System Overview Document

for

Selectors: *Going Beyond User-Interface Widgets*
(Updated)

Sponsor
Jeff Johnson, University of San Francisco

Submitted by

Stewart Powell
Hanglin (Will) Li
Serena Pang
Katrina Monje

in partial fulfillment
of the requirements for

CS 490/BUS 398, *Software Team Projects*
Prof. Doug Halperin and Prof. Luigi Lucaccini

at the University of San Francisco

9 December 2020

Table of Contents

Project Overview

UI Selectors is a front-end web development construction toolkit that allows programmers to build applications using a library of semantic based controls (i.e. sliders, menus, radio buttons, etc.), where the appropriate functionality of those controls is prioritized rather than the overall appearance and layout. These controls are called ***UI Selectors***. This project follows the Model-View-Controller (MVC) architectural model. A total of four semantic controls were made: (1) ***On/Off Switch***, (2) ***M-from-N or Multiple Choice***, (3) ***One-from-N Choice***, and (4) ***Value-in-Range Choice***. All of which have several presenters to choose from. Through UI Selectors, building software applications will become much more accessible, and users do not have to rely on traditional methods of the current UI development toolkits that exist today. Our team has addressed the following issues regarding usability of widgets instead of UI Selectors:

1. Most widgets are very low-level, which makes UI construction laborious.
2. When working with widgets, often appearance and layout are prioritized too much.
3. Designers are almost always the decision makers in developing UIs, and they easily make poor widget choices.
4. Widgets cannot properly work with application semantics—they cannot identify the data-type of the variables which is more prone to errors on the program's behavior.
5. If different presentations are available for the same data set, there is no laid foundation that will keep them all concurrent (i.e. extra code is necessary).

List of Key Features

Parent Classes

As previously mentioned, the four semantic controls we built have their own corresponding presenters. The parent classes are: Semantic Control, UI Runner, UI View, and Views. The Semantic Control class sets the name and size of the display and the shell, and its function is to add the views and values for building all the semantic controls. The UI Runner class launches the user interface, while the UI View class associates each of the semantic control models with its corresponding views. There are "add" and "remove" buttons for the views which have the ability to update the semantic control and its view. Finally, the Views class is an enum class for all the presenter types.

Semantic Control Model and Presenters

The following table lists all the presenters our team has created for each of the four semantic controls:

Semantic Control	Attributes	Presenters	Description
On/Off Switch	Boolean	Switch Button View, Checkbox View, Toggle Button View	Push-ON, push-OFF button; labeled checkboxes; standard toggle light switch
M-from-N Choice	Subset of Set	Double List View, Full List View, Scroll List View	Reference list and separate "selected items" list; fully displayed list of

			labeled checkboxes; scrollable list of labeled checkboxes
One-from-N Choice	Enumerated	Pop-up Menu View, Box Button View, Radio Button View	Button that pops up a choose-one menu; array of rectangular labeled buttons; array of bullets with labels next to each one
Value-in-Range	Float	Spinner View, Slider View	Dial; Knob that slides from one end to another

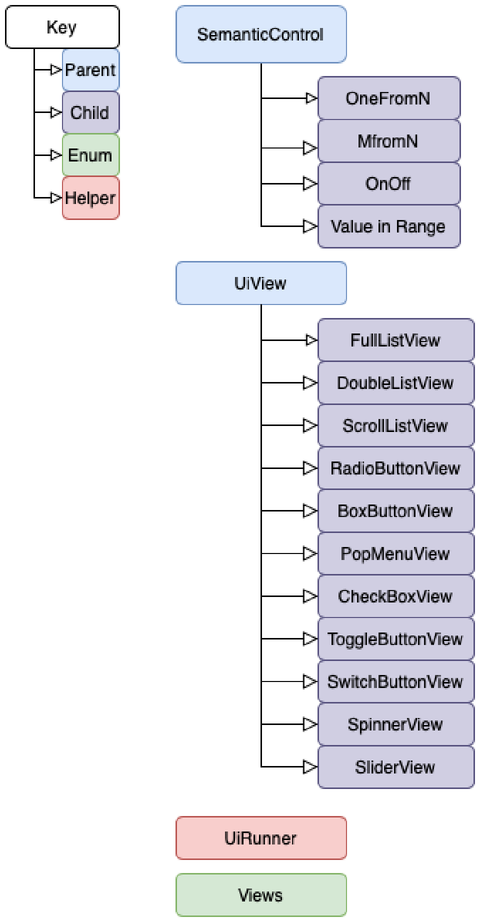
Next Steps

- What remains to be done:
  - More tests to ensure stability.
    More different tests are necessary. With the addition of more functions, the stability of the program is the key.
  - More controls and presenters.
    There are currently four controls, and each control has three presenters. Ideally, there should be at least five controls, and each control needs to have five presenters.
  - Extend the features of the control panel.
    The new controls and presenters should be added. And different input and documents for each control should be provided as well.
- Additional features and possible extensions that could be completed
  - Create separate presenters for data values.
    Create separate presenters for data values, so the same presenter can display values differently, e.g., colors can be presented as color patches or as color names; numbers can be presented as digits or words, etc.
  - Create an interface with the web front-end framework.
    Create and interface with the web front-end framework to support different "good-looking" presenters from different well-developed front-end frameworks.
  - Create a semi-realistic app that uses semantic controls.
    Create a semi-realistic app that uses semantic controls and show the app with two different sets of control presenters. This one would require designing a semi-realistic looking app.
  - Add SWT jar file into UI selector jar file
    Add the SWT jar file into UIselector jar file, so that users no longer need to download and import the SWT jar file before using our library.

Design and Software Architecture

UiSelectors was created in the Java programming language on the foundation of Standard Widget Toolkit and the architecture that follows the Model-View-Controller dynamics. Our library consists of three main types of classes: Parent, child, and two helper classes. The parent classes, named SemanticControl and the UiView, contain the foundations and commonalities between all the Models (Controls) and the Views. Next comes the child classes. These make up the majority of our library. We created four (4) children in the SemanticControl Class: OneFromN, MfromN, OnAndOff, and ValueInRange. All of these are the specific controls, or the models in MVC architecture. They all contain the specific components and methods that are independent from the parent class. We also created a total of eleven (11) children to the UiView class. These presenters all belong to specific controls. Each control has three presenters except for the ValueInRange, which only contains two.

One thing that might not be obvious, is that regarding the MVC architecture, our presenters are a combination of the View and Controller. They contain all the Ui components as well as the code that interacts with the semantic controls. When the buttons are clicked on the presenters, the values are sent and stored in the associated semantic control. The control then sends out that new information to update all of its presenters. Lastly, we created two helper classes: UiRunner and the Views enum. The UiRunner class is what launches our presenters and allows us to run multiple semantic controls at the same time. The Views enum is a list of all of our views and they are used in order to check the validity of a presenter when being added to a model. All of these classes are laid out in the class hierarchy below:

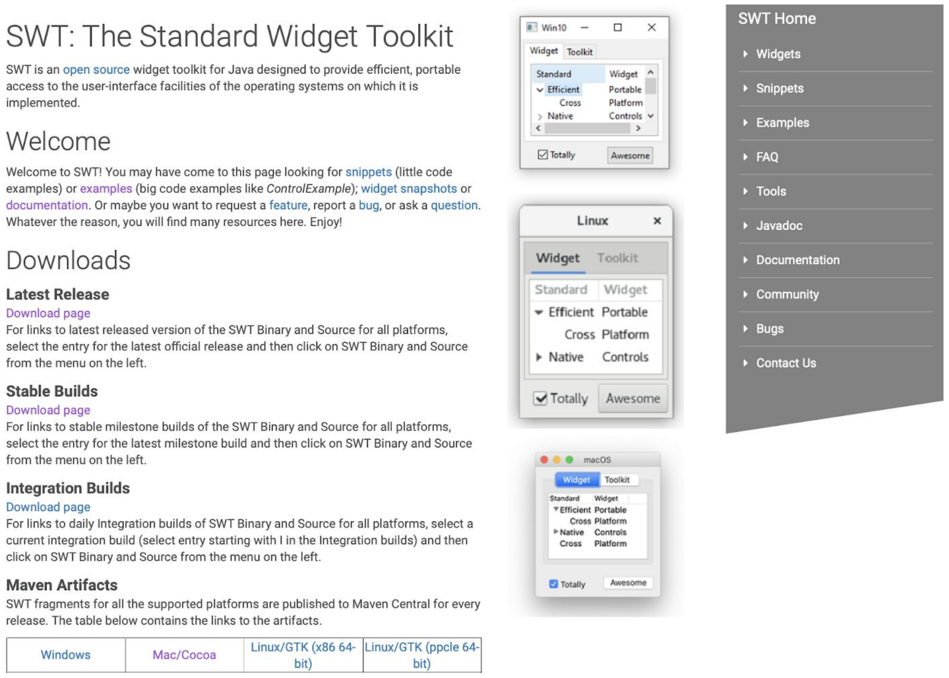


Build Instructions

Download the Standard Widget Toolkit

The Standard Widget Toolkit (SWT) is an open source project. However Eclipse SWT plugin does not bundle with SWT source code. We have to manually download it.
Here are the steps to download the SWT Source Code:

1. Visit the Standard Widget Toolkit (SWT) Official Website *http://www.eclipse.org/swt/*
2. Under Downloads, select your operating system environment (Windows, Linux, MacOS...), Click to download the .zip of SWT for your platform from the *SWT homepage*.



SWT: The Standard Widget Toolkit

SWT is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented.

Welcome

Welcome to SWT! You may have come to this page looking for snippets (little code examples) or examples (big code examples like *ControlExample*); widget snapshots or documentation. Or maybe you want to request a feature, report a bug, or ask a question. Whatever the reason, you will find many resources here. Enjoy!

Downloads

Latest Release
Download page
For links to latest released version of the SWT Binary and Source for all platforms, select the entry for the latest official release and then click on SWT Binary and Source from the menu on the left.

Stable Builds
Download page
For links to stable milestone builds of the SWT Binary and Source for all platforms, select the entry for the latest milestone build and then click on SWT Binary and Source from the menu on the left.

Integration Builds
Download page
For links to daily Integration builds of SWT Binary and Source for all platforms, select a current integration build (select entry starting with I in the Integration builds) and then click on SWT Binary and Source from the menu on the left.

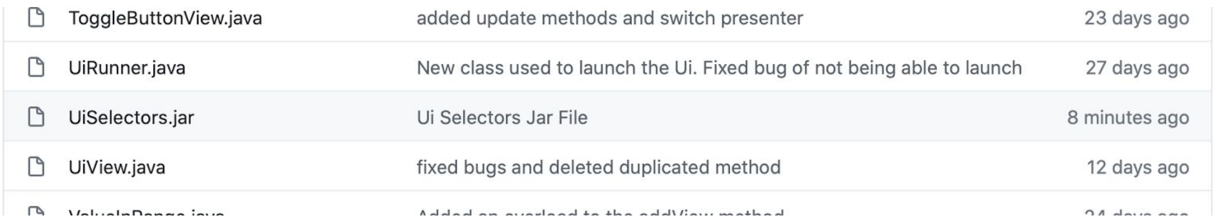
Maven Artifacts
SWT fragments for all the supported platforms are published to Maven Central for every release. The table below contains the links to the artifacts.

Windows	Mac/Cocoa	Linux/GTK (x86 64-bit)	Linux/GTK (ppcle 64-bit)

3. Once downloaded to your machine, unzip the file, and proceed to the Adding User Libraries Section.

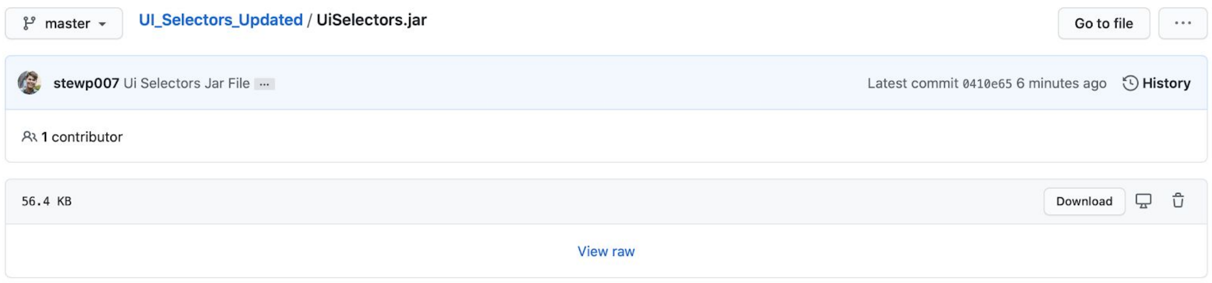
Downloading the UiSelectors.jar file

1. First thing is to go to the GitHub repository and find the UiSelectors.jar file.
2. Once found, click on the UiSelectors.jar file



ToggleButtonView.java	added update methods and switch presenter	23 days ago
UiRunner.java	New class used to launch the Ui. Fixed bug of not being able to launch	27 days ago
UiSelectors.jar	Ui Selectors Jar File	8 minutes ago
UiView.java	fixed bugs and deleted duplicated method	12 days ago
ValueInRange.java	Added an overload to the addView method	24 days ago

3. Then click download to save it to your local machine.

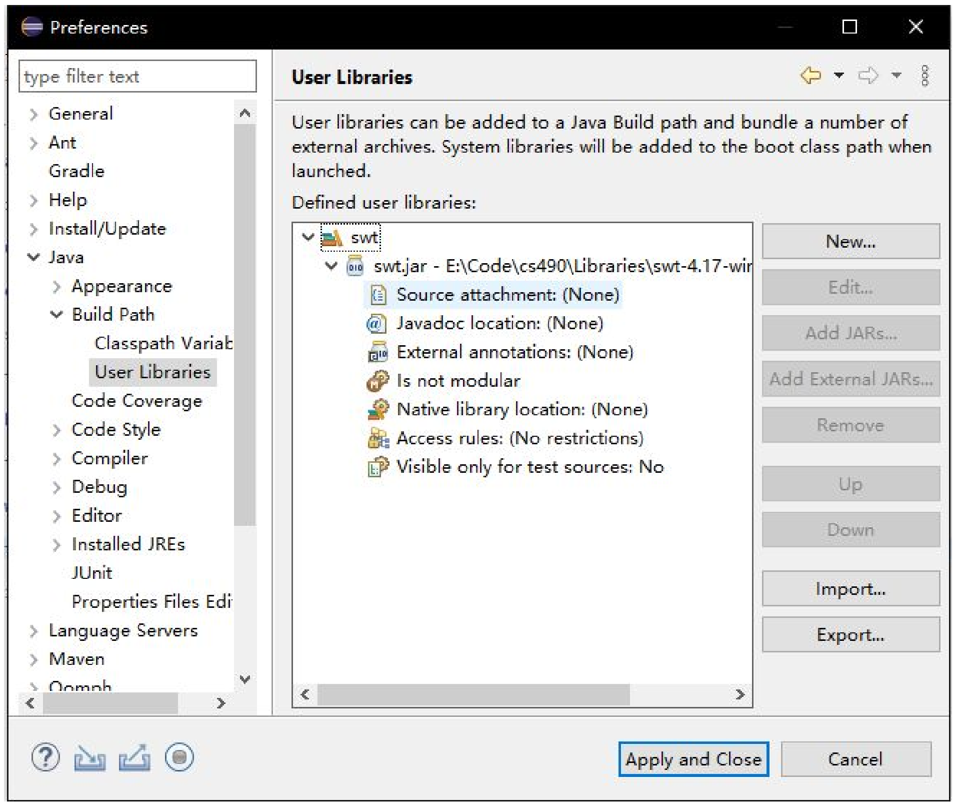

master
UI_Selectors_Updated / UiSelectors.jar
Go to file
stewp007 Ui Selectors Jar File
Latest commit 0410e65 6 minutes ago
History
1 contributor
56.4 KB
Download
View raw

4. From there, follow the next steps in order to add it as a User Library to your IDE (Eclipse)

Adding User Libraries in Eclipse

This guide shows you how to add the UI Selector Updated and SWT libraries in Eclipse, and add this library to the build path for your project. If your IDE of choice is not Eclipse, please refer to the documentation provided by your specific IDE.

To get started, open the "Preferences" window in Eclipse. Navigate to "Java » Build Path » User Libraries" on the left-hand side and click the "New" button. Enter the library name and click the "OK" button(If it is the SWT jar file, make sure the name is “swt”, similarly put “UiSelectors” for the UiSelectors library). Then, you need to click the "Add External JARs..." button to add the jar file. Browse to the jar file(s) required for the library and click the "Open" button. By now, you should be seeing something similar to:


Preferences

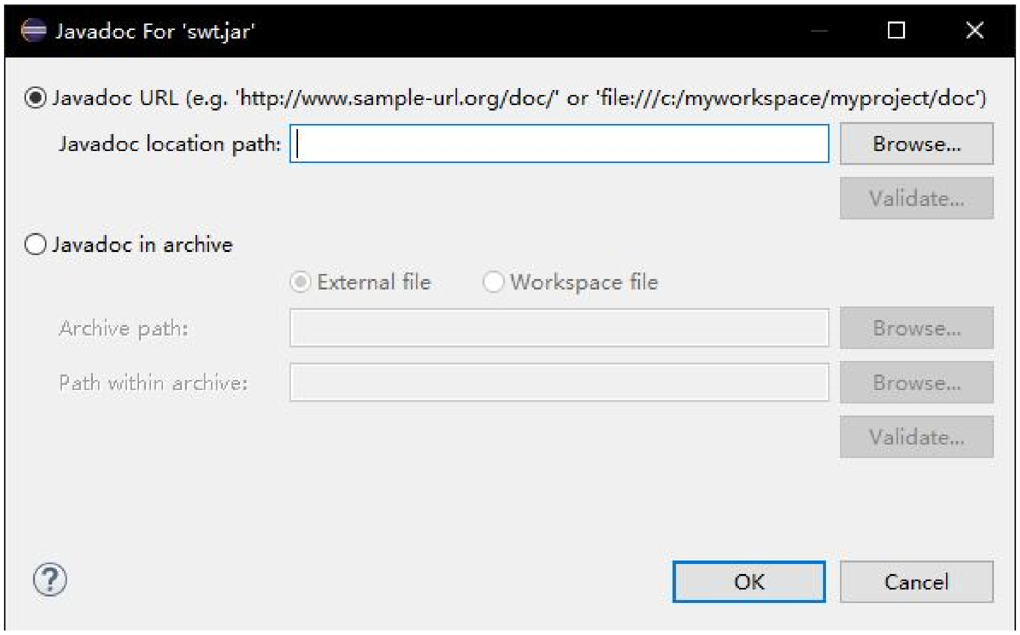
User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

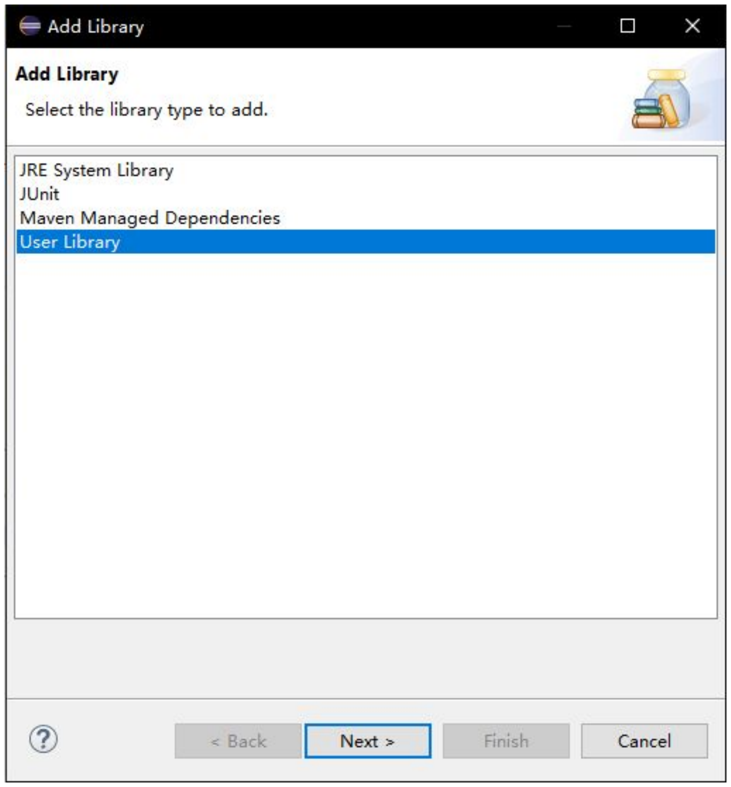
- swt
  - swt.jar - E:\Code\cs490\Libraries\swt-4.17-wir
    - Source attachment: (None)
    - Javadoc location: (None)
    - External annotations: (None)
    - Is not modular
    - Native library location: (None)
    - Access rules: (No restrictions)
    - Visible only for test sources: No

For the SWT library, if you know where the API documentation is located on your computer, go ahead and select the "Javadoc location:" entry and click the "Edit" button. Browse to the folder or directory containing the API documentation and click the "Validate" button:

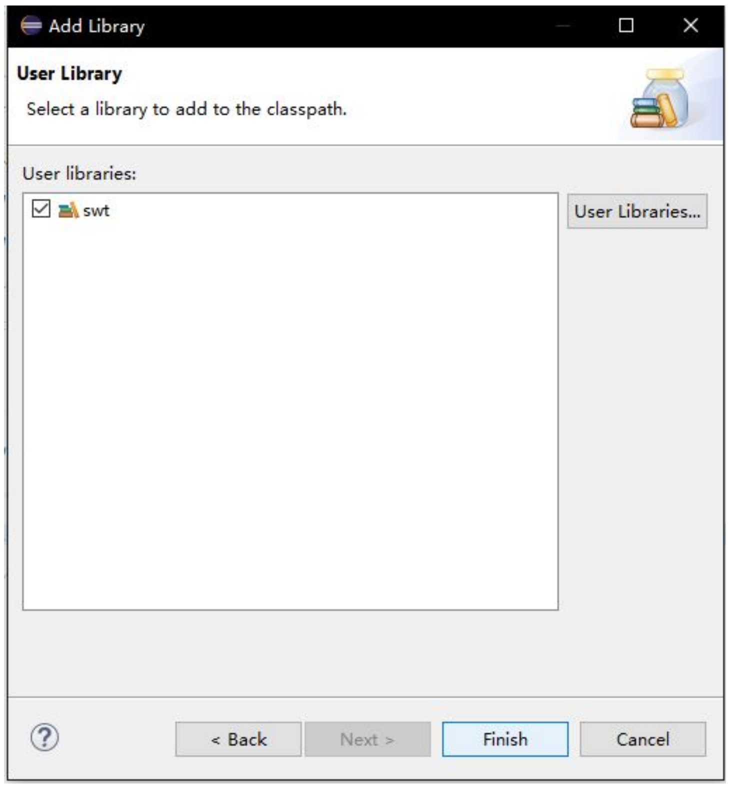


Configuring Build Path:

You need to add the UI Selector Updated and SWT libraries to the build path for the project that uses it. Right-click the project in the package explorer and select "Build Path » Add Libraries..." from the pop-up menu. From the dialog window that pops up, select "User Library" and click the "Next" button.



On the following dialog window, select the user libraries you want to add and click "Finish":

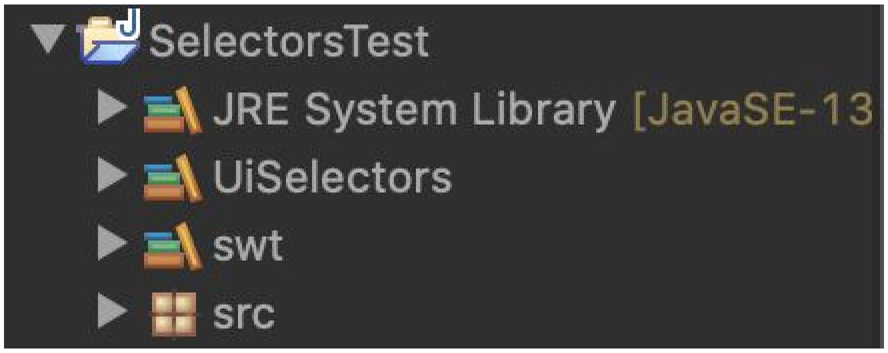


These processes are the same for the SWT and UiSelectors libraries. Once both libraries are added to the build path of your project, you are all set and ready to start using our Semantic Controls.

Project Testing

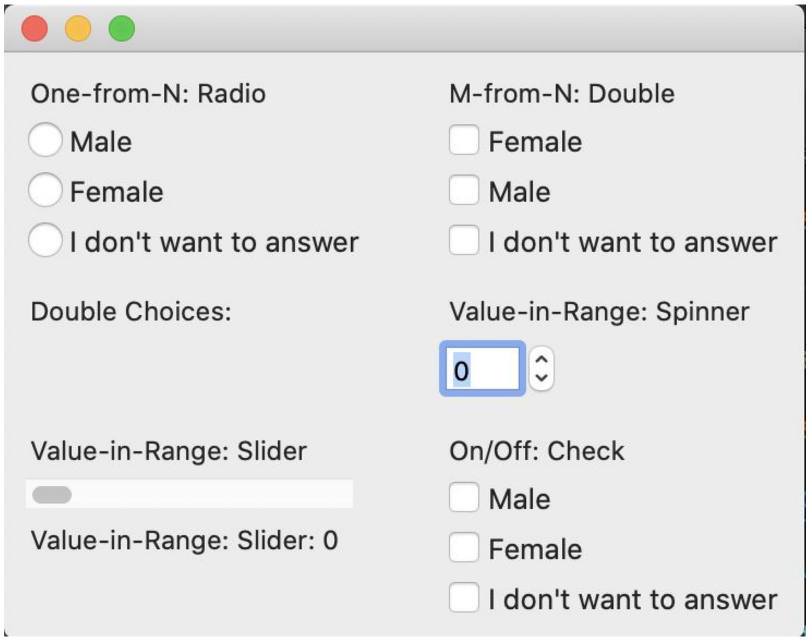
We have followed the above instructions and added the SWT user library as well as the UiSelectors library to the build path of a Test-project and the intended functionality remained the same. We were able to instantiate and use all of our models and presenters. Shown below is the package explorer as well as the test output with all of our models and some of the presenters being demonstrated.

The Package Explorer of our Test Project:



```
▼ SelectorsTest
    ▶ JRE System Library [JavaSE-13
    ▶ UiSelectors
    ▶ swt
    ▶ src
```

The Output of our test file using out UiSelectors Library:

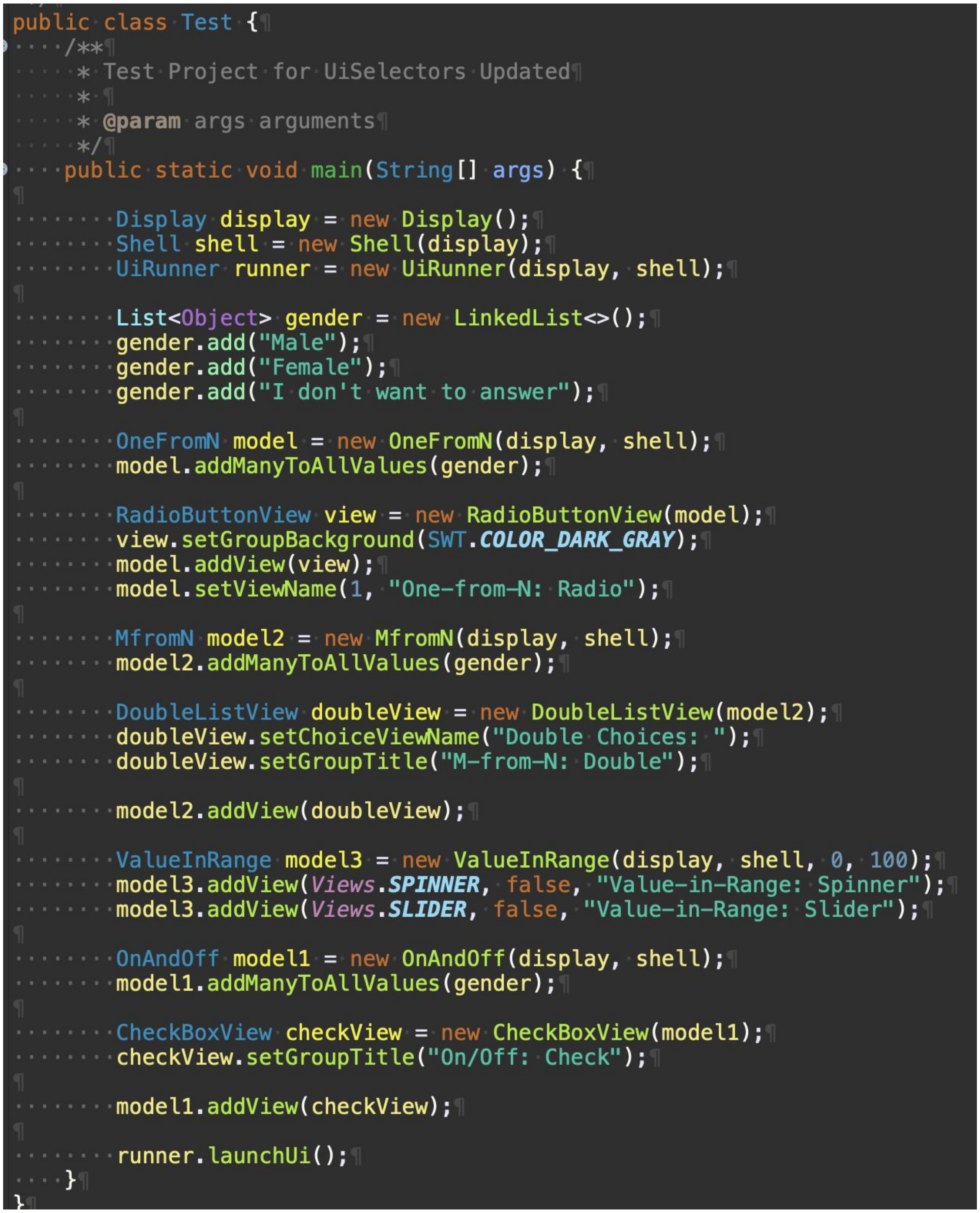


```
One-from-N: Radio              M-from-N: Double
○ Male                         ☐ Female
○ Female                       ☐ Male
○ I don't want to answer       ☐ I don't want to answer

Double Choices:                Value-in-Range: Spinner
                               0

Value-in-Range: Slider         On/Off: Check
                               ☐ Male
Value-in-Range: Slider: 0      ☐ Female
                               ☐ I don't want to answer
```

The Test file used to create the test output:



```
public class Test {
    /**
     * Test Project for UiSelectors Updated
     *
     * @param args arguments
     */
    public static void main(String[] args) {

        Display display = new Display();
        Shell shell = new Shell(display);
        UiRunner runner = new UiRunner(display, shell);

        List<Object> gender = new LinkedList<>();
        gender.add("Male");
        gender.add("Female");
        gender.add("I don't want to answer");

        OneFromN model = new OneFromN(display, shell);
        model.addManyToAllValues(gender);

        RadioButtonView view = new RadioButtonView(model);
        view.setGroupBackground(SWT.COLOR_DARK_GRAY);
        model.addView(view);
        model.setViewName(1, "One-from-N: Radio");

        MfromN model2 = new MfromN(display, shell);
        model2.addManyToAllValues(gender);

        DoubleListView doubleView = new DoubleListView(model2);
        doubleView.setChoiceViewName("Double Choices: ");
        doubleView.setGroupTitle("M-from-N: Double");

        model2.addView(doubleView);

        ValueInRange model3 = new ValueInRange(display, shell, 0, 100);
        model3.addView(Views.SPINNER, false, "Value-in-Range: Spinner");
        model3.addView(Views.SLIDER, false, "Value-in-Range: Slider");

        OnAndOff model1 = new OnAndOff(display, shell);
        model1.addManyToAllValues(gender);

        CheckBoxView checkView = new CheckBoxView(model1);
        checkView.setGroupTitle("On/Off: Check");

        model1.addView(checkView);

        runner.launchUi();
    }
}
```