Container)



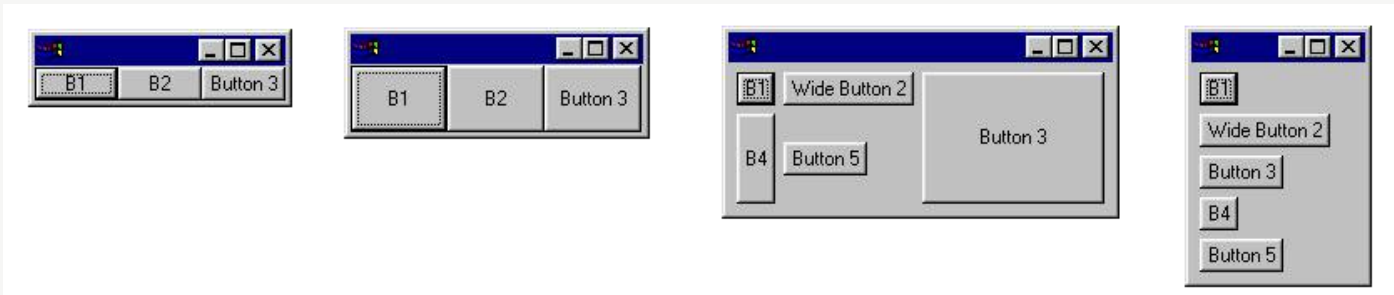
8.3- Controls



9- SWT Layout

9.1- What is Layout?

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

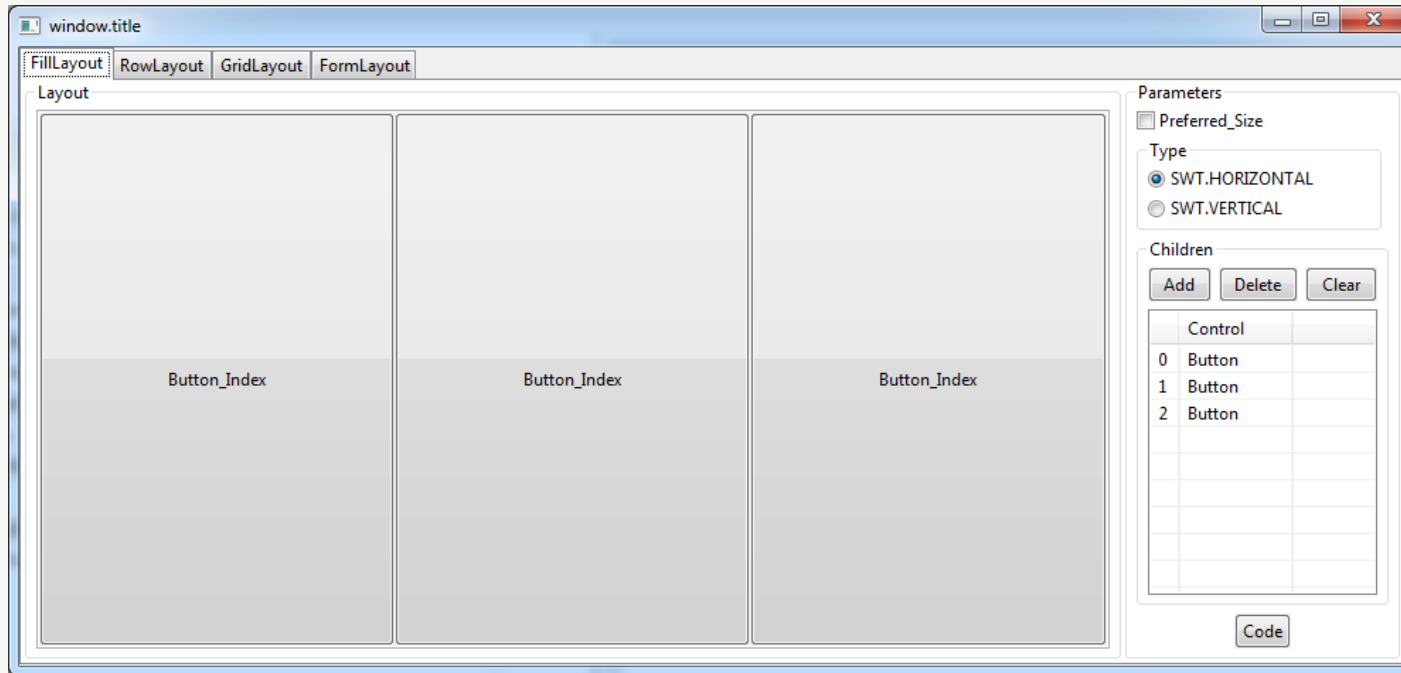


The standard layout classes in the SWT library are:

- *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




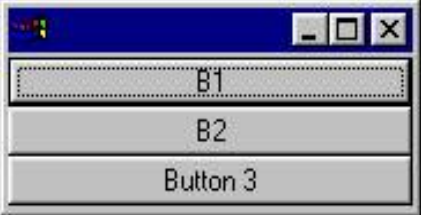
This is an example online, allowing you to see how the exercise of the Layout.



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.

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

	Initial	After resize
fillLayout.type = SWT.HORIZONTAL (default)	 A small window titled 'SWT' with a blue title bar. It contains three buttons labeled 'B1', 'B2', and 'Button 3' arranged horizontally.	 The same window after being resized horizontally. The buttons 'B1', 'B2', and 'Button 3' remain in their horizontal arrangement, but the window is wider.
fillLayout.type = SWT.VERTICAL	 A small window titled 'SWT' with a blue title bar. It contains three buttons labeled 'B1', 'B2', and 'Button 3' arranged vertically.	 The same window after being resized vertically. The buttons 'B1', 'B2', and 'Button 3' remain in their vertical arrangement, but the window is taller.

Video:

- **FillLayoutExample.java**

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

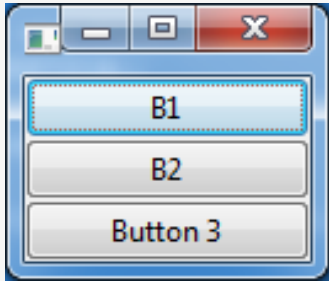
```
2
3 import org.eclipse.swt.SWT;
4 import org.eclipse.swt.layout.FillLayout;
5 import org.eclipse.swt.layout.RowLayout;
6 import org.eclipse.swt.widgets.Button;
7 import org.eclipse.swt.widgets.Composite;
8 import org.eclipse.swt.widgets.Display;
9 import org.eclipse.swt.widgets.Shell;
10
11 public class FillLayoutExample {
12
13     public static void main(String[] args) {
14         Display display = new Display();
15         final Shell shell = new Shell(display);
16         shell.setLayout(new FillLayout());
17
18         //
19         Composite parent = new Composite(shell, SWT.NONE);
20
21         FillLayout fillLayout= new FillLayout();
22         fillLayout.type= SWT.VERTICAL;
23
24         parent.setLayout(fillLayout);
25
26         Button b1 = new Button(parent, SWT.NONE);
27         b1.setText("B1");
28
29         Button b2 = new Button(parent, SWT.NONE);
30         b2.setText("B2");
31
32         Button button3 = new Button(parent, SWT.NONE);
33         button3.setText("Button 3");
34
35         // Windows back to natural size.
36         shell.pack();
37         //
38         shell.open();
39         while (!shell.isDisposed()) {
40             if (!display.readAndDispatch())
41                 display.sleep();
42         }
```

```

43         // tear down the SWT window
44         display.dispose();
45     }
46 }

```

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
wrap = true pack = true justify = false (defaults)		

wrap = false

(clips if not enough space)



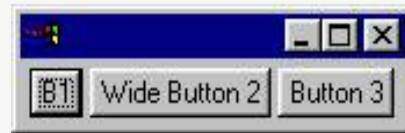
pack = false

(all widgets are the same size)



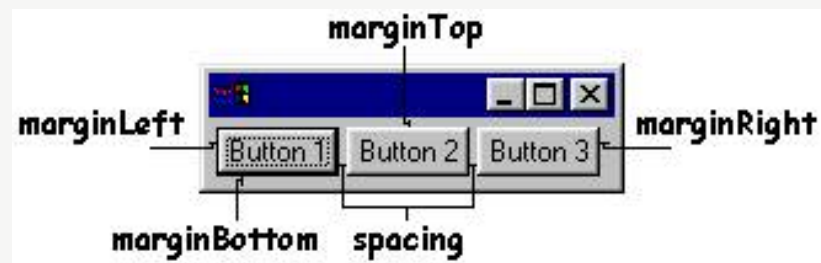
justify = true

(widgets are spread across the available space)



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:

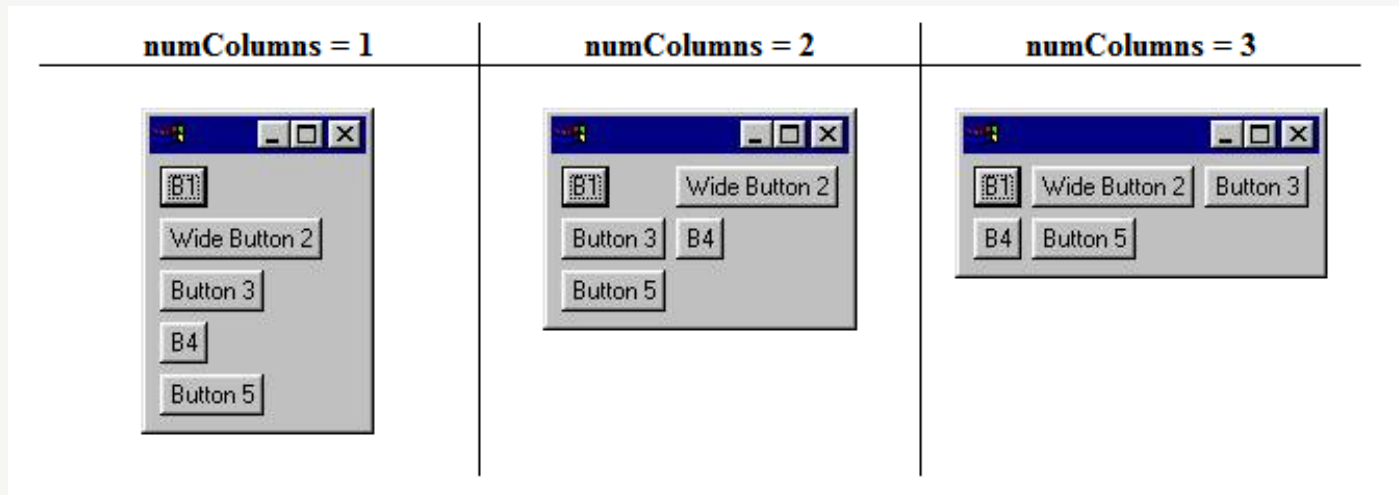
9.5- GridLayout

GridLayout is the most useful and powerful of the standard layouts, but it is also the most

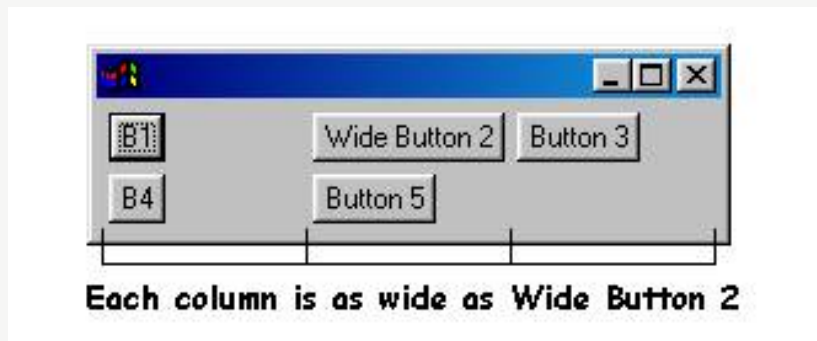
complicated. With a *GridLayout*, the widget children of a *Composite* are laid out in a grid. *GridLayout* has a number of configuration fields, and, like *RowLayout*, the widgets it lays out can have an associated layout data object, called *GridData*. The power of *GridLayout* lies in the ability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

The configuration of the GridLayout:

- NumColumns



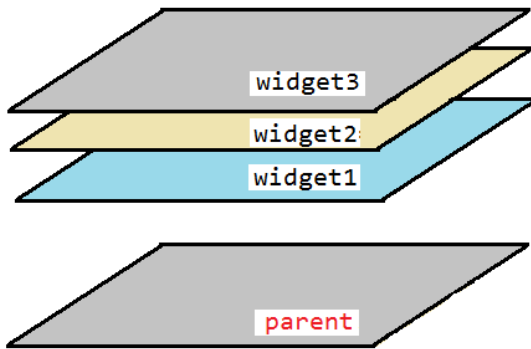
- MakeColumnsEqualWidth



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() 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.

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.

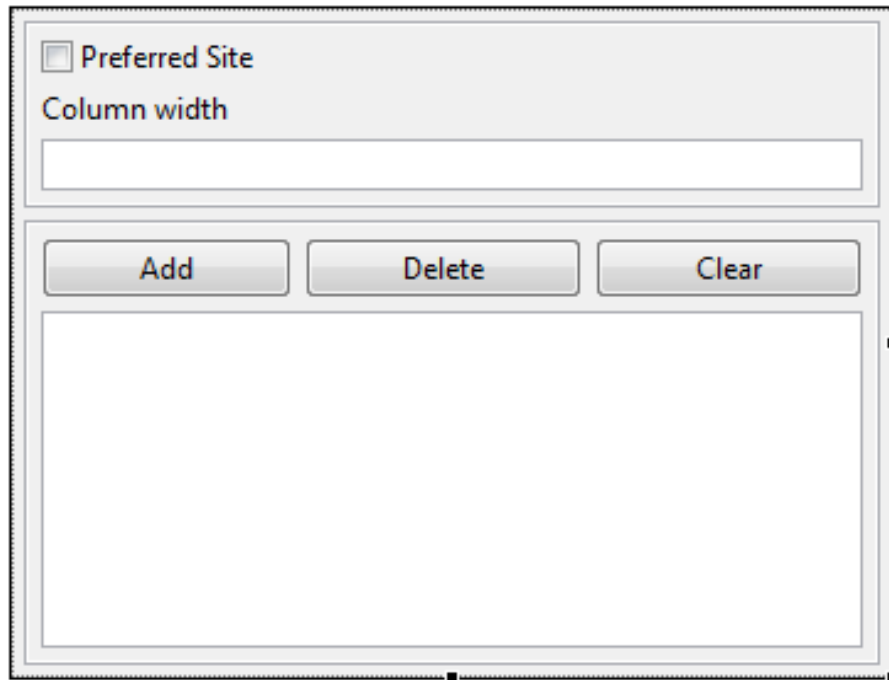
```
1  package org.o7planning.tutorial.swt.swtsubclass;
2
3  import org.eclipse.swt.widgets.Button;
4  import org.eclipse.swt.widgets.Composite;
5
6  public class MyButton extends Button {
7
8      public MyButton(Composite parent, int style) {
9          super(parent, style);
10     }
11
12     // You have to override this method.
13     @Override
14     protected void checkSubclass() {
15         // No need to do anything.
16     }
17
18
19 }
```

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.

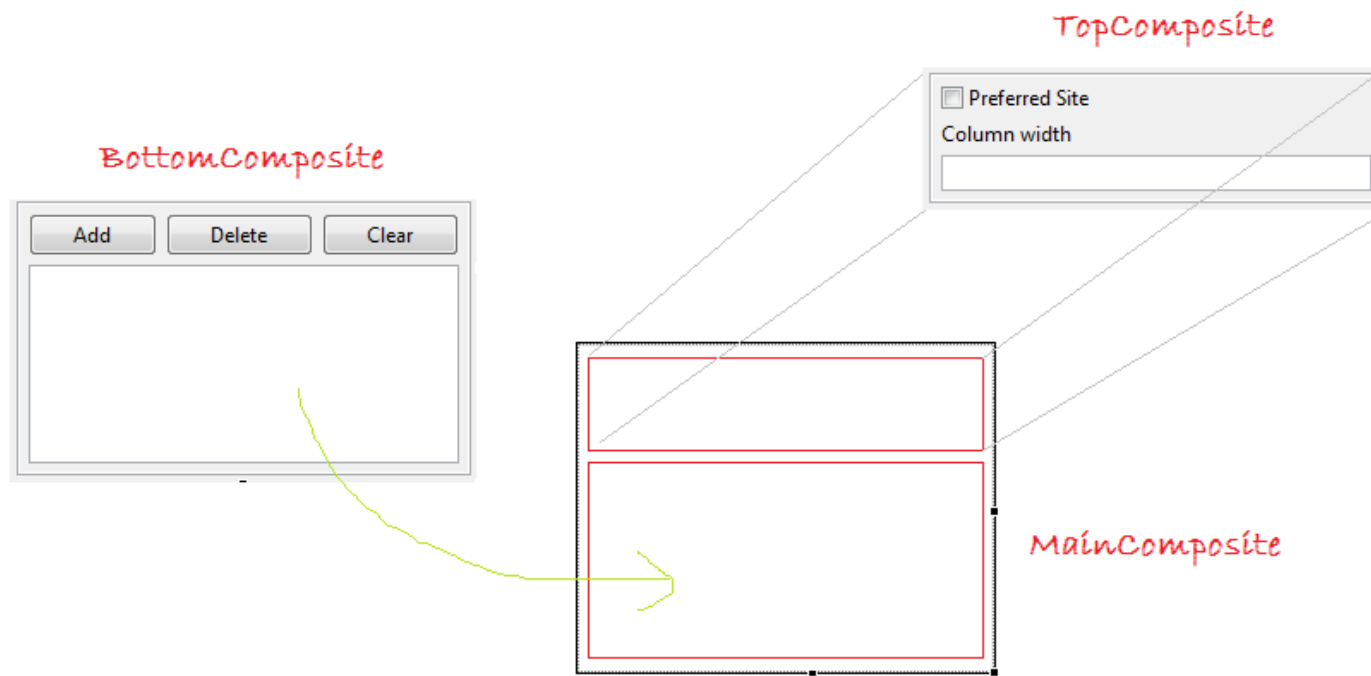
Please see the following interface, and we will try to split it up.

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)



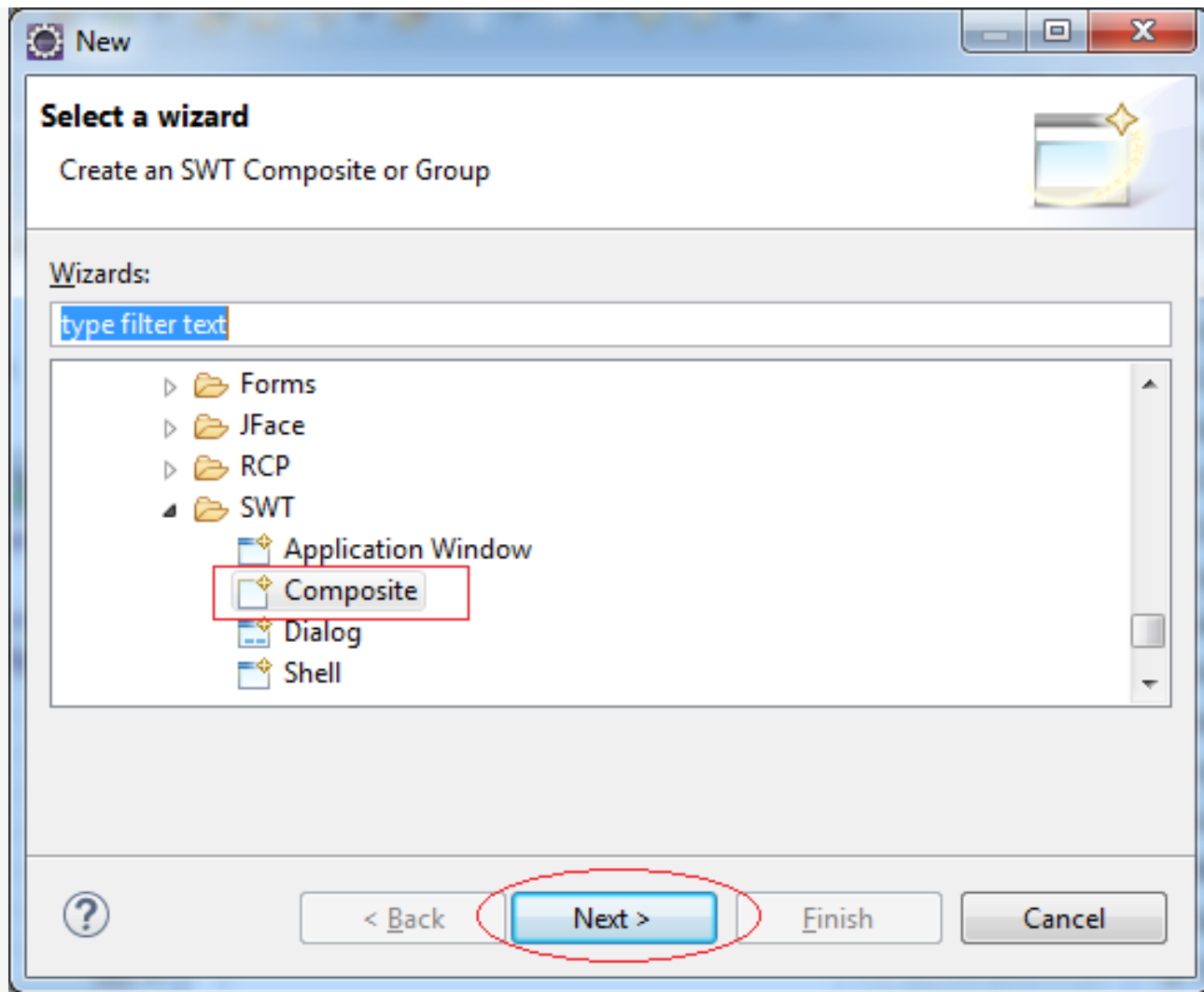
The image shows a web form with a light gray background. At the top left, there is a checkbox labeled "Preferred Site". Below this, the text "Column width" is followed by a text input field. Underneath the input field, there are three buttons: "Add", "Delete", and "Clear". Below the buttons is a large, empty rectangular area, likely intended for a list or table. The form has a simple, clean design with a thin border.

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



TopComposite

- File/New/Other...



New SWT Composite

Create SWT Composite

Create empty SWT Composite. Composites can be reused later in complex forms.

Source folder:

Package:

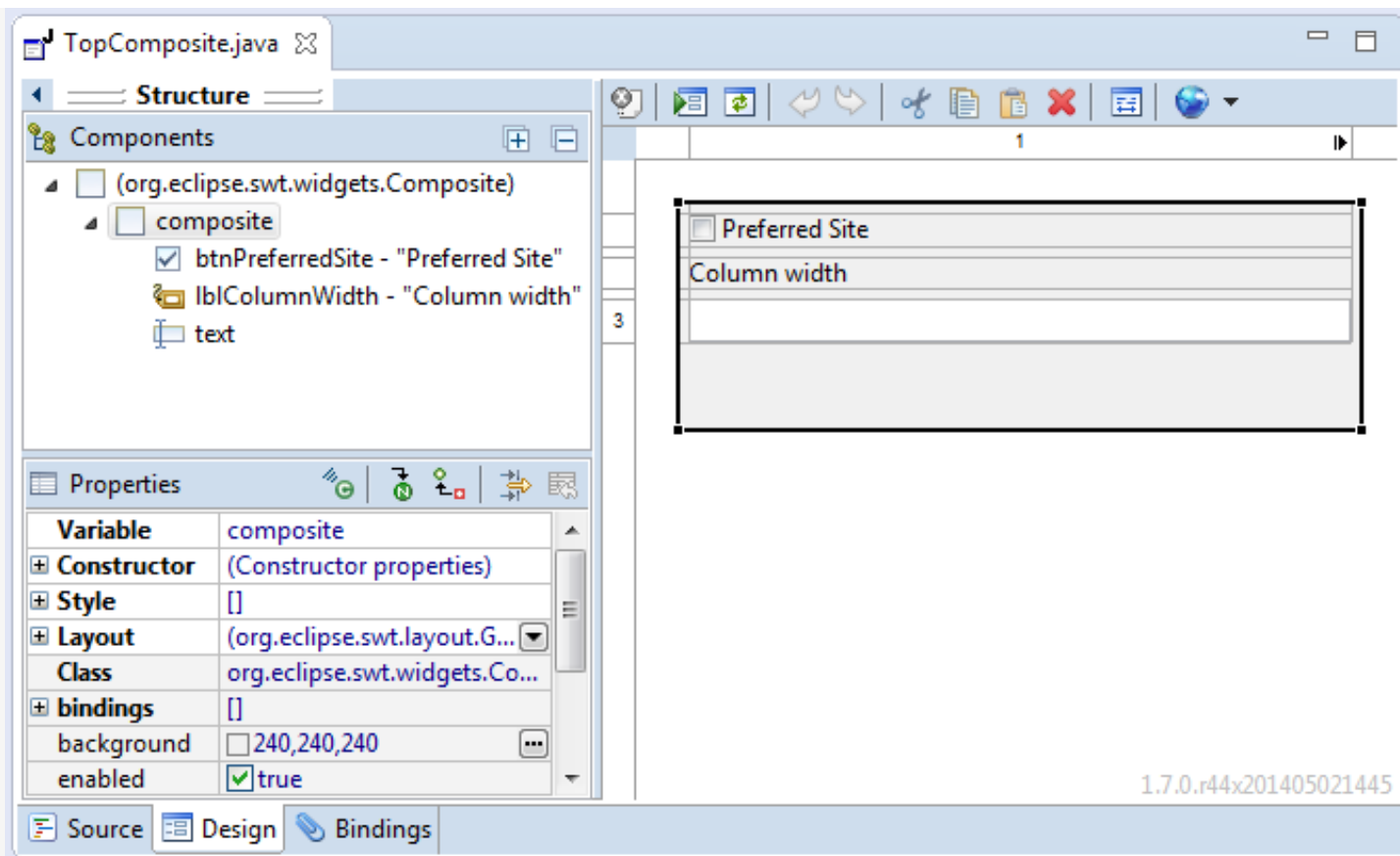
Name:

Superclass:

Select superclass:

- ☒ org.eclipse.swt.widgets.Composite
- ☐ org.eclipse.swt.widgets.Group

Interface Design for **TopComposite**.



- **TopComposite.java**

```
1 package org.o7planning.tutorial.swt.module;
2
3 import org.eclipse.swt.widgets.Composite;
4
5 public class TopComposite extends Composite {
6     private Text text;
7
8     /**
9      * Create the composite.
10     * @param parent
11     * @param style
```


```

12     */
13     public TopComposite(Composite parent, int style) {
14         super(parent, style);
15         setLayout(new FillLayout(SWT.HORIZONTAL));
16
17         Composite composite = new Composite(this, SWT.NONE);
18         composite.setLayout(new GridLayout(1, false));
19
20         Button btnPreferredSite = new Button(composite, SWT.CHECK);
21         btnPreferredSite.setText("Preferred Site");
22
23         Label lblColumnWidth = new Label(composite, SWT.NONE);
24         lblColumnWidth.setText("Column width");
25
26         text = new Text(composite, SWT.BORDER);
27         text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true,
28
29     }
30
31     @Override
32     protected void checkSubclass() {
33     }
34 }

```


BottomComposite

Similarly, create class **BottomComposite**

 New SWT Composite

Create SWT Composite

Create empty SWT Composite. Composites can be reused later in complex forms.



Source folder:


Package:

Name:

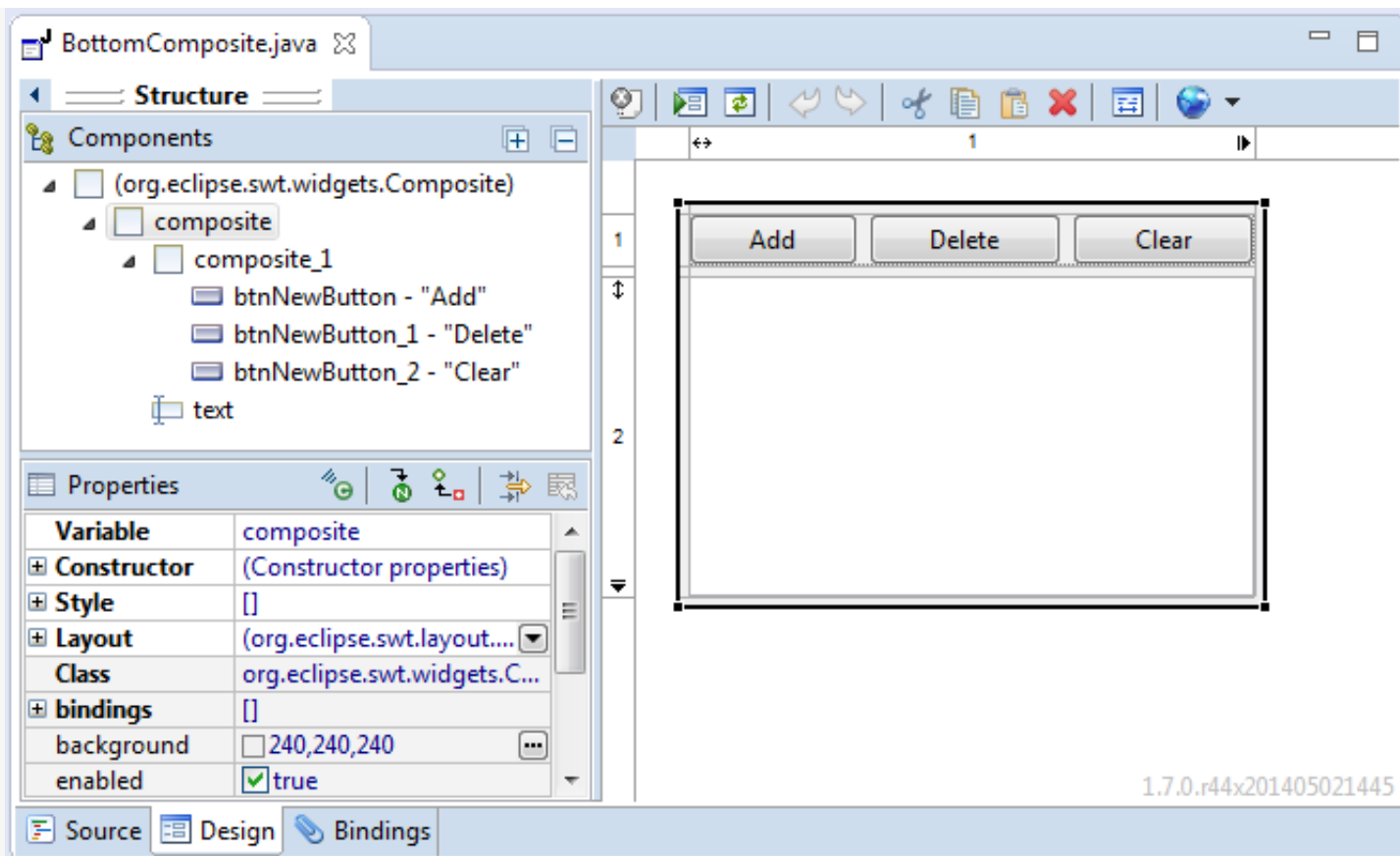
Superclass:

Select superclass:

- ☒ org.eclipse.swt.widgets.Composite
- ☐ org.eclipse.swt.widgets.Group



Interface Design for **BottomComposite**:



- BottomComposite.java

```
1 package org.o7planning.tutorial.swt.module;
2
3 import org.eclipse.swt.SWT;
4
5 public class BottomComposite extends Composite {
6     private Text text;
7
8     /**
9      * Create the composite.
10     * @param parent
11     * @param style
```

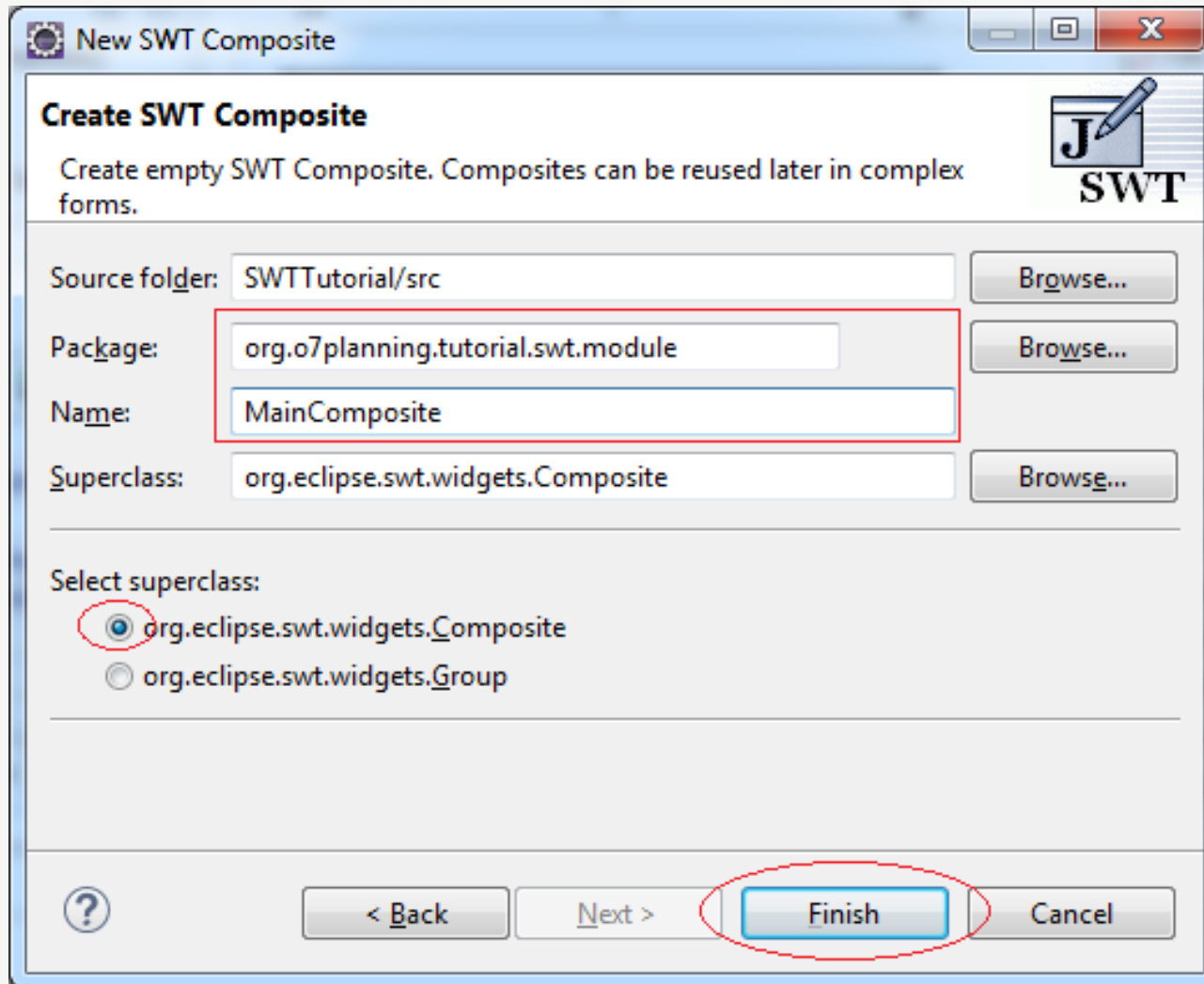
```

12 */
13 public BottomComposite(Composite parent, int style) {
14     super(parent, style);
15     setLayout(new FillLayout(SWT.HORIZONTAL));
16
17     Composite composite = new Composite(this, SWT.NONE);
18     composite.setLayout(new GridLayout(1, false));
19
20     Composite composite_1 = new Composite(composite, SWT.NONE);
21     GridLayout gl_composite_1 = new GridLayout(3, false);
22     gl_composite_1.marginHeight = 0;
23     gl_composite_1.marginWidth = 0;
24     composite_1.setLayout(gl_composite_1);
25     composite_1.setLayoutData(new GridData(SWT.FILL, SWT.CENT, true, true));
26
27     Button btnNewButton = new Button(composite_1, SWT.NONE);
28     btnNewButton.setLayoutData(new GridData(SWT.FILL, SWT.CENT, true, true));
29     btnNewButton.setText("Add");
30
31     Button btnNewButton_1 = new Button(composite_1, SWT.NONE);
32     btnNewButton_1.setLayoutData(new GridData(SWT.FILL, SWT.CENT, true, true));
33     btnNewButton_1.setText("Delete");
34
35     Button btnNewButton_2 = new Button(composite_1, SWT.NONE);
36     btnNewButton_2.setLayoutData(new GridData(SWT.FILL, SWT.CENT, true, true));
37     btnNewButton_2.setText("Clear");
38
39     text = new Text(composite, SWT.BORDER | SWT.MULTI);
40     text.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
41
42 }
43
44 @Override
45 protected void checkSubclass() {
46     // Disable the check that prevents subclassing of SWT controls
47 }
48
49 }

```

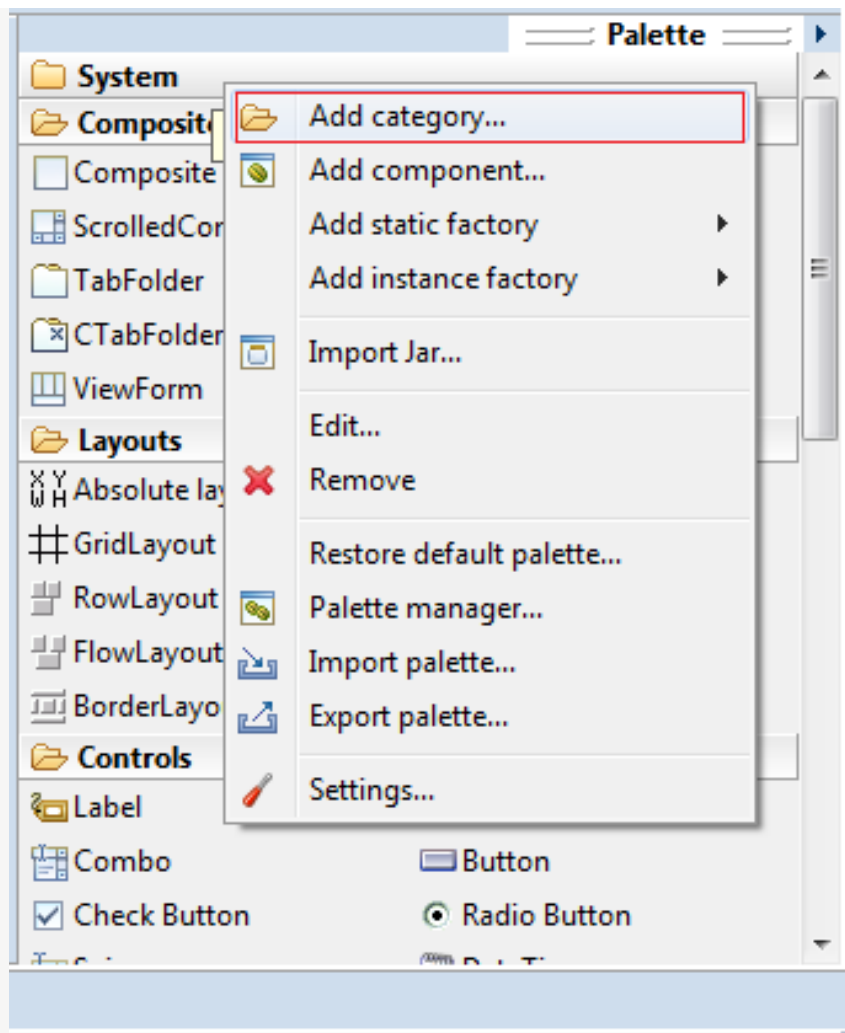
MainComposite

Similarly create class **MainComposite**:




Register TopComposite & BottomComposite into Palette

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



Named Palette Category:

- My Composite

 New palette category

Add a new category to the palette
Specify the name and description of the category.

ID:

custom_1414864672083

Name:

My Composite

Description:

State

☒ Visible

☒ Open by default

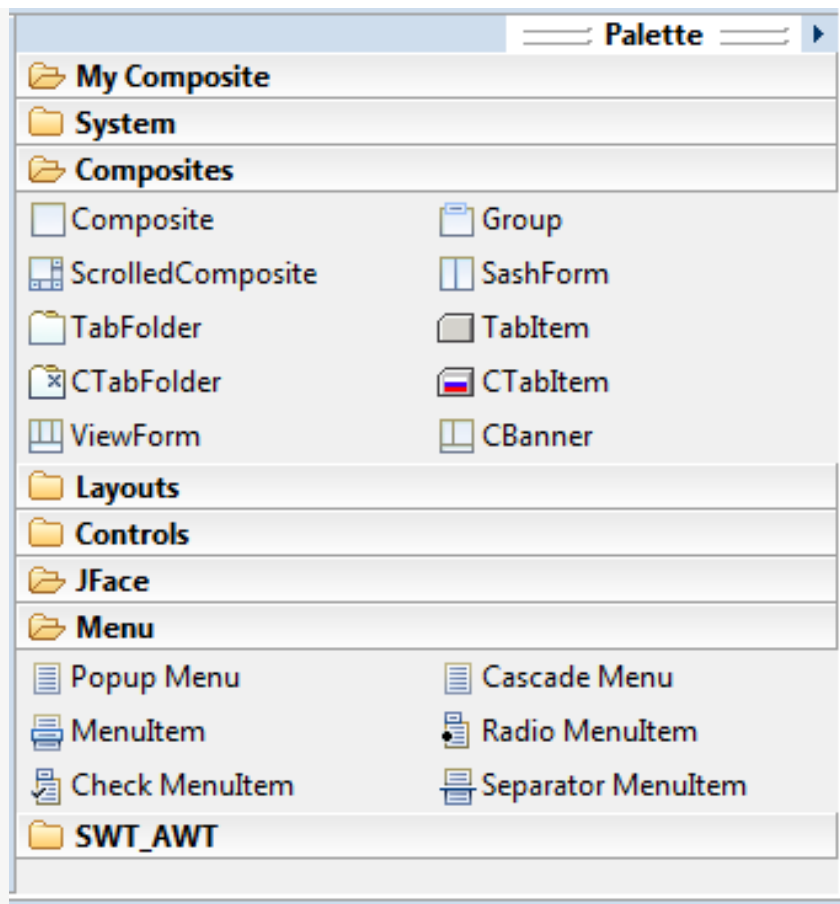
Insert new category:

before "System"

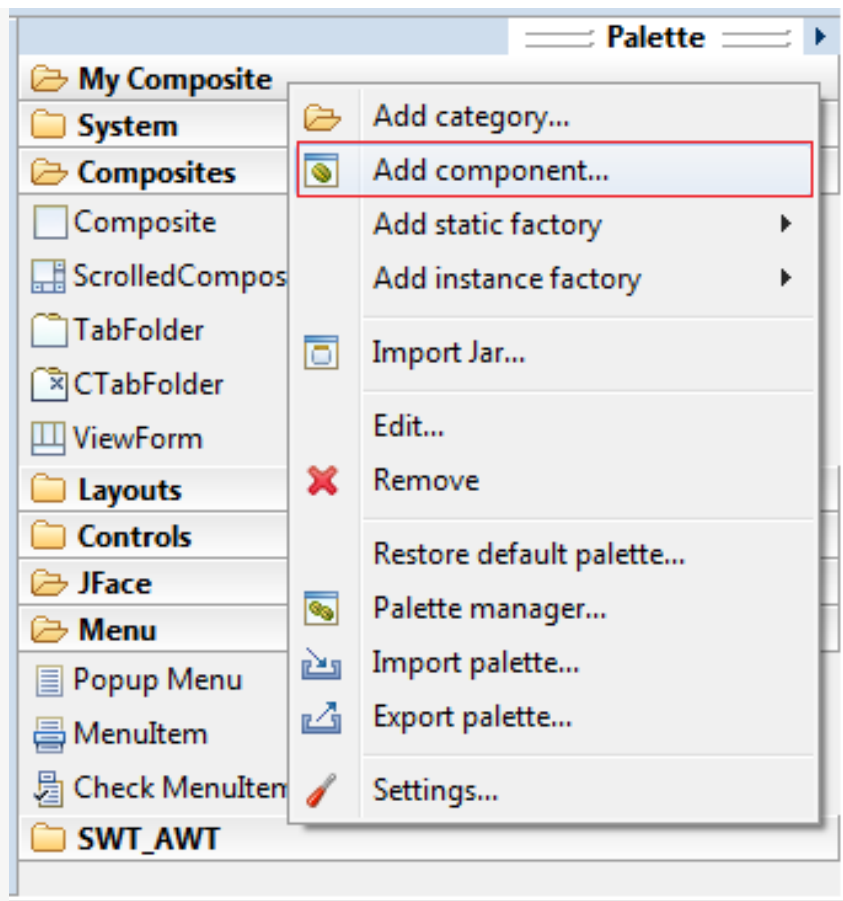
?


OK

Cancel




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



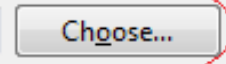
 Add component

Add component to the palette.

 Component class name can not be empty.

ID:


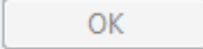
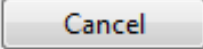
Name:

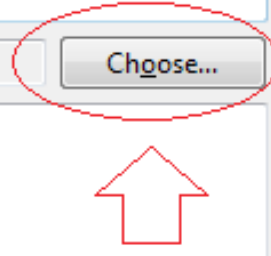
Class name: 

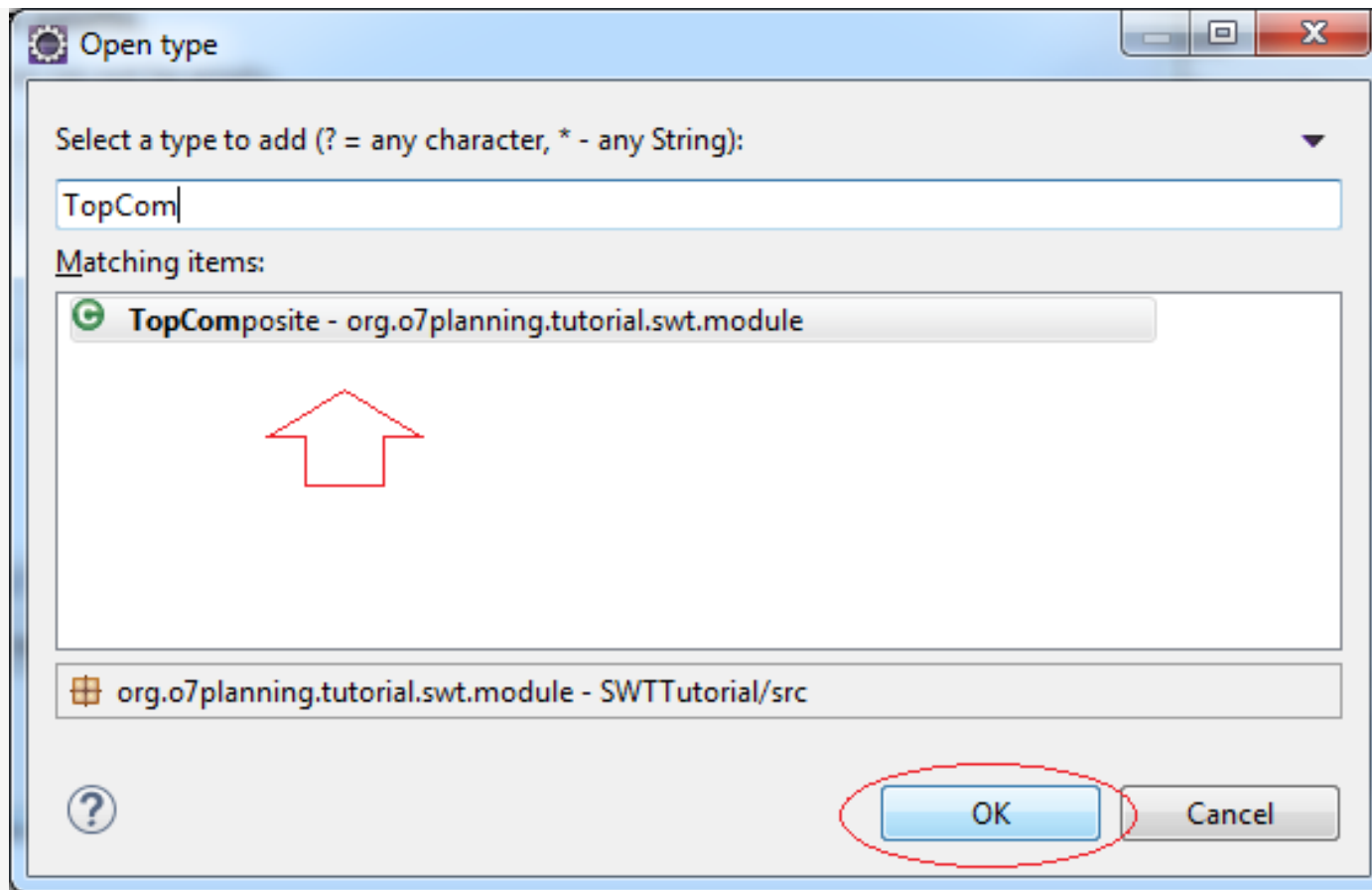
Description:


Visible: ☒

Add to palette category:





 Add component ✕

Add component to the palette.
Specify the name, class and description of the component.

ID:


Name:

Class name: Choose...

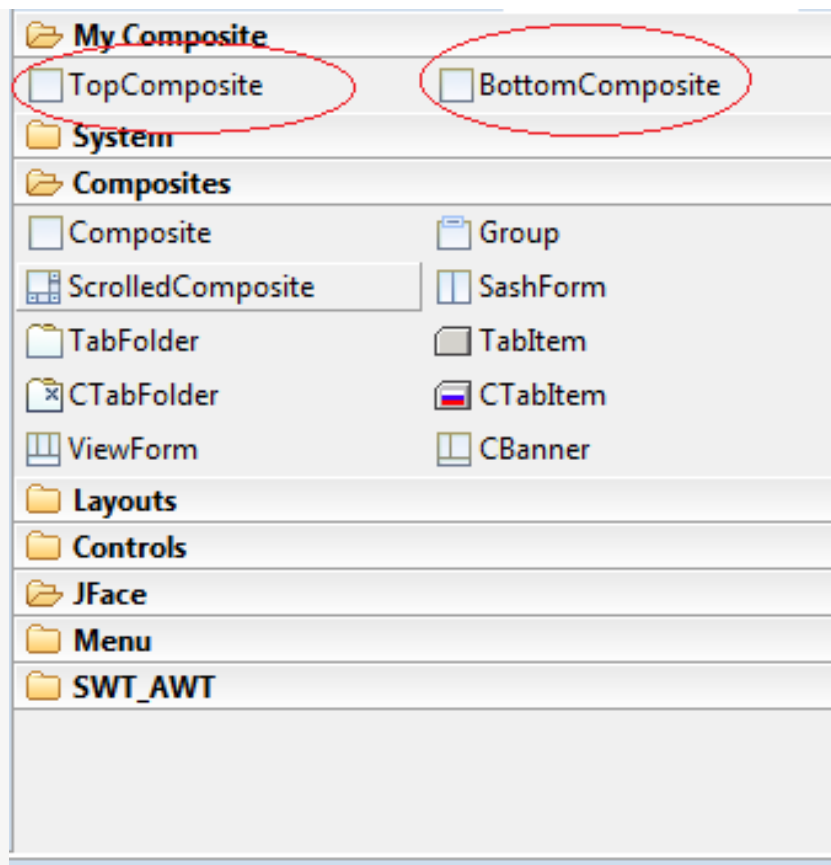
Description:

Visible: ☒

Add to palette category:

 OK Cancel

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



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

- **MainComposite.java**

```
1 package org.o7planning.tutorial.swt.module;
2
3 import org.eclipse.swt.SWT;
4 import org.eclipse.swt.layout.FillLayout;
5 import org.eclipse.swt.layout.GridData;
6 import org.eclipse.swt.layout.GridLayout;
7 import org.eclipse.swt.widgets.Composite;
8
```

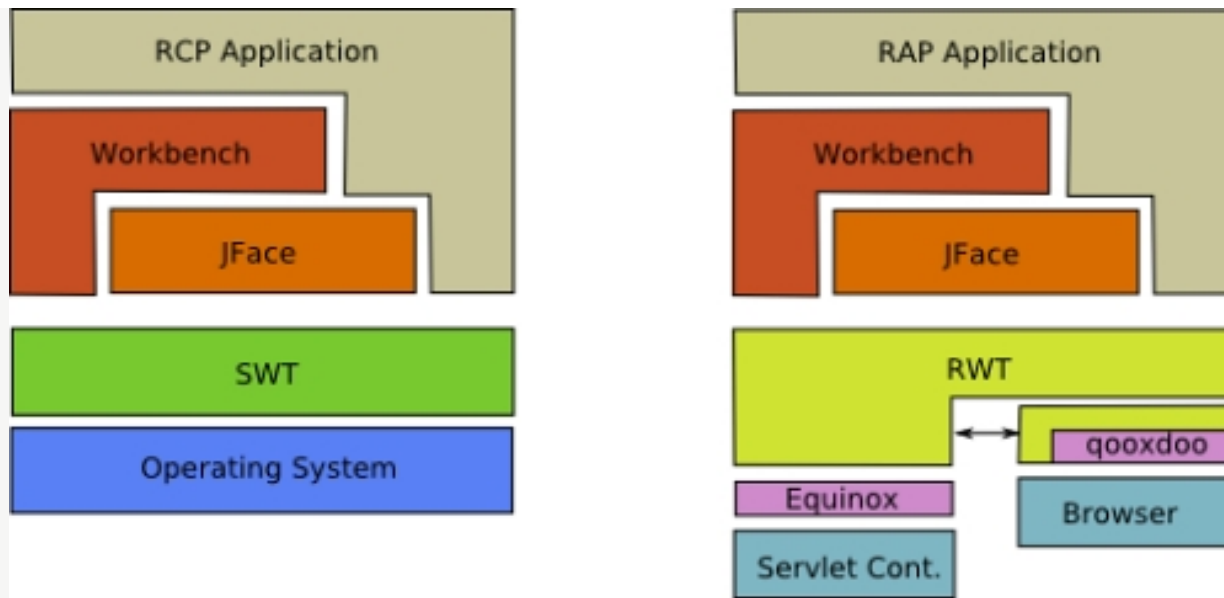


```

9  public class MainComposite extends Composite {
10
11      /**
12       * Create the composite.
13       * @param parent
14       * @param style
15       */
16      public MainComposite(Composite parent, int style) {
17          super(parent, style);
18          setLayout(new FillLayout(SWT.HORIZONTAL));
19
20          Composite composite = new Composite(this, SWT.NONE);
21          composite.setLayout(new GridLayout(1, false));
22
23          TopComposite topComposite = new TopComposite(composite, S
24          topComposite.setLayoutData(new GridData(SWT.FILL, SWT.CEN
25
26          BottomComposite bottomComposite = new BottomComposite(con
27          bottomComposite.setLayoutData(new GridData(SWT.FILL, SWT.
28
29      }
30
31      @Override
32      protected void checkSubclass() {
33      }
34
35  }

```

12- JFace



You can see more JFace, an additional API for SWT:

- <http://o7planning.org/web/fe/default/en/document/26330/eclipse-jface-tutorial>



o7planning.org

