The Google Web Toolkit (GWT):
The Model-View-Presenter (MVP)
Architecture – Official MVP Framework
(GWT 2.5 Version)

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/gwt.html

Customized Java EE Training: http://courses.coreservlets.com/
GWT, Java 7 and 8, PrimeFaces, HTML5, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, REST, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Ajax & GWT training, see training courses at http://courses.coreservlets.com/.

Taught by the author of Core Servlets and JSP,
More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

• Courses developed and taught by Marty Hall
  – JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  – Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
• Courses developed and taught by coreservlets.com experts (edited by Marty)
  – Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services
Contact hall@coreservlets.com for details
Topics in This Section

• **Motivation**
  – What’s wrong with what we already have?

• **Pros and cons**

• **Main components of MVP framework**
  – Model
  – View
  – ClientFactory
  – Activity
  – Place
  – PlaceHistoryMapper
  – ActivityMapper
So What’s Wrong with What We Already Had?

- History management is all over the app
  - Presenters hard code and/or know too much about history
- Defining the Display (i.e., view) interface inside the presenter doesn’t allow for bi-directional communication
  - Impossible to use @UiHandler shortcut with UiBinder
- View creation means a lot of DOM calls
  - Expensive and slow!
    - If no state is saved in the view, why create new view every time? Just reuse the same view instance.

So What’s Wrong with What We Already Had? (Continued)

- Recycling view instances is technically not part of MVP but very useful and fits well into MVP
  - DOM operations are expensive so we try to avoid them
  - Note that this implies that no state should ever be stored in a view
- Keep the presenters disposable
  - No DOM code resides there, so they are very lightweight and cheap to recreate every time
    - No need to reset their state
- GWT MVP elegantly abstracts history handling
- All these are advantages of GWT MVP Framework
  - Besides all the same advantages of MVP in general
Disadvantages of GWT MVP

• All the same as before, but now a REALLY steep learning curve!
  – Can feel a bit like this at first:

• Bottom line:
  – Give yourself time to get used to all the standard steps

Components of MVP Framework
Main Components of GWT MVP Framework

• **Model**
  – Responsible for holding raw data
  – No business logic allowed (just like any model really)

• **Example:**
  ```java
  public class Customer {
      // can be Serializable
      private String firstName, lastName;

      public Customer(String fName, String lName) {...}
      public String getFirstName() { ...}
      public void setFirstName(String firstName) {...}
      ...
  }
  ```

Main Components of GWT MVP Framework (Continued)

• **View**
  – Responsible for displaying data
  – No business logic allowed
  – Based on a custom standalone interface which defines the methods of this view
    • Inside view interface, we define the Presenter interface for the concrete presenter that will be associated with this view
    • Allows UiBinder to pass through UiHandler methods to the presenter
  – View interface extends isWidget interface that defines asWidget method

• **See next slide for example**
Main Components of GWT MVP Framework (Continued)

• Example View interface:

```java
public interface HelloView extends isWidget {
    void setName(String name);
    void setPresenter(Presenter presenter);

    public interface Presenter {
        String getName();
        ...
    }
}
```

Main Components of GWT MVP Framework (Continued)

• ClientFactory
  – Not technically part of MVP, but very useful to facilitate creation of a view once and only once
  – The rest of the app uses ClientFactory to obtain view references

• Example:

```java
public interface ClientFactory {
    EventBus getEventBus();
    HelloView getHelloView();
    ...
}
```
Main Components of GWT MVP Framework (Continued)

• **Activity**
  – It’s a Presenter with “bells and whistles” built in
    • Has lifecycle methods to hook into when:
      – User is trying to navigate away from this activity
      – Activity’s widget (i.e., the view it controls) is removed from user’s view
      – Etc.
    – Implements the Presenter interface defined in the associated View interface
    – All business logic goes here
      • Responses to events on the view, server calls, etc.

• See example on next slide

```java
public class HelloActivity extends AbstractActivity implements HelloView.Presenter {
    private String name;
    private ClientFactory clientFactory;
    ...
    @Override
    public void start(AcceptsOneWidget container, EventBus eventBus) {
        HelloView view = clientFactory.getHelloView();
        view.setName(name);
        ...
    }
    ...
}
```

Equivalent to the “go” method we had in the Presenter before.
Main Components of GWT MVP Framework (Continued)

• **Place**
  - Encapsulates a navigational state
  - Most Activities have an associated Place
  - Each $Blah$Place extends the Place class
  - Usually has a Tokenizer defined as an inner public class
    - Implements PlaceTokenizer<$Blah$Place>
    - Knows how to take a URL history token and reconstruct a $Blah$Place instance
      - public $Blah$Place getPlace(String token);
    - Knows how to take a $Blah$Place instance and reconstruct a URL history token
      - public String getToken($Blah$Place place);
  - A place doesn’t need a custom Tokenizer if the only URL data it cares about is the name of the place itself

• **Place example:**

```java
public class HelloPlace extends Place {
    private String name;
    ...

    public HelloPlace(String token) { this.name = token; }
    public String getName() { return name; }

    public static class Tokenizer implements PlaceTokenizer<HelloPlace> {
        @Override
        public String getToken(HelloPlace place) {...}

        @Override
        public HelloPlace getPlace(String token) {...}
    }
}```
Main Components of GWT MVP Framework (Continued)

• **PlaceHistoryMapper**
  – A way to let GWT know about your history tokenizers
  – Link between PlaceTokenizers and GWT’s history handling mechanism
    • I.e., Link between PlaceTokenizers and PlaceHistoryHandler

• **Example:**
  ```java
  @WithTokenizers({HelloPlace.Tokenizer.class, ...}
  public interface MyPlaceHistoryMapper extends PlaceHistoryMapper {}
  ```

Main Components of GWT MVP Framework (Continued)

• **ActivityMapper**
  – A way to map each Place to some Activity
    • Remember that Activity implements Presenter interface
      – I.e., it IS a presenter
    – GWT MVP framework uses it to automatically route the app to an Activity

• **Example:**
  ```java
  public class MyActivityMapper implements ActivityMapper {
      ...
      @Override
      public Activity getActivity(Place place) {
          if (place instanceof HelloPlace) {
              return new HelloActivity(...);
          } else if (place instanceof ...) {
              ...
          }
      }
  }
  ```
Summary: Main Components of GWT MVP Framework

• **Model**
  – Responsible for holding raw data (no business logic)

• **View**
  – Responsible for displaying data (extends isWidget)
  – Defines `BlurView` and an associated Presenter interface

• **ClientFactory**
  – Holds on to instances of views and other reusable system resources

• **Activity**
  – Implements view’s Presenter interface and drives all business logic (i.e., it is a presenter)
    • Responds to events on the view, makes server calls, etc.

Summary: Main Components of GWT MVP Framework (Cont.)

• **Place**
  – Encapsulates a navigational state
  – Defines Tokenizer that knows how to make a Place out of history token and make a URL history token out of Place

• **PlaceHistoryMapper**
  – Maps places and their tokenizers in GWT MVP framework’s history handling mechanism

• **ActivityMapper**
  – Maps each Place to some Activity
    • GWT MVP framework uses it to automatically route to the app to that Activity
Some Other GWT MVP Components
(Provided by GWT MVP Framework)

• **PlaceController**
  – Provided by GWT to programmatically go to a Place, etc.

• **ActivityManager**
  – Keeps track of all Activities within the context of one container widget
  – Listens for PlaceChangeRequestEvents and calls the lifecycle methods of Activity to proceed
    • Activity can let the user reject the request by returning a non-null value (a message to the user) in Activity.onMayStop()
Example Setup

- Using Google’s GWT MVP example app
  - Very simple (easier to understand the architecture)

```java
public class HelloMVP implements EntryPoint {
  private Place defaultPlace = new HelloPlace("World!");
  private SimplePanel appWidget = new SimplePanel();

  public void onModuleLoad() {
    ClientFactory clientFactory = GWT.create(ClientFactory.class);
    EventBus eventBus = clientFactory.getEventBus();
    PlaceController placeController = clientFactory.getPlaceController();
    ActivityMapper activityMapper = new AppActivityMapper(clientFactory);
    ActivityManager activityManager = new ActivityManager(activityMapper, eventBus);
    activityManager.setDisplay(appWidget);
```

Create ClientFactory using differed binding.
Start ActivityManager for the main widget with our ActivityMapper.
AppPlaceHistoryMapper historyMapper = GWT.create(AppPlaceHistoryMapper.class);
PlaceHistoryHandler historyHandler =
    new PlaceHistoryHandler(historyMapper);
historyHandler.register(placeController,
                       eventBus,
                       defaultPlace);

RootPanel.get().add(appWidget);
historyHandler.handleCurrentHistory();
}
}

HelloView.java

... public interface HelloView extends IsWidget {
    void setName(String helloName);
    void setPresenter(Presenter listener);

    public interface Presenter {
        void goTo(Place place);
    }

}
HelloViewImpl.java

```java
public class HelloViewImpl extends Composite implements HelloView {
    ...  
    public void setName(String name) {
        this.name = name;
        nameSpan.setInnerText(name);
    }

    @UiHandler("goodbyeLink")
    void onClickGoodbye(ClickEvent e) {
        presenter.goTo(new GoodbyePlace(name));
    }
    
    @Override
    public void setPresenter(Presenter presenter) {
        this.presenter = presenter;
    }
} 
```

HelloPlace.java

```java
public class HelloPlace extends Place {
    private String helloName;
    public HelloPlace(String token) {
        this.helloName = token;
    }

    public String getHelloName() {
        return helloName;
    }

    public static class Tokenizer implements PlaceTokenizer<HelloPlace> {
        public String getToken(HelloPlace place) {
            return place.getHelloName();
        }

        public HelloPlace getPlace(String token) {
            return new HelloPlace(token);
        }
    }
} 
```
AppPlaceHistoryMapper.java

...  
@WithTokenizers( { HelloPlace.Tokenizer.class,  
                 GoodbyePlace.Tokenizer.class })  
public interface AppPlaceHistoryMapper extends PlaceHistoryMapper {  
}  

Let's GWT know about our Places and Tokenizers.

HelloActivity.java

public class HelloActivity extends AbstractActivity  
       implements HelloView.Presenter {  
    private ClientFactory clientFactory;  
    private String name;  
    
    public HelloActivity(HelloPlace place,  
                         ClientFactory clientFactory) {  
        this.name = place.getHelloName();  
        this.clientFactory = clientFactory;  
    }  
    
    public void start(AcceptsOneWidget containerWidget,  
                      EventBus eventBus) {  
        HelloView helloView = clientFactory.getHelloView();  
        helloView.setName(name);  
        helloView.setPresenter(this);  
        containerWidget.setWidget(helloView.asWidget());  
    }  

Contains views, eventBus, placeController.  
Retrieve navigational state so the view can be recreated with it.  
ActivityManager calls start to start this activity's processing.
HelloActivity.java (continued)

```java
public String mayStop() {
    return "Please hold on. This activity is stopping."
;
}

public void goTo(Place place) {
    clientFactory.getPlaceController().goTo(place);
}
```

Asks user before stopping activity

Method from HelloView interface.

AppActivityMapper.java

```java
public class AppActivityMapper implements ActivityMapper {
    private ClientFactory clientFactory;

    public AppActivityMapper(ClientFactory clientFactory) {
        this.clientFactory = clientFactory;
    }

    public Activity getActivity(Place place) {
        if (place instanceof HelloPlace) {
            return new HelloActivity((HelloPlace) place,
                clientFactory);
        } else if (place instanceof GoodbyePlace) {
            return new GoodbyeActivity((GoodbyePlace) place,
                clientFactory);
        }
        return null;
    }
}
```

If Place can't be mapped to an Activity, stays on the same Activity as before, i.e., does nothing.
public class GoodbyeActivity extends AbstractActivity {
    private ClientFactory clientFactory;
    private String name;

    public GoodbyeActivity(GoodbyePlace place,
                            ClientFactory clientFactory) {
        this.name = place.getGoodbyeName();
        this.clientFactory = clientFactory;
    }

    public void start(AcceptsOneWidget containerWidget,
                      EventBus eventBus) {
        GoodbyeView goodbyeView = clientFactory.getGoodbyeView();
        goodbyeView.setName(name);
        containerWidget.setWidget(goodbyeView.asWidget());
    }
}
Deferred Binding

Deferred Binding For ClientFactory

- Very similar to IoC (Inversion of Control)
- When you call GWT.create(ClientFactory.class), GWT looks up the implementing class in its app.gwt.xml
- Example:
  
  ```xml
  <!-- Use ClientFactoryImpl by default -->
  <replace-with class="com.hellomvp.client.ClientFactoryImpl">
    <when-type-is class="com.hellomvp.client.ClientFactory"/>
  </replace-with>
  ```

- Can use <when-property-is>, specifying user agent (type of browser), i.e. mobile, desktop, etc.
  - Allows to give different view impl for mobile vs desktop
Wrap-Up

Summary

- REALLY big learning curve, but very flexible and customizable
  - Still worth it for large scale apps
- Views define their interface as well as Activities interface (its presenter)
- Places handle navigational state
  - Place-defined Tokenizer allows conversion from token to Place instance and back
- Use ClientFactory approach to cache reusable resources in your app
  - Using appName.gwt.xml, use deferred binding to create different views for different browsers, devices, etc.
Questions?

Customized Java EE Training: http://courses.coreservlets.com/
GWT, Java 7 and 8, JSF 2, PrimeFaces, HTML5, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, REST, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.