SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – V-Rich Internet Applications – SCS1401

V. Building Ria With GWT

GWT Panels and Layouts - Event Handling - Internationalisation - Advanced GWT - RPC and AJAX – Writing a service implementation - Handling browser back button functionality - MVP Design pattern.

GWT Layout Panels

- Layout Panel helps us to design the user interface of the panels.
- This panel helps to fit all the content inside the windows.
- Every Panel widget inherits properties from Panel class which in turn inherits properties from Widget class and which in turn inherits properties from UIObject class.

Types of Layout Panels

1.Flow Panel

2. Horizontal Panel

3. Vertical Panel

4. Horizontal Split Panel

5. Vertical Split Panel

6. Flex Table

7. Grid

8. Deck Panel

9. Dock Panel

10. HTML Panel

11. Tab Panel

12. Composite

13. Simple Panel

14. Scroll Panel

15. Focus Panel

16. Form Panel

17. Popup Panel

18. Dialog Box

Flow Panel

- This widget represents a panel that formats its child widgets using the default HTML layout behavior shown in Figure 5.1.

  Class Declaration

  public class FlowPanel extends ComplexPanel

  implements InsertPanel.ForIsWidget

Constructor

FlowPanel()

Class Methods

- **void add(Widget w)**-Adds a new child widget to the panel.
- **void clear()**-Removes all child widgets.
- **void insert(Widget w, int beforeIndex)**

EXAMPLE

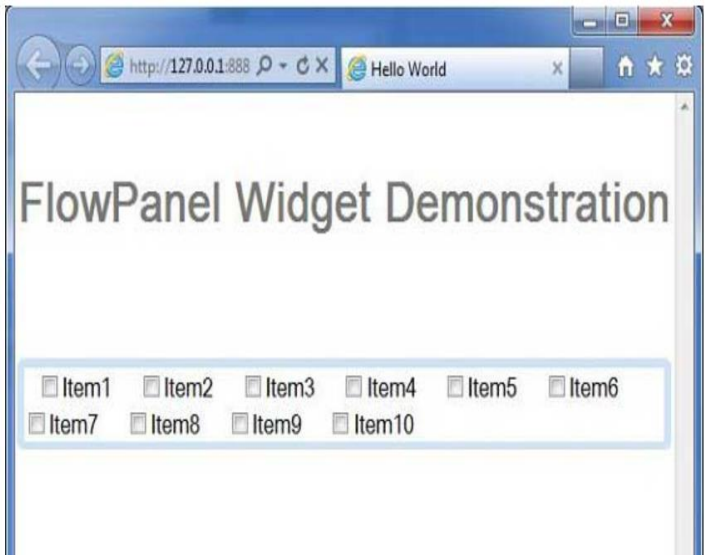

Figure 5.1 Flow Layout

HorizontalPanel

- The **HorizantalPanel** widget represents a panel that lays all of its widgets out in a single horizontal column in figure 5.2.

  Class Declaration

  public class HorizontalPanel extends CellPanel

  implements HasAlignment, InsertPanel.ForIsWidget

Class Constructors

HorizontalPanel()

Class Methods

- **void add(Widget w)-** Adds a child widget.
- **HasHorizontalAlignment.HorizontalAlignmentConstant getHorizontalAlignment()**- Gets the horizontal alignment.
- **HasVerticalAlignment.VerticalAlignmentConstant getVerticalAlignment()**-Gets the vertical alignment.
- **void insert(Widget w, int beforeIndex)**- Inserts a child widget before the specified index.
- **boolean remove(Widget w)-** Removes a child widget.

Figure 5.2 Horizontal Layout

Vertical Panel

- The **VerticalPanel** widget represents a panel that lays all of its widgets out in a single vertical in Figure 5.3.
- row.

Class Declaration

public class VerticalPanel extends CellPanel

implements HasAlignment, InsertPanel.ForIsWidget

Class Constructors

VerticalPanel()

Class Methods

- **void add(Widget w)**-Adds a child widget.
- **boolean remove(Widget w)**-Removes a child widget.

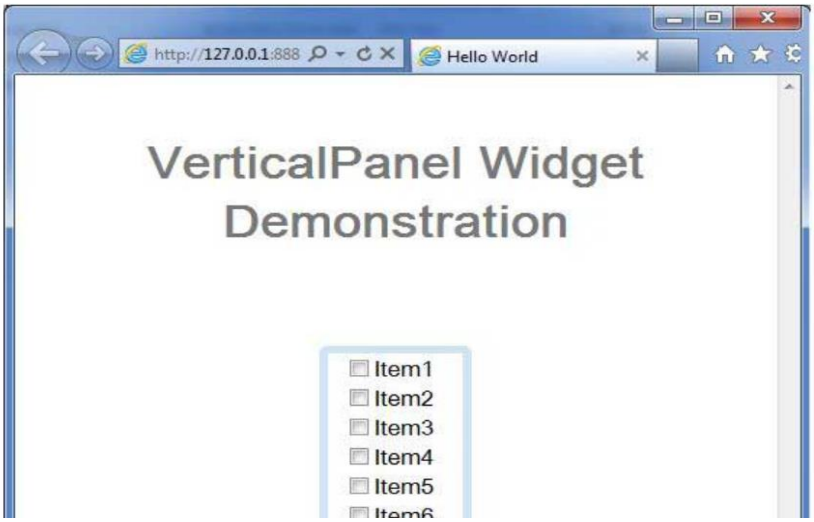

Figure 5.3 Vertical Panel

Horizontal Split Panel

- The **HorizantalSplitPanel** widget represents a panel that arranges two widgets in a single horizontal row and allows the user to interactively change the proportion of the width dedicated to each of the two widgets.

Class Declaration

public final class HorizontalSplitPanel extends Panel

Constructors

HorizontalSplitPanel()

HorizontalSplitPanel(HorizontalSplitPanel.Resources resources)

Class Methods

- **void add(Widget w)**
- **Widget getLeftWidget()**-Gets the widget in the left side of the panel.
- **Widget getRightWidget()**-Gets the widget in the right side of the panel.
- **boolean isResizing()**-Indicates whether the split panel is being resized.
- **boolean remove(Widget widget)**-Removes a child widget.
- **void setLeftWidget(Widget w)**-Sets the widget in the left side of the panel.
- **void setRightWidget(Widget w)**-Sets the widget in the right side of the panel.

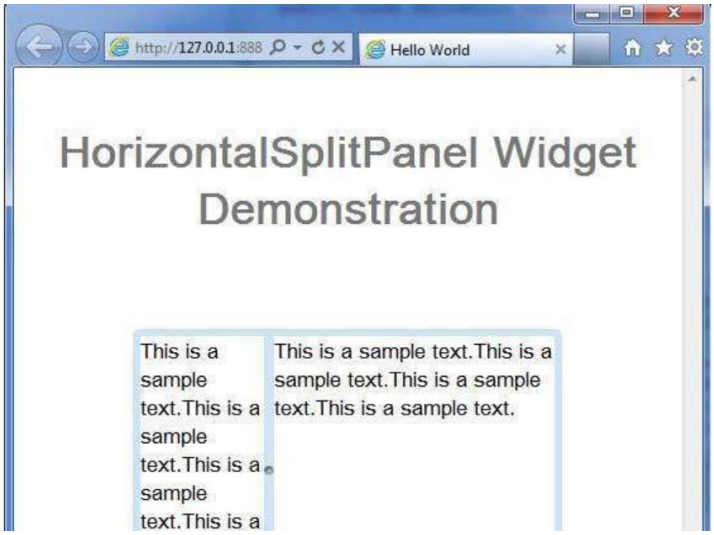

Figure 5.4 Horizontal Split Panel

VerticalSplit Panel

- The **VerticalSplitPanel** widget represents a panel that arranges two widgets in a single vertical column and allows the user to interactively change the proportion of the height dedicated to each of the two widgets.
- Widgets contained within a VerticalSplitterPanel will be automatically decorated with scrollbars when necessary.

Class Declaration

public final class VerticalSplitPanel extends Panel

FlexTable

- The **FlexTable** widget represents a flexible table that creates cells on demand.
- It can be jagged (that is, each row can contain a different number of cells) and individual cells can be set to span multiple rows or columns.

Class Declaration

public class FlexTable extends HTMLTable

Class Methods

void addCell(int row)-Appends a cell to the specified row.

int getCellCount(int row)-Gets the number of cells on a given row.

int getRowCount()-Gets the number of rows.

void insertCell(int beforeRow, int beforeColumn)-Inserts a cell into the FlexTable.

int insertRow(int beforeRow)-Inserts a row into the FlexTable.

Grid

- This widget represents a A rectangular grid that can contain text, html, or a child Widget within its cells.
- It must be resized explicitly to the desired number of rows and columns.

Class Declaration

public class Grid extends HTMLTable

Constructors

Grid()

Grid(int rows, int columns)

Class Methods

- **boolean clearCell(int row, int column)-**Replaces the contents of the specified cell with a single space.
- **protected Element createCell()-**Creates a new, empty cell.
- **int getCellCount(int row)-**Return number of columns.
- **int getColumnCount()-**Gets the number of columns in this grid.
- **int getRowCount()-**Return number of rows.
- **void removeRow(int row)**-Removes the specified row from the table.
- **void resize(int rows, int columns)-**Resizes the grid.

Deck Panel

- Panel that displays all of its child widgets in a 'deck', where only one can be visible at a time. It is used by TabPanel.

Class Declaration

public class DeckPanel extends ComplexPanel

implements HasAnimation, InsertPanel.ForIsWidget

Class Methods

void add(Widget w)-Adds a child widget.

int getVisibleWidget()-Gets the index of the currently-visible widget.

DockPanel

- This widget represents a panel that lays its child widgets out "docked" at its outer edges, and allows its last widget to take up the remaining space in its center is shown in Figure 5.5.

Class Declaration

- public class DockPanel extends CellPanel implements HasAlignment

Class Methods

void add(Widget widget, DockPanel. DockLayoutConstant direction)- Adds a widget to the specified edge of the dock.

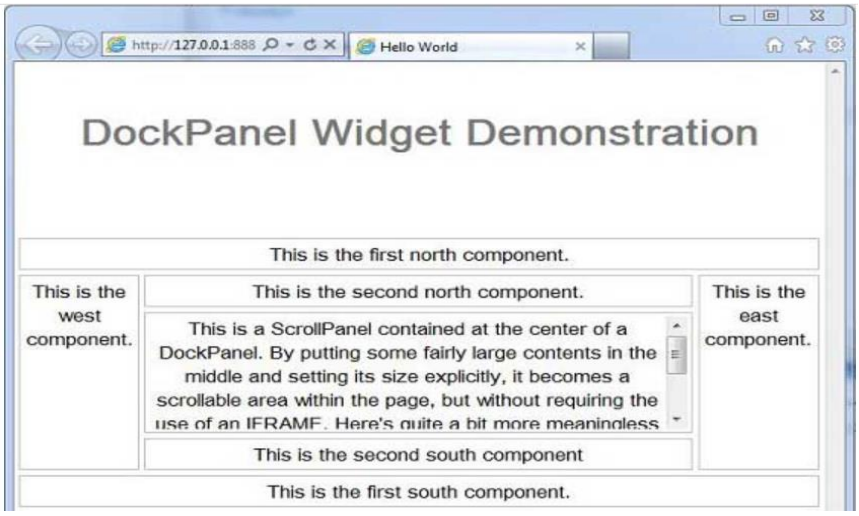

Figure 5.5 Dock Panel

HTMLPanel

- This widget represents a panel that contains HTML, and which can attach child widgets to identified elements within that HTML is shown in Figure 5.6.

Class

public class HTMLPanel extends ComplexPanel

Class Methods

- **void add(Widget widget, Element elem)**
- **void addAndReplaceElement(Widget widget, Element toReplace)-**Adds a child widget to the panel, replacing the HTML element.
- **void addAndReplaceElement(Widget widget, java.lang.String id)-**Adds a child widget to the panel, replacing the HTML element specified by a given id.

Figure 5.6 HTML Panel

TabPanel

- The **TabPanel** widget represents panel that represents a tabbed set of pages, each of which contains another widget.
- Its child widgets are shown as the user selects the various tabs associated with them.

Class Declaration

public class TabPanel extends Composite

Methods

- **void add(Widget w)**
- **void add(Widget w, Widget tabWidget)**
- **TabBar getTabBar()**

Composite Widget

- The **Composite** widget is a type of widget that can wrap another widget, hiding the wrapped widget's methods is shown in Figure 5.7.

Class

public abstract class Composite extends Widget

Class Methods

- **protected Widget getWidget()-**Provides subclasses access to the topmost widget that defines this composite.
- **protected void initWidget(Widget widget)**-Sets the widget to be wrapped by the composite.
- **boolean isAttached()**-Determines whether this widget is currently attached to the browser's document.

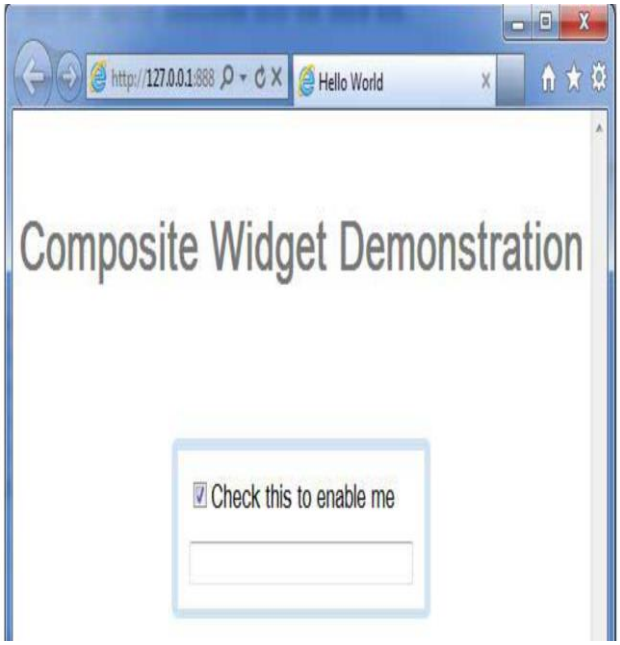

Figure 5.7 Composite Widget

SimplePanel Widget

- The **SimplePanel** widget represents a base class for panels that contain only one widget in Figure 5.8.

  Class Declaration

  public class SimplePanel extends Panel implements HasOneWidget

  Class Methods

  - **void add(Widget w)**-Adds a widget to this panel.
  - **Widget getWidget()**-Gets the panel's child widget.
  - **boolean remove(Widget w)**-Removes a child widget.
  - **void setWidget(IsWidget w)**-Set the only widget of the receiver, replacing the previous widget if there was one.

Figure 5.8 Simple Panel Widget

ScrollPanel Widget

- The **ScrollPanel** widget represents a simple panel that wraps its contents in a scrollable area in figure 5.9.

  Class Declaration

  - public class ScrollPanel extends SimplePanel implements SourcesScrollEvents, HasScrollHandlers, RequiresResize, ProvidesResize

  Class Methods

  - **void ensureVisible(UIObject item)-**Ensures that the specified item is visible, by adjusting the panel's scroll position.
  - **int getHorizontalScrollPosition()-**Gets the horizontal scroll position.
  - **int getScrollPosition()-**Gets the vertical scroll position.
  - **void scrollToBottom()-**Scroll to the bottom of this panel.
  - **void scrollToLeft()-**Scroll to the far left of this panel.
  - **void scrollToRight()-**Scroll to the far right of this panel.

Figure 5.9 Scroll Panel Widget

FocusPanel Widget

- The **FocusPanel** widget represents a simple panel that makes its contents focusable, and adds the ability to catch mouse and keyboard events in Figure 5.10.

  Class Declaration

  public class FocusPanel extends SimplePanel

  Class Methods

  - **void addClickListener(ClickListener listener)**

void addMouseListener(MouseListener listener)

Figure 5.10 FocusPanel Widget

FormPanel Widget

- The **FormPanel** widget represents a panel that wraps its contents in an HTML <FORM> element in Figure 5.11.

  Class

  public class FormPanel extends SimplePanel

  Class Methods

  - **void add Form Handler (FormHandler handler)**
  - **boolean onFormSubmit()**

Figure 5.11 FormPanel Widget

PopupPanel

- The **PopupPanel** widget represents a panel that can **pop up** over other widgets Figure 5.12.

  Class Declaration

  - public class PopupPanel extends SimplePanel

  Class Methods

  - **void addPopupListener(PopupListener listener)**
  - **void center()**

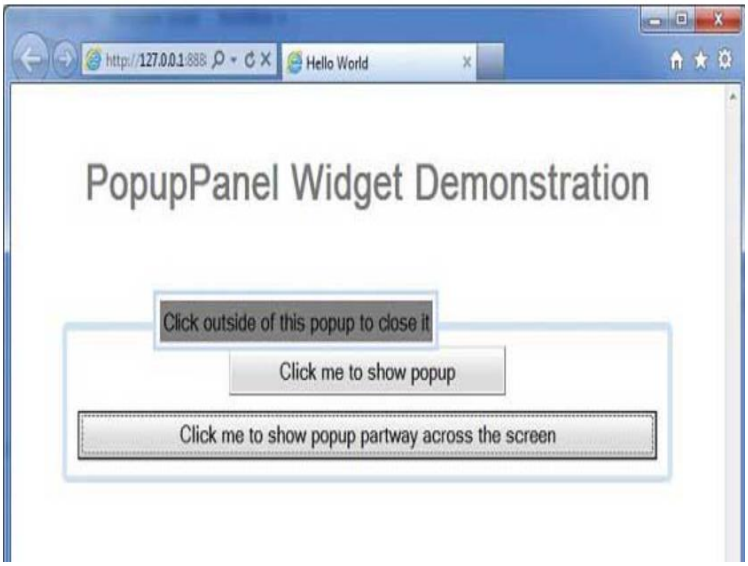

Figure 5.12 Popup Panel

DialogBox Widget

- The **DialogBox** widget represents a form of popup that has a caption area at the top and can be dragged by the user shown in Figure 5.13.

  Class

  - public class DialogBox extends DecoratedPopupPanel

  Methods

  - **void onMouseMove(Widget sender, int x, int y)**
  - **void onMouseUp(Widget sender, int x, int y)**

Figure 5.13 DialogBox Widget

Event Handling in GWT

- GWT provides a list of interfaces corresponding to various possible events.
- A listener interface defines one or more methods that the widget calls to announce an event.
- For example, the **Button** class publishes **click events** so you will have to write a class to implement *ClickHandler* to handle **click** event.

Event Handler Interfaces

- All GWT event handlers have been extended from *EventHandler* interface and each handler has only a single method with a single argument.
- Each **event** object have a number of methods to manipulate the passed event object.
    - public class MyClickHandler implements ClickHandler { @Override public void onClick(ClickEvent event) { Window.alert("Hello World!"); } }
- **BlurHandler-void on Blur(Blur Event event);**
- **ChangeHandler-void on Change(ChangeEvent event);-**Called when a change event is fired.
- **ClickHandler-void on Click(ClickEvent event);**
- **CloseHandler-void on Close(CloseEvent<T> event);-**Called when CloseEvent is fired.
- **Context Menu Handler-void on Context Menu(Context Menu Event event);**
- **Double Click Handler-void on Double Click(Double Click Event event);-**Called when a Double Click Event is fired.
- **Error Handler-void on Error(Error Event event);-**Called when Error Event is fired.
- **Focus Handler-void on Focus(Focus Event event);**
- **FormPanel.SubmitHandler-void on Submit(Form Panel.Submit Event event);-** Fired when the form is submitted.

- **Key Down Handler-void on Key Down(Key Down Event event);-**Called when KeyDownEvent is fired.
- **KeyPressHandler-void on KeyPress(KeyPressEvent event);**-Called when KeyPressEvent is fired.
- **KeyUpHandler-void on KeyUp(KeyUpEvent event);-**Called when KeyUpEvent is fired.
- **LoadHandler-void on Load(LoadEvent event);**-Called when LoadEvent is fired.
- **MouseDownHandler-void on MouseDown(MouseDownEvent event);-**Called when MouseDown is fired.

GWT Internationalization

- It is similar to Java programming language, where internationalization is implemented by means of Resource Bundles, Where **.properties** file is created for each locale that needs to be supported.
- Internationalization is changing the language of the text based on the locale. For example, the browser should display the website content in Hindi for a user sitting in India and French for the user accessing the website from France.
- **Types of Internationalization Techniques**
    1. Static String Internationalization
    2. Dynamic String Internationalization
    3. Localizable Interface

Static String Internationalization

- It is a good technique for translating both constant and parameterized strings.
- It is the simplest technique to implement as it requires very less over head.
- It uses standard Java properties files to store translated strings and parameterized messages.

Dynamic String Internationalization

- Dynamic string internationalization is slower but more flexible than static string internationalization.
- Applications using this technique look **like** localized strings in the module's home page. Due to this technique they do not need to be recompiled when you add a new locale.

Localizable Interface

- It is the most powerful technique to implement the interface.
- It is an advanced internationalization technique that is **used rarely.**

- We require advance level to implement Localizable interface **for** simple string substitution. It also creates localized versions of custom types.

Advanced GWT

- Advanced GWT Components is an extension of the standard Google Web Toolkit library.
- It allows making rich web interfaces extremely quickly even if you're not skilled in DHTML and JavaScript programming.
- Currently the library supports such popular browsers like Internet Explorer, Firefox, Safari, Opera and Chrome.

Widgets

- Editable (updatable) Grid
- Hierarchical Grid
- Tree Grid
- Advanced FlexTable
- SimpleGrid
- AdvancedTabPanel
- SingleBorder
- RoundCornerBorder
- Pager
- Grid Toolbar
- Grid Panel
- Master-Detail Panel
- Date Picker
- Combo Box
- Suggestion Box

Advanced GWT features

- Improved browser independent table body scrolling
- Editable grids
- Hierarchical and master-detail representations of complicated data models
- Multiple and single row selections
- Client side paging and sorting
- Server side data loading

- Client and server side content rendering
- Full keyboard control
- Flexible tab panels
- Customizable borders API
- Localization and internationalization
- <thead> HTML tag support in tables
- <tfoot> HTML tag support in tables
- Splitting to view and model (MVP design)
- Data model events
- Lazy rendering for large drop down lists
- Customizable event management and rendering
- GWT 1.6.4, 1.7.1, 2.0.x, 2.1.x, 2.2.x, 2.3.x, 2.4.x, 2.5.x or higher compatibility
- No dependencies to other third-party libraries
- Theme runtime switching for non-styled widgets
- Latest releases are available in the Maven Central Repository

MVP Design Pattern

- MVP (Model View Presenter) is a design pattern which allows the application developing in GWT to follow MVP architecture in Figure 5.14.
- MVP provides the solution of the problem of complexity for developing application.
- Application development is complex as many developers working on same code due to which all follow same design pattern.

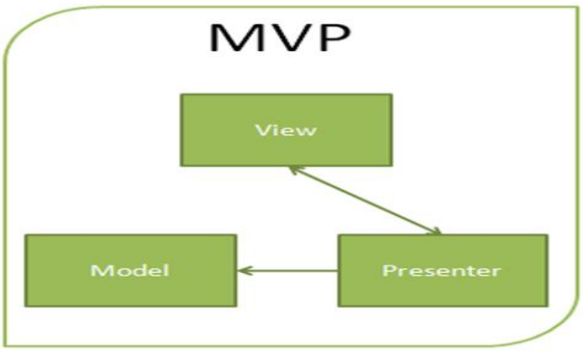

Figure 5.14 MVP

- Model: This segment model consists of data only. It holds within the business object which is to be manipulated and calculated according to application need.

- View: It only consists of view i.e. display the data which is given by presenter. It provides reusability of view code as we can swap the new view very easily. It only deals with the HTML and CSS which also helps in separate testing.
- Presenter: It contains all the logic which is to be implemented in the application development. It communicates with model as well as view. It is complete distinct in operation which provides separate JUnit testing.

MVP Vs MVC

The difference between MVP and MVC is discussed in Table 5.1.

MVP (Model View Presenter)	MVC (Model View Controller)
It is advance form of MVC	It is the basic method to separate project structure.
In this View handles user gesture and call presenter.	In this controller handles user gesture and commands model.
View is dumb i.e. all interaction goes through Presenter.	In this view has some intelligence. It can query the model directly.
It highly supports unit testing.	It provides limited support to unit testing.
It has high degree of loose coupling.	It has fairly loose coupling.
In this presenter will update its associated view.	It identifies which view to update.

Table 5.1 MVP vs MVC

MVP Design Pattern – Benefits

- MVP is a design pattern that breaks your app up into the components Model, View and Presenter.
- The MVP pattern is extremely useful when building large, web-based applications with GWT.
- It helps to make code more readable, and more maintainable.
- It also makes it much easier to implement new features, optimizations, and automated testing.

Model

- Houses all of the data objects that are presented and acted upon within your UI.

- The number and granularity of models is a design decision. In our PhotoApp, we will have one very simple model:
- PhotoDetails - holds data about a photo, including the thumbnail URL, original URL, title, description, tags, and so on.

View

- These are the UI components that display model data and send user commands back to the presenter component.
- It is not a requirement to have a view per model. You may have a view that uses several models, or several views for one model. We will keep things simple for our Photo Application, and will have three views:
    - WelcomeView – A very simple welcome page.
    - PhotoListView - Displays a list of thumbnail photos and their title.
    - PhotoDetailsView - displays the photo together with title and other data and allows the user to change some of those details.

Presenter

- The presenter will hold the complex application and business logic used to drive UIs and changes to the model. It also has the code to handle changes in the UI that the view has sent back.
- Usually for each view, there will be an associated presenter. In our photo application, this means we will have the following three presenters:
- WelcomePresenter - pushes the welcome screen in front of the user, and handles the jump to PhotoListView.
- PhotoListPresenter - drives the thumbnail view.
- PhotoDetailsPresenter - drives the view of the original photo.

Creating Views

- Remember that our view should have no application logic in it, at all. It should be just UI components that the presenter can access to set or get values from.
- All of our views will be implemented as three separate items: a generic interface, a specific interface and an implementation.

```
public interface View extends IsWidget{

void setPresenter(PhotoDetailsPresenter presenter);

}
```

Implementing the views

- Take the detailed view, each implementation implements the setPresenter method

```
public void setPresenter(PhotoDetailsPresenter presenter) {
  this.presenter = presenter;
}
```

Presenters

- Presenters are where all the application logic sits and will have no UI components.
- In a similar way to views, we provide a generic presenter interface, a specific one, and an implementation for each presenter.

Handling browser back button functionality

- GWT History mechanism is similar to the Ajax history implementations such as RSH (Really Simple History).
- Basic idea is to track application internal state in the URL fragment identifier.

Main advantages of this mechanism are:

- It provides browser history reliable.
- It provides good feedback to the user.
- It is bookmarkable i.e., the user can create a bookmark to the current state and save it or can email it etc.

GWT History Syntax

```
public class History extends java.lang.Object
```

GWT History Tokens

- A token is simply a string that the application can parse to return to a particular state.
- This token will be saved in browser history as a URL fragment (in the location bar, after the "#"), and this fragment is passed back to the application when the user goes back or forward in history, or follows a link.
- Example: History token name - javatpoint.
- http://www.example.com/com.example.gwt.HistoryExample/HistoryExample.html#javatpoint

GWT Hyperlink Widgets

- Hyperlinks are convenient to use to incorporate history support into an application. Hyperlink widgets are GWT widgets that look like regular HTML anchors. You can associate a history token with the Hyperlink, and when it is clicked, the history token is automatically added to the browser history stack. The History.newItem(token) step is done automatically.
- Handling an onValueChange() callback

- The first step of handling the onValueChange() callback method in a ValueChangeHandler is to get the new history token with ValueChangeEvent.getValue() then we will parse the token. Once the token is parsed, we can reset the state of the application.
- When the onValueChange() method is invoked, application handles two cases:
- The application was just started and was passed a history token.

  The application is already running and was passed a history token.

AJAX and RPC

Server-side Code

- Everything that happens within your web server is referred to as server-side processing.
- When your application running in the user's browser needs to interact with your server (for example, to load or save data), it makes an HTTP request across the network using a remote procedure call (RPC).
- While processing an RPC, your server is executing server-side code.
- GWT provides an RPC mechanism based on Java Servlets to provide access to server-side resources.
- This mechanism includes generation of efficient client-side and server-side code to serialize objects across the network using deferred binding.

Remote Procedure Calls

- A fundamental difference between AJAX applications and traditional HTML web applications is that AJAX applications do not need to fetch new HTML pages while they execute.
- Because AJAX pages actually run more like applications within the browser, there is no need to request new HTML from the server to make user interface updates.
- However, like all client/server applications, AJAX applications usually do need to fetch data from the server as they execute.
- The mechanism for interacting with a server across a network is called making a remote procedure call (RPC), also sometimes referred to as a server call is shown in Figure 5.15 and Figure 5.16.

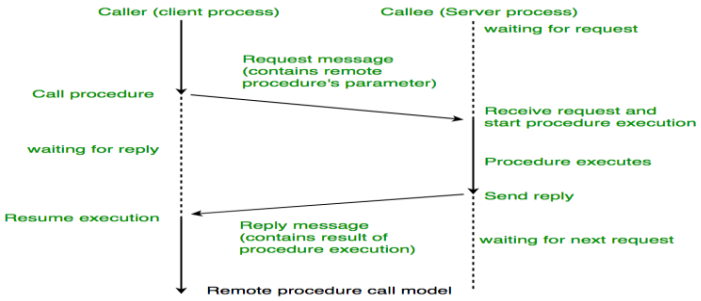

Figure 5.15 RPC

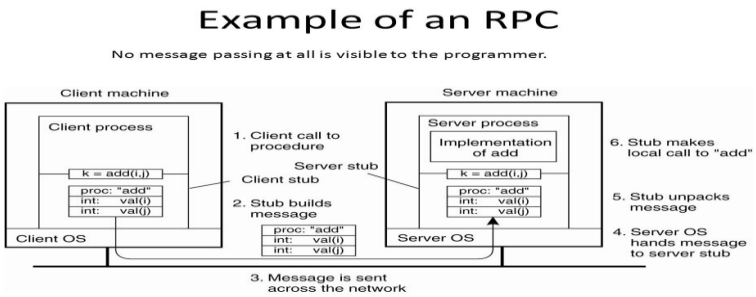

Figure 5.16 Example of an RPC

- GWT RPC makes it easy for the client and server to pass Java objects back and forth over HTTP.
- When used properly, RPCs give you the opportunity to move all of your UI logic to the client, resulting in greatly improved performance, reduced bandwidth, reduced web server load, and a pleasantly fluid user experience.
- The server-side code that gets invoked from the client is often referred to as a service, so the act of making a remote procedure call is sometimes referred to as invoking a service

Creating Services

- In order to define your RPC interface, you need to:
- Define an interface for your service that extends RemoteService and lists all your RPC methods.
- Define a class to implement the server-side code that extends RemoteServiceServlet and implements the interface you created above.
- Define an asynchronous interface to your service to be called from the client-side code

Synchronous Interface

- To begin developing a new service interface, create a client-side Java interface that extends the RemoteService tag interface.

```
package com.example.foo.client;
```

```
import com.google.gwt.user.client.rpc.RemoteService;

public interface MyService extends RemoteService {

  public String myMethod(String s);

}
```

- Any implementation of this service on the server-side must extend RemoteServiceServlet and implement this service interface.

```
package com.example.foo.server;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;

import com.example.foo.client.MyService;

public class MyServiceImpl extends RemoteServiceServlet
implements

  MyService {

 public String myMethod(String s) {

 return s;

        } }
```

- It is not possible to call this version of the RPC directly from the client.
- You must create an asynchronous interface to all your services as shown below.

Asynchronous Interfaces

- Before you can actually attempt to make a remote call from the client, you must create another client interface, an asynchronous one, based on your original service interface.
- Continuing with the example above, create a new interface in the client subpackage:

```
package com.example.foo.client;

interface MyServiceAsync {

  public void myMethod(String s, AsyncCallback<String> callback);

}
```

- An interaction is synchronous if the caller of a method must wait for the method's work to complete before the caller can continue its processing.
- An interaction is asynchronous if the called method returns immediately, allowing the caller to continue its processing without delay.

References

1. Federico Kerek , " Essential GWT: Building for the Web with Google Web Toolkit 2 ", Addison-Wesley Professional,2010.