Eclipse

Eclipse Window Builder Tutorial for GUI Creation

Mohanraj Gurubatham • December 24th, 2015 Last Updated: March 27th, 2019 1

1.Introduction

In this example, we will show you how to develop Java GUI Application using Eclipse WindowBuilder plug-in.

Eclipse WindowBuilder is a powerful and easy to use bi-directional Java GUI designer that makes it very easy to create Java GUI applications without spending a lot of time writing code to display simple forms.

The bi-directional Java GUI designer means the developer can seamlessly move between a Drag n' Drop designer and the generated code.

Using Eclipse WindowBuilder, the developer will enjoy creating Java GUI based applications. One can create complicated windows in minutes using WindowBuilder.

WYSIWYG (What You See Is What You Get) layout tools in WindowBuilder are used to generate back-end java code by drag-and-drop of components to the container.

2.Simple Java Window Application

Now, we will see how fast a simple Java GUI application can be created uisng Eclipse WindowsBuilder.

2.1 System requirements

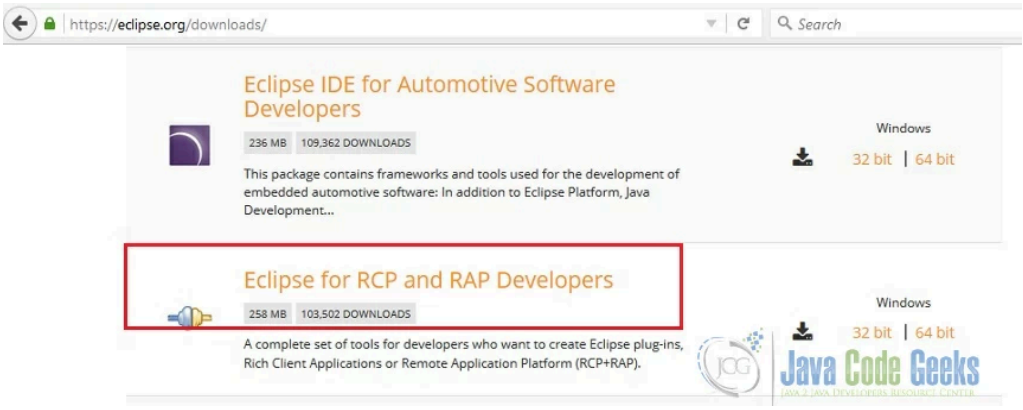
Tools required to run this example are:

2.1.1 Eclipse

WindowBuilder is built as a plug-in to Eclipse. 'Eclipse for RCP and RAP Developers' is the default IDE bundled with 'Windows Builder' plug-in. This IDE

has a complete set of tools for developers who want to create Eclipse plug-ins, Rich Client Applications (RCA).

Download 'Eclipse for RCP and RAP Developers' from here. Please refer the picture given below to identify the correct IDE.



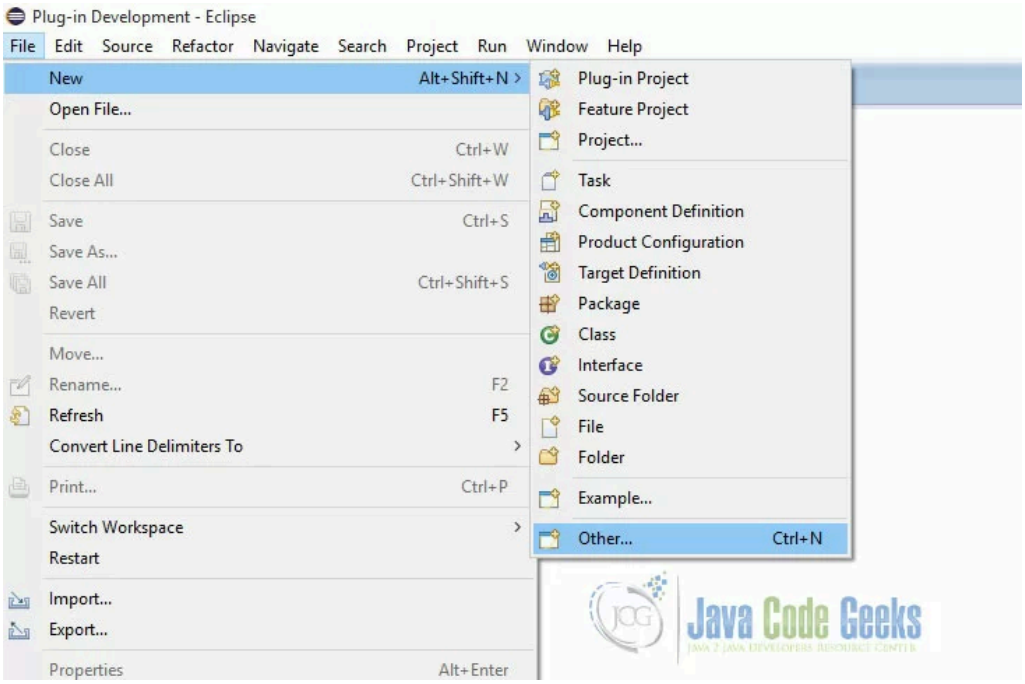
Eclipse IDE for RCP and RAD

2.1.2 Java

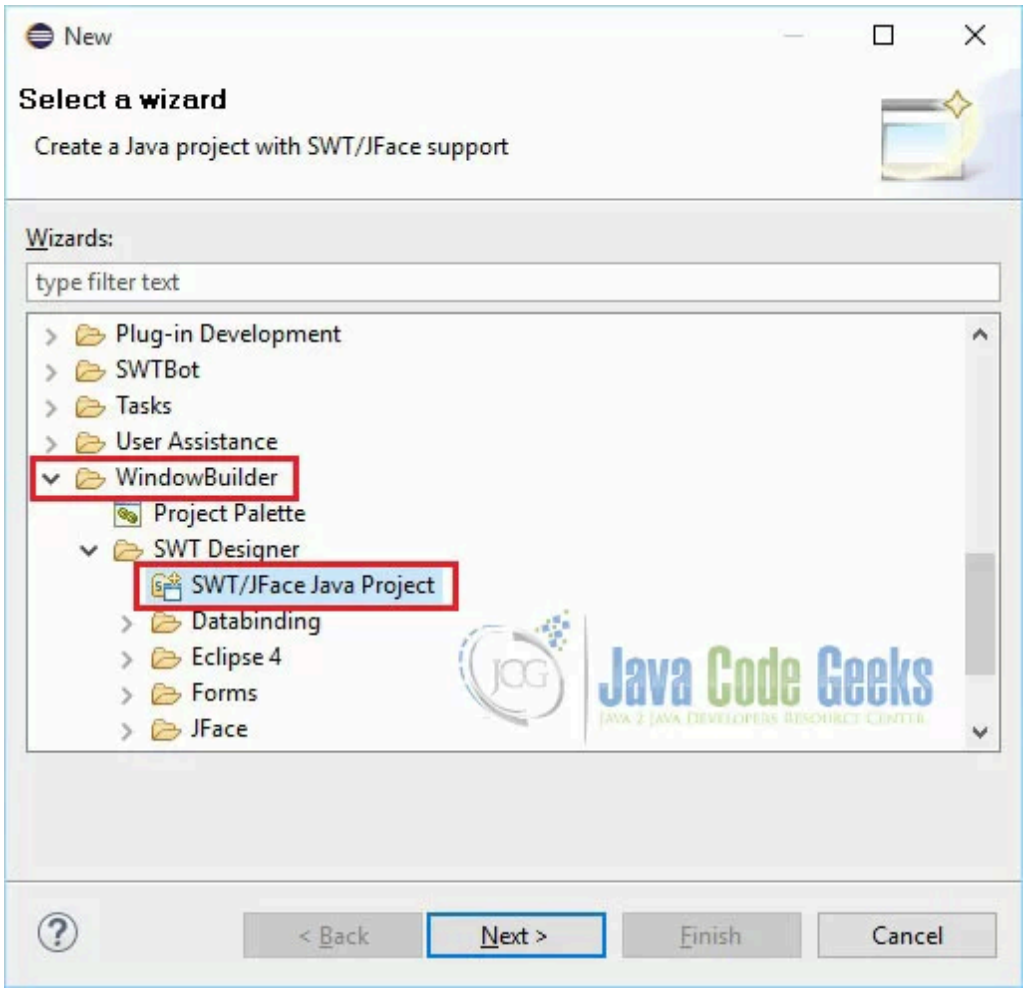
- Download Java SE 7 or above from here

3. Open New Project

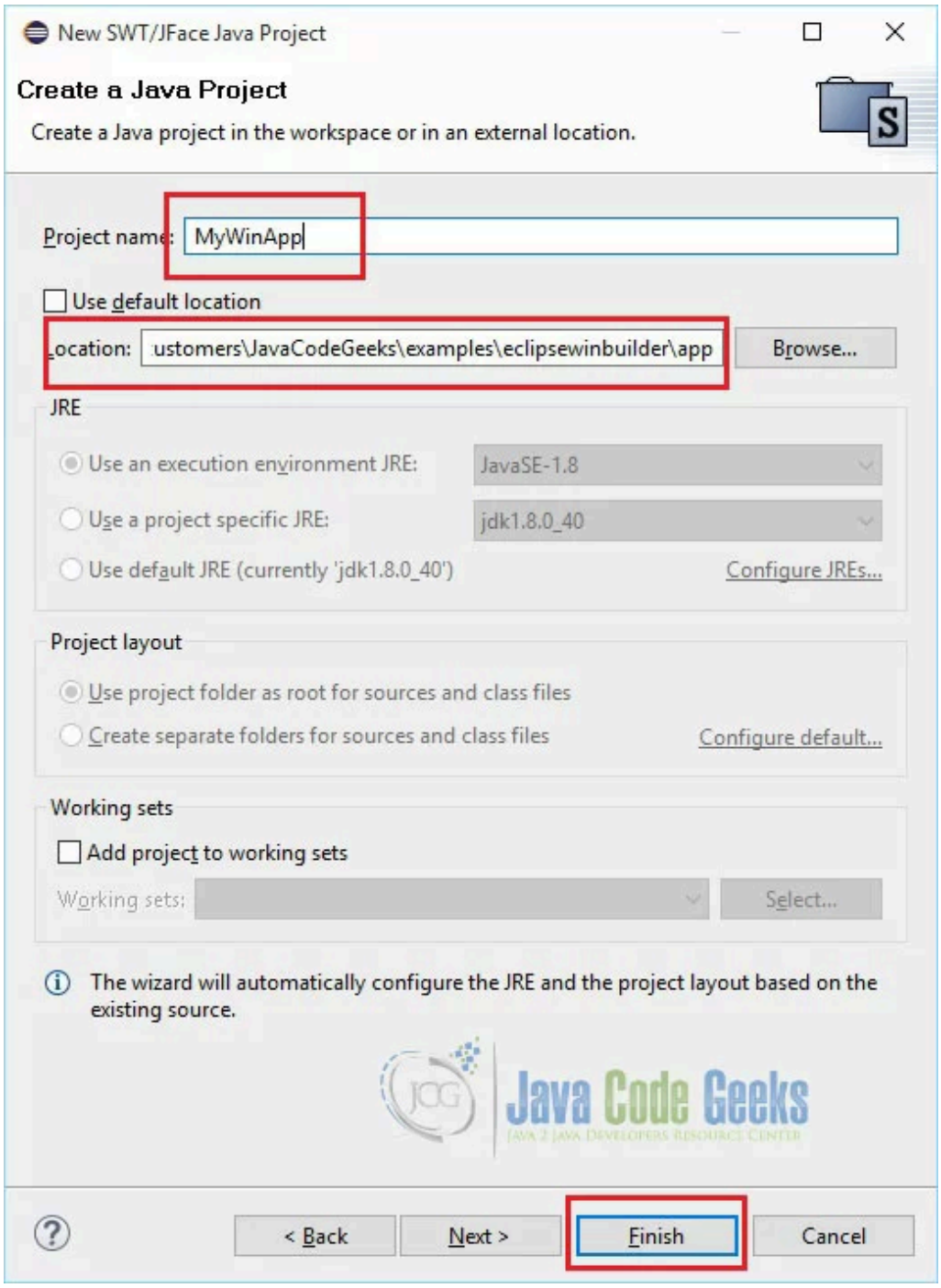
Let us create a new 'SWT/JFace Java Project' to see the usage of WindowBuilder for building GUI components. Open 'File – New – Other' and then click 'SWT/JFace Project' as depicted below



Open Project



SWT/JFace Java Project



Project Name

The reason for creating new project as 'SWT/JFace Java Project' is to have all the necessary JARs and native libraries included by the IDE itself. Otherwise, you have to add all these dependent JARs and native libraries on your own.

The Standard Widget Toolkit (SWT) is a graphical widget toolkit to be used with the Java platform. It provides a portable graphics API independent of the OS but that relies on the native widgets.

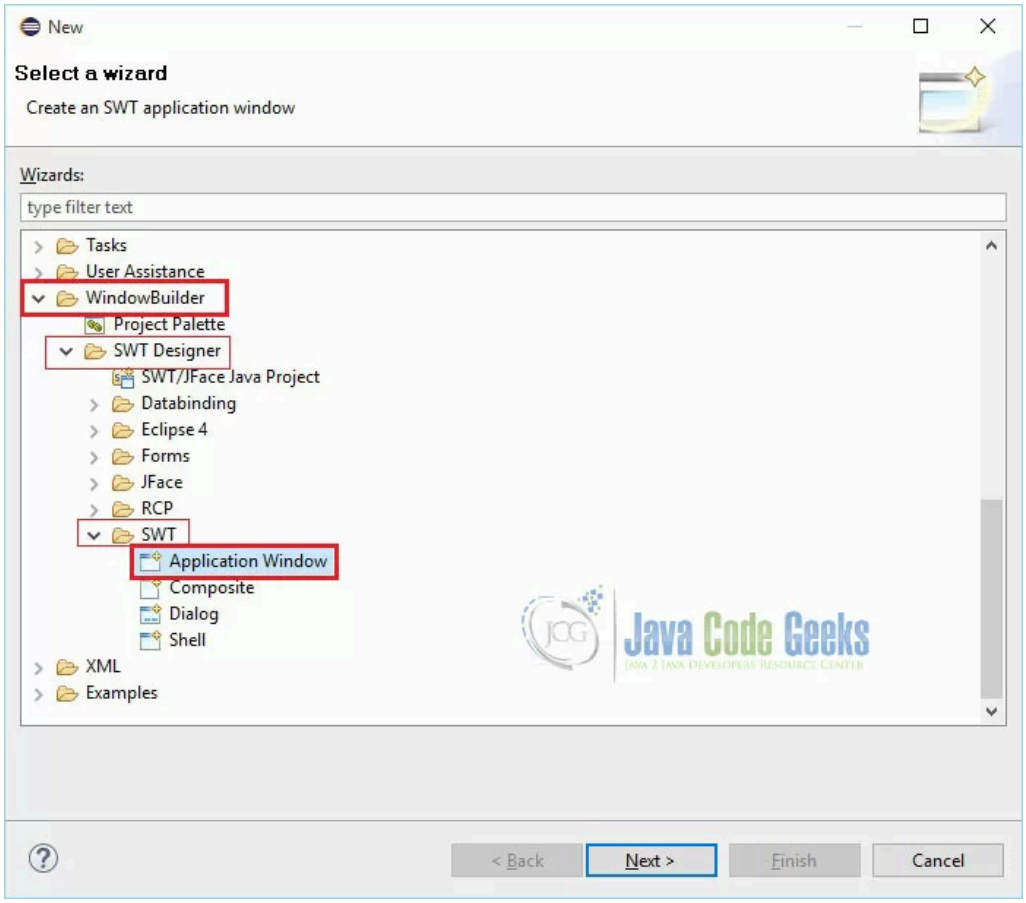
JFace is a UI toolkit with classes for handling many common UI programming tasks. JFace is window-system-independent in both its API and

implementation, and is designed to work with SWT without hiding it.

JFace is a higher-level user interface toolkit that uses the raw SWT widgets to provide model-driven widgets, and to some extent some functionality that isn't available in the Swing libraries, such as advanced editors, dialog boxes, and wizards.

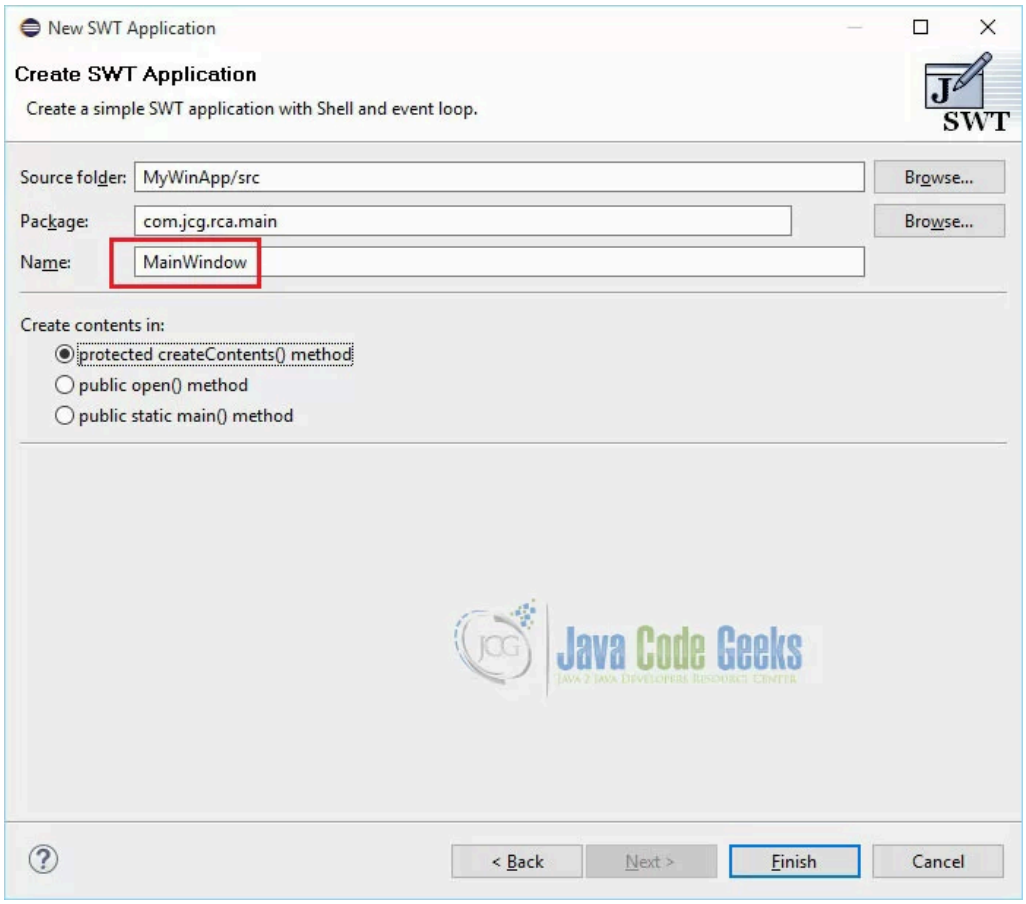
4. New SWT Application

Let us add widget to the project. As a main window, create Application Window as shown below. Right click on the project and select 'New – Other – Window Builder – SWT Designer – SWT – Application Window'. And then click 'Next'



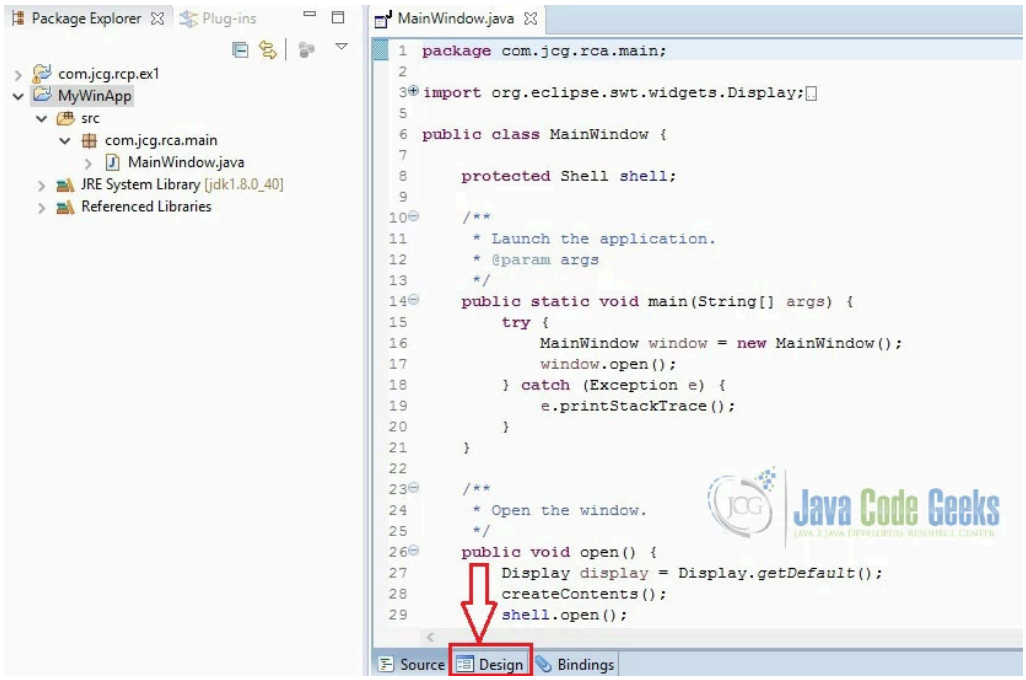
Application Window

Enter Class Name and click 'Finish'



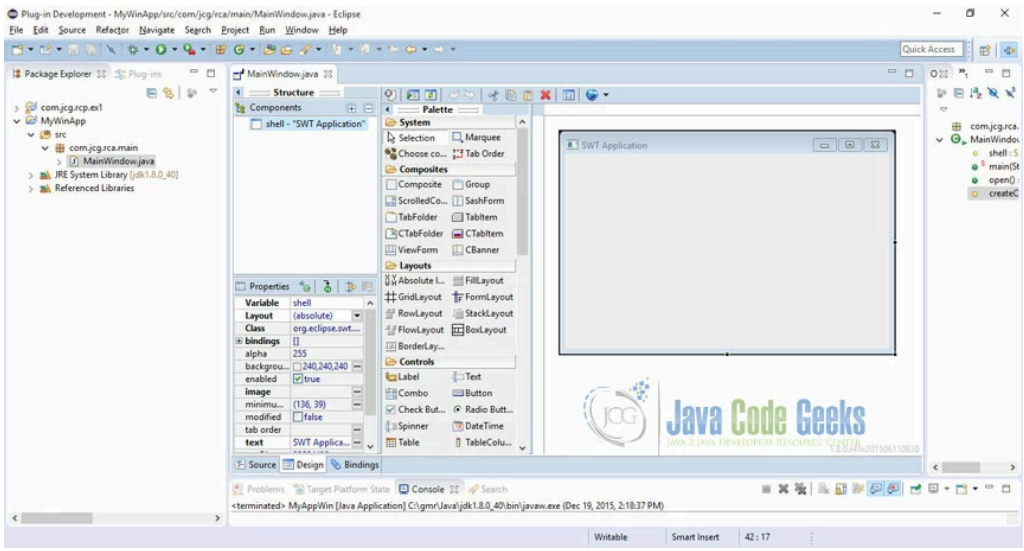
Class Name

A basic window application has been created. Window Builder can be used to get your UI up and running quickly. Click 'Design' tab as shown below.



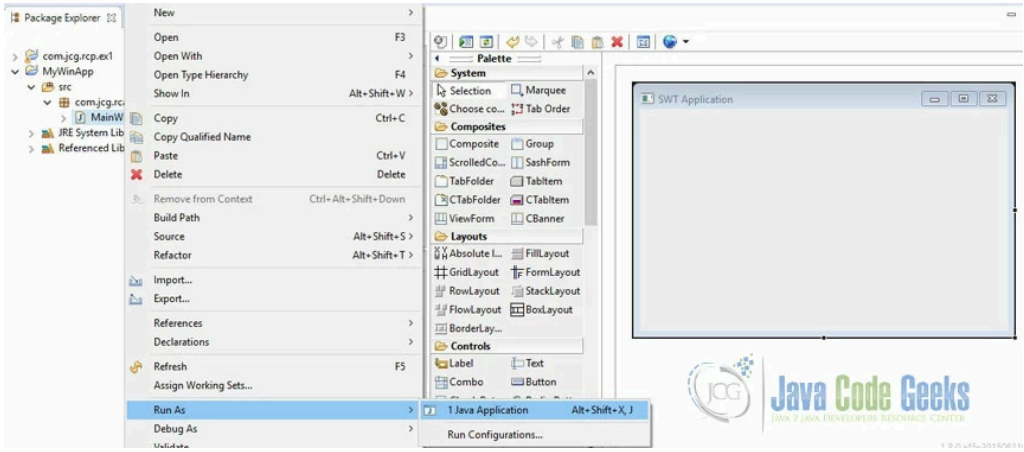
Basic Window Application

Now, you will see the graphical representation (Design View) of your code.



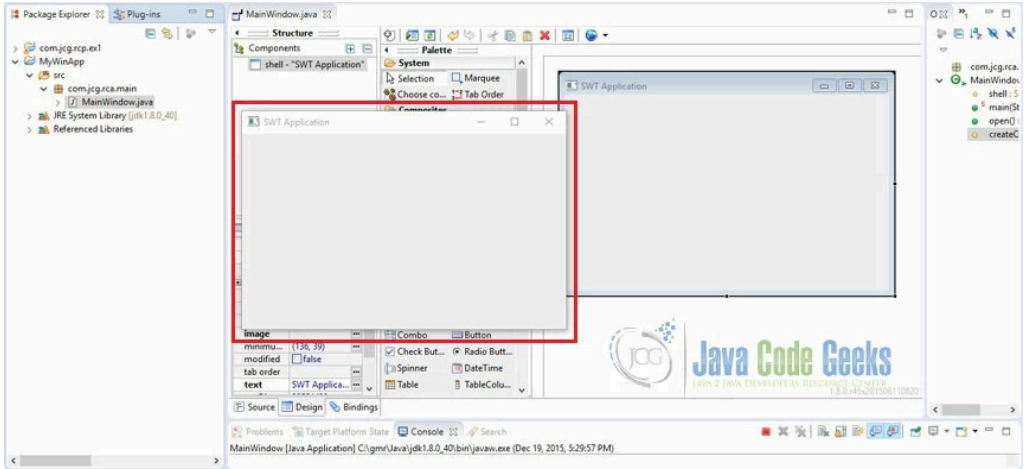
Design View

This application can be simply executed like any other java program with main method. Right click on the class name and 'Run As – Java Application'



Run Application

As we have not yet added any other elements, you will see a simple window popping-up as a result of the execution.



5. Components in the editor

As shown above, the editor is composed of the following major components:

- **Design View** – the main visual layout area.
- **Source View** – write code and review the generated code
- **Structure View** – composed of the **Component Tree** and the **Property Pane.**
    - **Component Tree** – shows the hierarchical relationship between all of the components.
    - **Property Pane** – displays properties and events of the selected components.
- **Palette** – provides quick access to toolkit-specific components.
- **Toolbar** – provides access to commonly used commands.
- **Context Menu** – provides access to commonly used commands.

6. Editor Features

The editor supports the following major features;

- **Bi-directional Code Generation –** read and write almost any format and reverse-engineer most hand-written code
- **Internationalization (i18n) / Localization –** externalize component strings, create and manage resource bundles.
- **Custom Composites & Panels** – create custom, reusable components.
- **Factories** – create custom factory classes and methods.
- **Visual Inheritance** – create visual component hierarchies.
- **Event Handling** – add event handlers to your components.
- **Menu Editing** – visually create and edit menubars, menu items and popup menus.
- **Morphing** – convert one component type into another.

7. Layouts in SWT

Layouts are non-visible widgets used to give GUI windows a specific look and it helps to control the position and size of children in a *Composite*.

To make sure the GUI application developed in one environment works perfect in another platform, Java provides a system of portable layout managers. We use these layout managers to specify rules and constraints for the layout of the UI in a way that will be portable.

Layout managers gives you the advantages as given below,

- Correctly positioned components that are independent of fonts, screen resolutions, and platform differences.
- Intelligent component placement for containers that are dynamically resized at runtime.
- Ease of translation. If a string increases in length after translation, the associated components stay properly aligned.

SWT Designer supports the following layout managers.

Layout Manager	Description
AbsoluteLayout	AbsoluteLayout or Null Layout helps to specify the exact position, the width and the height of components. In a generic environment where the size of the screens may vary, this layout manager should be avoided.
FillLayout	FillLayout is the simplest layout class. It lays out controls in a single row or column, forcing them to be the same size.
RowLayout	Puts the widgets in rows or columns and allows you to control the layout with options, e.g., wrap, spacing, fill and so on.
GridLayout	Arranges widgets in a grid.
FormLayout	Arranges the widgets with the help of the associated attachments.
StackLayout	A StackLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.
BorderLayout	BorderLayout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.
BoxLayout	BoxLayout allows multiple components to be laid out either vertically or horizontally. The components will not wrap so, for example, a vertical arrangement of components will stay vertically arranged when the frame is resized. Nesting multiple panels with different combinations of horizontal and vertical gives an effect similar to GridBagLayout, without the complexity.

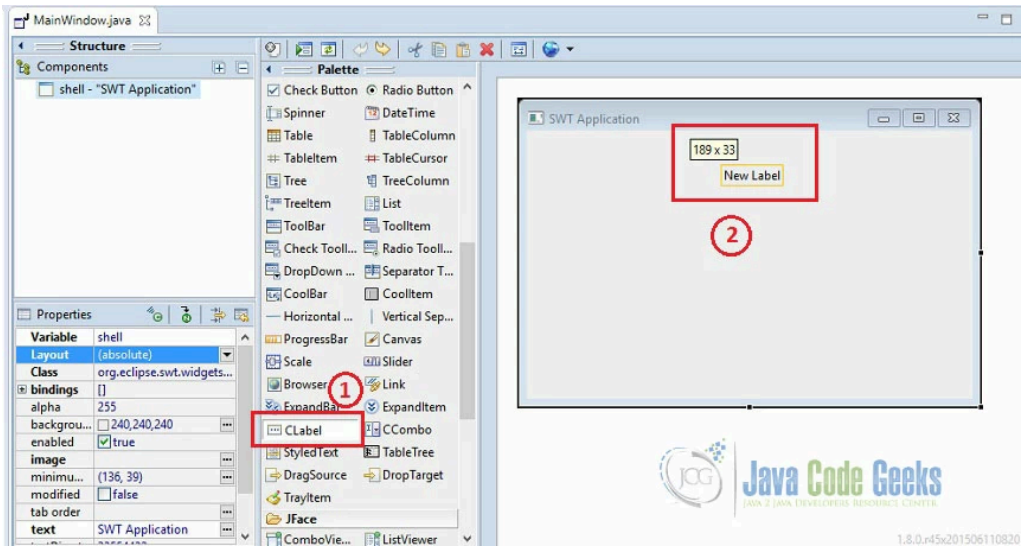
Layout Manager	Description
FlowLayout	A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line.

8. New UI Page

We will now design a new Login UI page using Window Builder. For this normal size screen, we will continue with the default (absolute) layout. We are going to have an image, two labels, one text field, one password field and a button on the screen.

To display image use CLabel widget. CLabel supports aligned text and/or an image and different border styles.

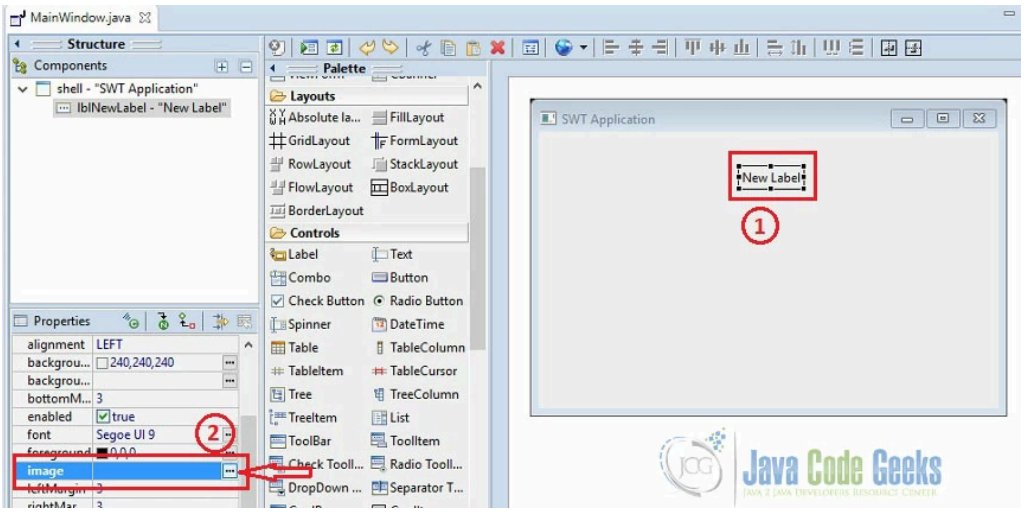
As shown below, click 'CLabel' once and keep your cursor on the screen and click. Now, the 'CLabel' is placed on the screen.



New Login UI

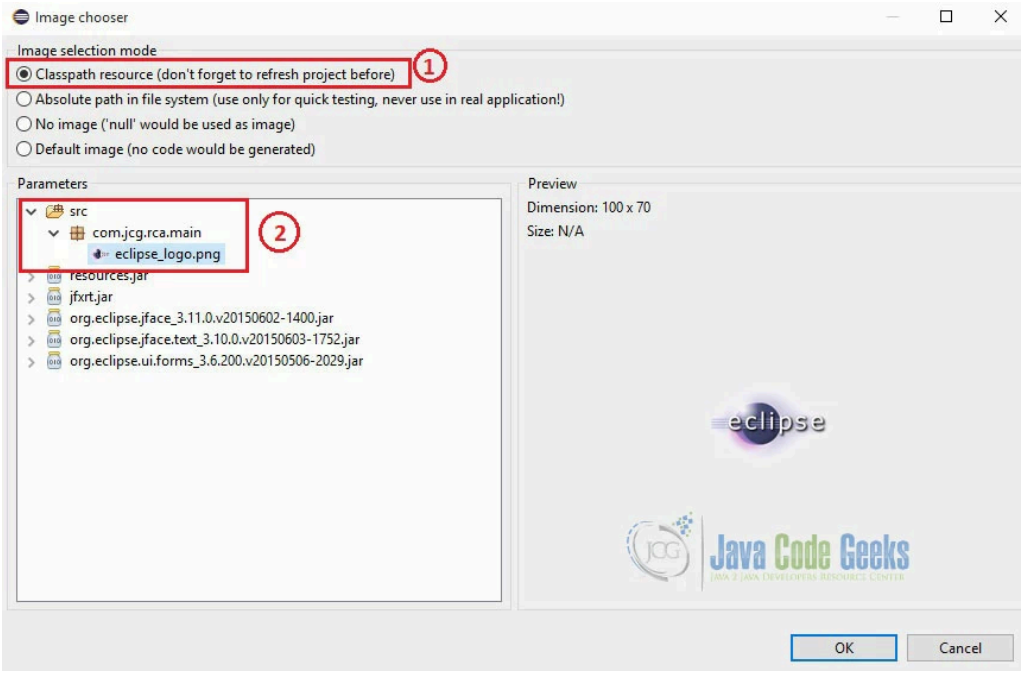
Let us attach an image with 'CLabel'. For this, you need to have an image in the folder where your 'MainWindow' source file is placed. For this example, I have used eclipse logo.

Click on the 'CLabel' and then, in the 'Properties' window select 'image'.



CLabel Image

You will now see the Image chooser window pops up. Select 'Classpath resource' option and navigate to the image file, select it and then click 'OK'.



Select Image

Adjust the field bounds according to the size of the logo so that the image is visible on the screen.

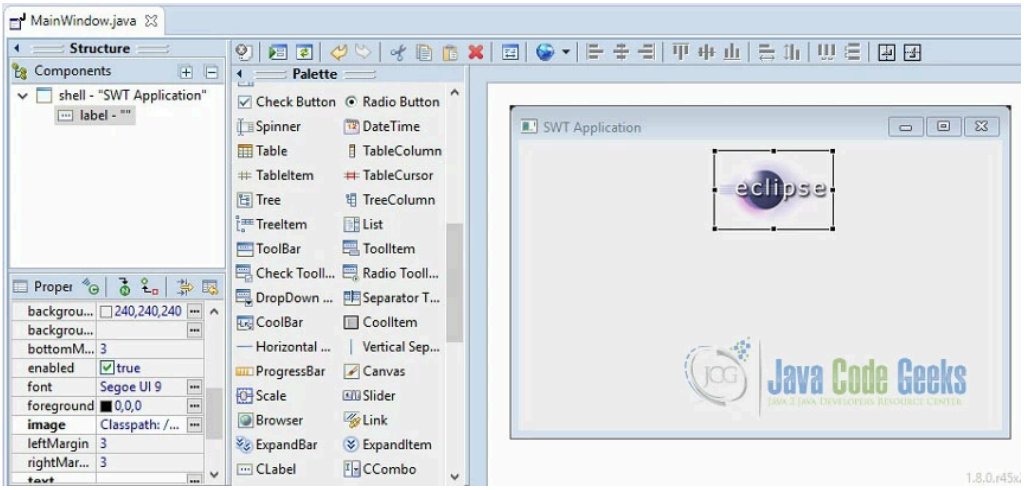
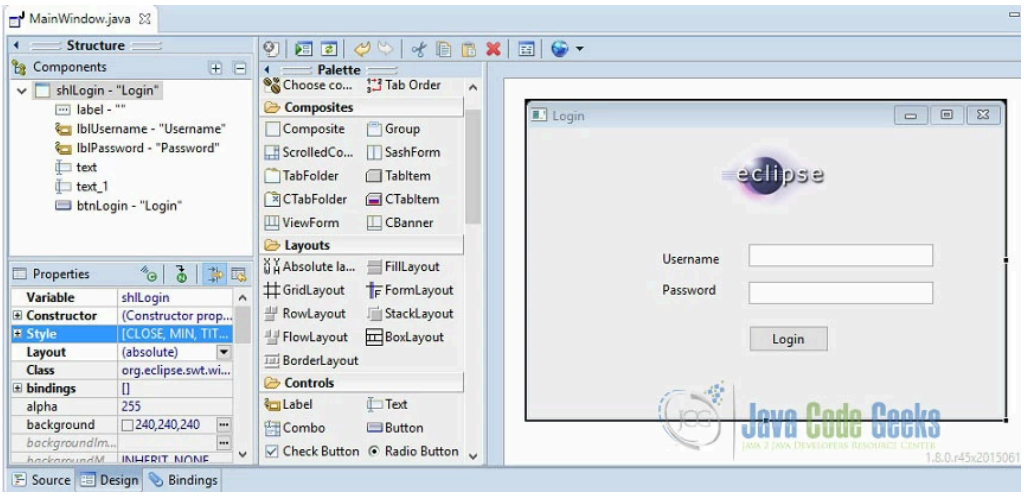


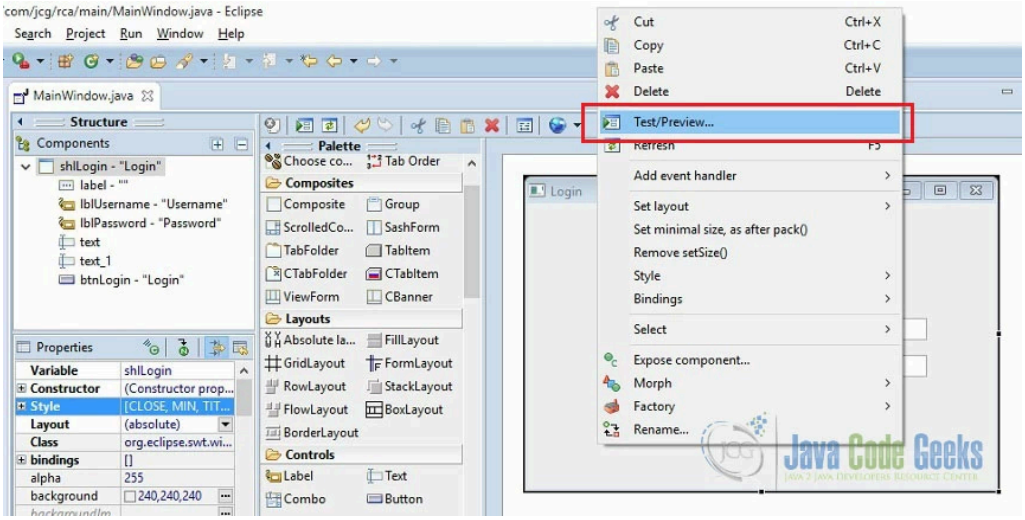
Image Attached

Similarly, add Labels, Text Fields and a Button. Finally the screen will be looking like the one shown below.

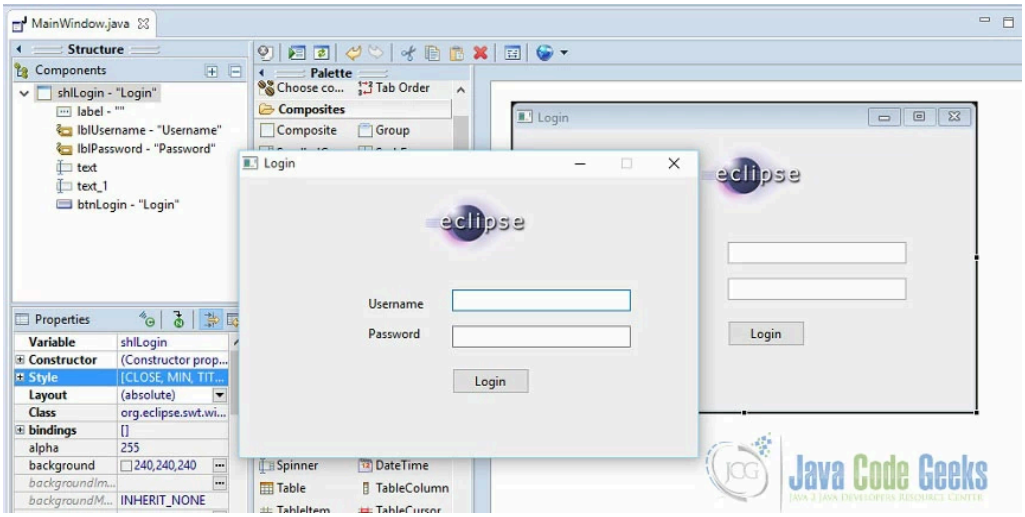


Login UI Page

To test this design, right click on the window and select 'Test/Preview' from the popup menu.



Test GUI



Test / Preview

9. Source View

Click 'Source' tab to see the code generated by the IDE. Single line of code in this was not written manually.

```
protected void createContents() {
    shlLogin = new Shell(SWT.CLOSE | SWT.TITLE | SWT.MIN);
    shlLogin.setSize(450, 300);
    shlLogin.setText("Login");

    CLabel label = new CLabel(shlLogin, SWT.NONE);
    label.setImage(SWTResourceManager.getImage(MainWindow.class, "/com/jcg/rca/main/eclipse_logo.png"));
    label.setBounds(176, 10, 106, 70);
    label.setText("");

    Label lblUsername = new Label(shlLogin, SWT.NONE);
    lblUsername.setBounds(125, 115, 55, 15);
    lblUsername.setText("Username");

    Label lblPassword = new Label(shlLogin, SWT.NONE);
    lblPassword.setBounds(125, 144, 55, 15);
    lblPassword.setText("Password");

    text = new Text(shlLogin, SWT.BORDER);
    text.setBounds(206, 109, 173, 21);

    text_1 = new Text(shlLogin, SWT.BORDER);
    text_1.setBounds(206, 144, 173, 21);

    Button btnLogin = new Button(shlLogin, SWT.NONE);
    btnLogin.setBounds(206, 185, 75, 25);
    btnLogin.setText("Login");
```


Source View

10. Button Listener

Attach listener with the button to validate field entries. Refer the source code of the main file given below.

MainWindow.java

```java
package com.jcg.rca.main;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.CLabel;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.eclipse.wb.swt.SWTResourceManager;

public class MainWindow {

    protected Shell shlLogin;
    private Text userNameTxt;
    private Text passwordTxt;

    private String userName = null;
    private String password = null;

    /**
     * Launch the application.
     *
     * @param args
     */
    public static void main(String[] args) {
        try {
            MainWindow window = new MainWindow();
            window.open();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Open the window.
     */
    public void open() {
        Display display = Display.getDefault();
        createContents();
        shlLogin.open();
        shlLogin.layout();
        while (!shlLogin.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
    }

    /**
     * Create contents of the window.
     */
    protected void createContents() {
        shlLogin = new Shell(SWT.CLOSE | SWT.TITLE | SWT
        shlLogin.setSize(450, 300);
        shlLogin.setText("Login");

        CLabel label = new CLabel(shlLogin, SWT.NONE);
        label.setImage(SWTResourceManager.getImage(MainW
```

```
063             label.setBounds(176, 10, 106, 70);
064             label.setText("");
065
066             Label lblUsername = new Label(shlLogin, SWT.NONE
067             lblUsername.setBounds(125, 115, 55, 15);
068             lblUsername.setText("Username");
069
070             Label lblPassword = new Label(shlLogin, SWT.NONE
071             lblPassword.setBounds(125, 144, 55, 15);
072             lblPassword.setText("Password");
073
074             userNameTxt = new Text(shlLogin, SWT.BORDER);
075             userNameTxt.setBounds(206, 109, 173, 21);
076
077             passwordTxt = new Text(shlLogin, SWT.BORDER | SW
078             passwordTxt.setBounds(206, 144, 173, 21);
079
080             Button btnLogin = new Button(shlLogin, SWT.NONE)
081             btnLogin.setBounds(206, 185, 75, 25);
082             btnLogin.setText("Login");
083
084             btnLogin.addListener(SWT.Selection, new Listener
085                 public void handleEvent(Event event) {
086
087                     userName = userNameTxt.getText();
088                     password = passwordTxt.getText();
089
090                     if (userName == null || userName.isEmpty
091                         String errorMsg = null;
092                         MessageBox messageBox = new MessageB
093
094                         messageBox.setText("Alert");
095                         if (userName == null || userName.isE
096                             errorMsg = "Please enter usernam
097                         } else if (password == null || passw
098                             errorMsg = "Please enter passwor
099                         }
100                         if (errorMsg != null) {
101                             messageBox.setMessage(errorMsg);
102                             messageBox.open();
103                         }
104                     } else {
105                         MessageBox messageBox = new MessageB
106                         messageBox.setText("Info");
107                         messageBox.setMessage("Valid");
108                         messageBox.open();
109                     }
110                 }
111             });
112
113         }
114
115 }
```

11. Conclusion

From this example, we learned how quickly a Java GUI application can be developed using Eclipse Window Builder. WindowBuilder Engine provides a rich

API for creating UI designers. It supports Java-based UI frameworks such as Swing, SWT/RCP, eRCP, GWT etc. It also supports XML-based UI frameworks like XWT, GWT UiBinder, Android etc.

12. Download the Code Project

This was a Tutorial about Eclipse Window Builder for GUI Creation.

Download

You can download the full source code of this example here: **WindowBuilderExample**

Tags desktop java Window Builder

1 COMMENT Oldest

Nate 6 years ago

One big thing: I installed Eclipse Oxygen and then the WindowBuilder and SWT plugins. Following the tutorial I find that the shell – SWT Application in the design window is upside down as is the text. It previews OK though.