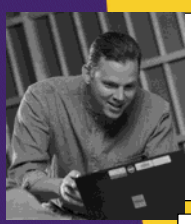
The Google Web Toolkit (GWT): Programming Basics

(GWT 2.5 Version)

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

© 2013 Marty Hall & Yaakov Chaikin

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

- **Development process**
  - Making a project
    - File → New → Web Application Project
  - Deleting extraneous files
    - Most Java classes other than AppName.java
  - Editing auto-generated HTML file
    - Preserve link to load JavaScript file
    - Create HTML elements with ID's for GWT to use
  - Editing auto-generated Java class
    - Add client-side event handlers
    - Create and call custom Java classes
- **Testing process**
  - Development mode, production mode, deployed mode

GWT Development Process

Review of Last Section

- **Installation and setup**
  - Install GWT Eclipse plugin
    - Select option to download GWT at the same time as the plugin
  - Install GWT browser plugin
    - Done first time you run in development mode
  - Bookmark GWT docs
- **Making new project**
  - File → New → Web Application Project
    - Or, click "g" at top 
- **Testing**
  - Development mode (R-click, Run As → Web Application)
  - Production mode (R-click, Google → GWT Compile)
  - Deployment mode on regular Java server (copy/rename "war" folder and deploy)

General Application Development Approach

- **Create new project**
- **Delete extraneous files**
  - Delete auto-generated classes that you are not using
  - If simple app with no server-side code:
    - Delete entire "server" package
    - Edit web.xml to remove reference to GWT servlet
- **Edit auto-generated HTML file**
  - war/MyApp.html
- **Edit auto-generated Java file**
  - src/package.client/MyApp.java
- **Test**

Editing Auto-Generated HTML File

- **File location**
  - war/AppName.html
- **Required script**
  - GWT builds <script…> tag in head section. Leave this unchanged.
  - Entire rest of sample HTML file can be deleted and replaced with custom content.
- **Body content**
  - Give ids to sections where buttons, textfields, and output will go. Usually div, span, td, or th elements.
    -
  - We will see in later section that you can reserve just a *single* region, then build up panels and widgets from Java

Auto-Generated HTML File: Typical Structure

```
<!doctype html>
<html>
<head><title>…</title>
…
<script type="text/javascript" language="javascript"
        src="appname/appname.nocache.js"></script>
</head>
<body>
…
<div id="placeholder-for-Java-to-refer-to"></div>
…
</body></html>
```

Editing Auto-Generated Java Class

- **File location**
  - src/package.client/AppName.java
- **Required structure**
  - Class must implement EntryPoint and contain onModuleLoad method. Entire rest of sample Java class can be deleted and replaced with custom content.
- **Usual code**
  - Create a control
    - Button button1 = new Button("Press Here");
  - Give it an event handler
    - button1.addClickHandler(new Button1Handler());
  - Insert the control into the page
    - RootPanel.get("placeholderForButton1").add(button1);

Auto-Generated Java Class: Minimal Structure

```
package packageName.client;

import com.google.gwt.core.client.*;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.event.dom.client.*;

public class AppName implements EntryPoint {
  @Override
  public void onModuleLoad() {  // Main entry point

  }
}
```

Editing war/WEB-INF/web.xml

```
<?xml version="1.0" ...><web-app ...>
  <!-- Servlets -->
  <servlet>
    <servlet-name>greetServlet</servlet-name>
    <servlet-class>blah.server.GreetingServiceImpl</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>greetServlet</servlet-name>
    <url-pattern>/mainappname/greet</url-pattern>
  </servlet-mapping>

  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>MainAppName.html</welcome-file>
  </welcome-file-list>

  <!-- More declarations that you can leave intact ... -->
  ...
</web-app>
```

Delete all this. Leave everything else in file as is.

Depending on the order in which you clean up the code, Eclipse may automatically remove these entries. If you first delete the code inside of the main EntryPoint class, Eclipse is usually smart enough to remove the servlet and servlet-mapping entries automatically.

Example: Button that Generates Random Number (on Client)

Big Ideas

- **Overall development process**
  - Delete extraneous code
  - Edit auto-generated HTML file
  - Edit auto-generated Java class
- **HTML**
  - <script> tag
  - Placeholders where Java will insert widgets
- **Java**
  - General class structure
    - Extending EntryPoint
    - Starting point: onModuleLoad
  - Client-side event handlers

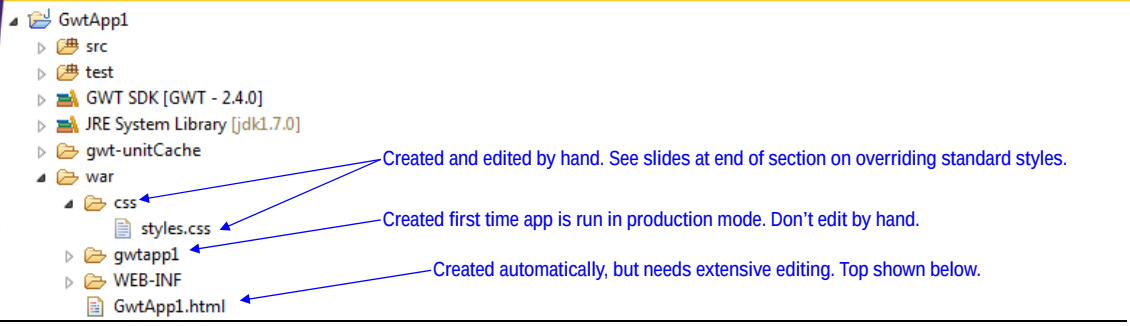
Overview of Process

- **Create GwtApp1 project**
  - Project name: GwtApp1
  - Package: coreservlets
- **Delete extraneous files**
  - RPC code, server-side code, CSS file, web.xml entries
    - Client code only; later tutorial section discusses RPC
- **Edit HTML**
  - war/GwtApp1.html
    - Need regions whose ids (not names!) are numberButtonHolder and numberResultHolder
- **Edit Java**
  - src/coreservlets.client/GwtApp1.java
    - Button named randomNumberButton
    - HTML (label) named randomNumberResult
    - Button's event handler should insert value into HTML

Deleting Extraneous Files

- **Motivation**
  - New apps contain sample files that are good examples to learn from, but that should be deleted from new projects
- **Files to delete**
  - Under src/package.client
    - GreetingService.java, GreetingServiceAsync.java
  - Under src/package.server
    - GreetingServiceImpl.java
  - Under src/package.shared
    - FieldVerifier.java
  - Under war
    - AppName.css (replace with own style sheet in subfolder)
- **Files to edit**
  - war/WEB-INF/web.xml
    - Remove servlet/servlet-mapping for GreetingServiceImpl

Auto-Generated HTML File




```
<!doctype html>
<html>
<head><title>First GWT App</title>
<link rel="stylesheet"
      href="./css/styles.css"
      type="text/css"/>
<script type="text/javascript" language="javascript"
        src="gwtapp1/gwtapp1.nocache.js"></script>
</head>
<body>
```

This code edited by hand.
Use normal HTML (often xhtml).

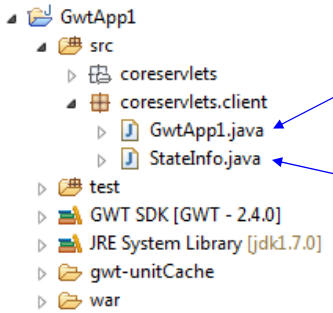
Inserted automatically by GWT. Leave it in unchanged. *Entire* rest of file can be modified in any way. There is an iframe in the body that is useful if you track history (see later tutorial section), but you can delete it for now.

Auto-Generated HTML File: Continued

```
...
<fieldset>
<legend>Client-Side Data</legend>
<table border="1">
  <tr><th>User Control</th>
      <th>Result</th></tr>
  <tr><td id="numberButtonHolder"></td>
      <td id="numberResultHolder"></td></tr>
  ...
</table><br/>
</fieldset>
...
</body>
</html>
```

These names will be referred to in Java code. In these simple examples we create one HTML section for each low-level GWT widget. But in more advanced applications, it is common to put GWT panels into large HTML sections, then put multiple GWT widgets in the panels. It is even possible to build a GUI completely with GWT, where the HTML just has one blank div section that will contain the main GWT panel.

Auto-Generated Java Class



Created automatically, but need extensive editing. Shown below.

Created by hand. Will be used later in this tutorial.

```
package coreservlets.client;
import com.google.gwt.core.client.*;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.event.dom.client.*;

public class GwtApp1 implements EntryPoint {
  public void onModuleLoad() {
    …
  }
}
```

Created automatically. Your code should be called by onModuleLoad.

Auto-Generated Java Class: Continued

```
public class GwtApp1 implements EntryPoint {
  public void onModuleLoad() {  // Main entry point
    buttonSetup();
    listSetup();
  }

  private void buttonSetup() {
    Button randomNumberButton = new Button("Show Random Number");
    HTML randomNumberResult = new HTML("<i>Num will go here</i>");
    randomNumberButton.addClickHandler
      (new RanNumHandler(randomNumberResult));
    RootPanel.get("numberButtonHolder").add(randomNumberButton);
    RootPanel.get("numberResultHolder").add(randomNumberResult);
  }
  ...
}
```

Button is builtin GWT class. All standard HTML controls have corresponding GWT Swing-like classes. GWT also defines many extended components that are combinations of HTML elements.

Matches ids in the HTML

Auto-Generated Java Class: Continued

Using an inner class for event handling. Alternative approaches discussed in upcoming section on GWT events.

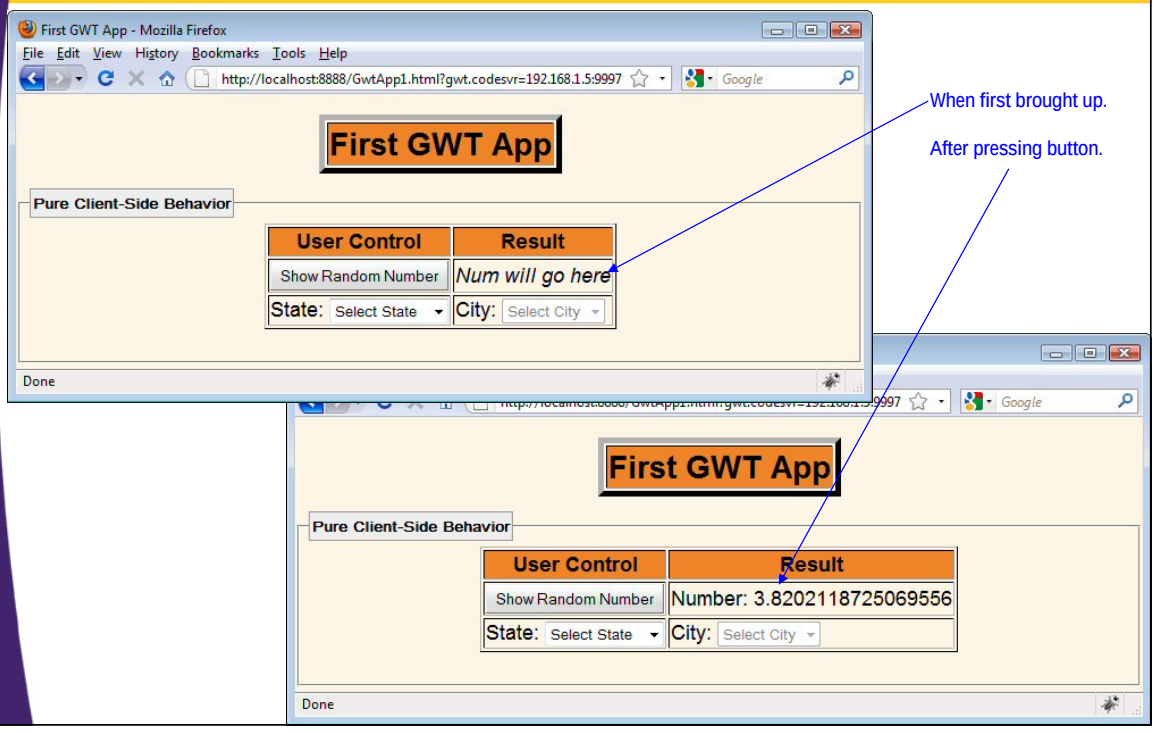
```
private class RanNumHandler implements ClickHandler {
   private HTML resultRegion;

   public RanNumHandler(HTML resultRegion) {
     this.resultRegion = resultRegion;
   }

   public void onClick(ClickEvent event) {
     resultRegion.setText("Number: " + Math.random()*10);
   }
 }
```

Whenever user clicks button, insert this string into the GWT **HTML** object (which is really a div).

Testing in Development Mode

- **Idea**
  - Running in development mode means running in a browser that uses Google browser plugin to link Java code to Eclipse
  - When you change Java or HTML in Eclipse, you do *not* need to restart development mode. Just go to browser and hit Reload.
- **Steps**
  - R-click project name and Run As → Web Application
  - Result:
    - Shows URL in "Development Mode" tab
    - You should paste this URL into a browser (R-click it for options)
      - This discussion assumes you already installed browser plugin as described in previous tutorial section
      - If you ran previous app in development mode, stop Jetty before running this new app. Go to "Development Mode" tab and click Red stop button.

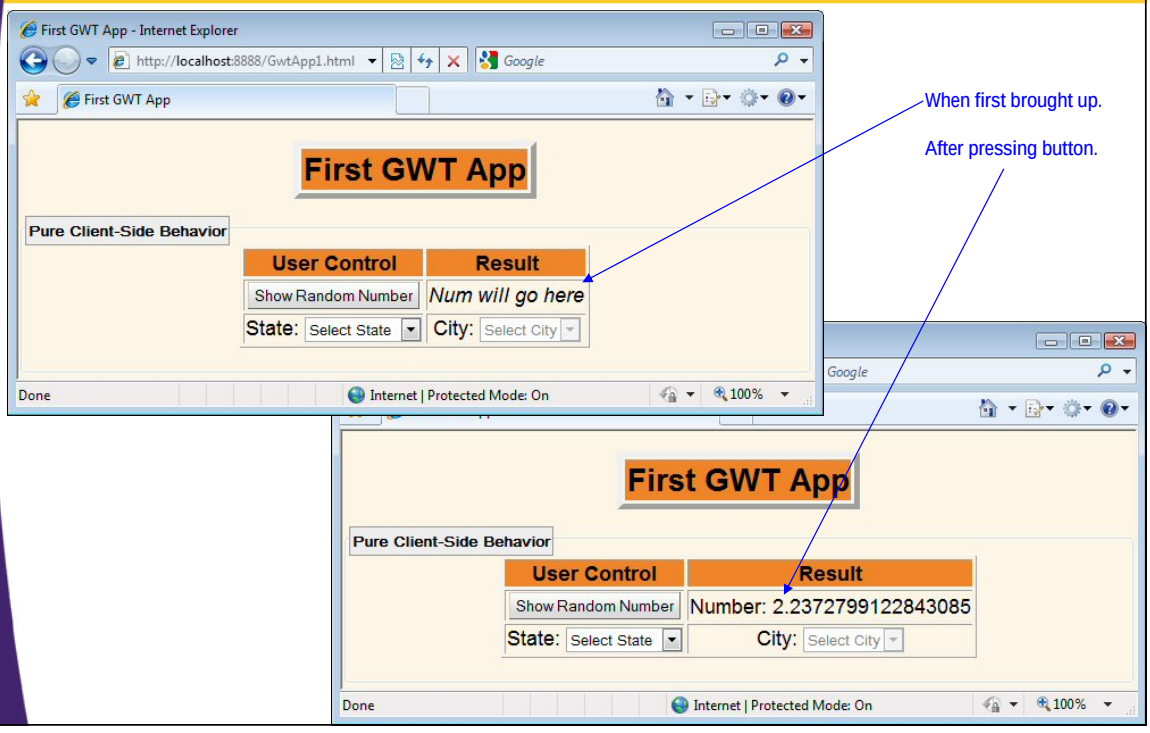
Testing in Development Mode



Testing in Production Mode

- **Idea**
  - Running in production mode means running in a regular browser with all client-side Java code converted to JavaScript
    - Only do this occasionally. Changes to code requires you to repeat production mode steps.
  - Uses embedded version of Jetty server on local machine
- **Steps**
  - Run the application in development mode
  - R-click app, Google → GWT Compile
  - Change URL to http://localhost:8888/AppName.html
    - If you have no server-side code, you can also drag war/AppName.html into a browser
  - Result:
    - Creates JavaScript files in war/appname folder

Testing in Production Mode

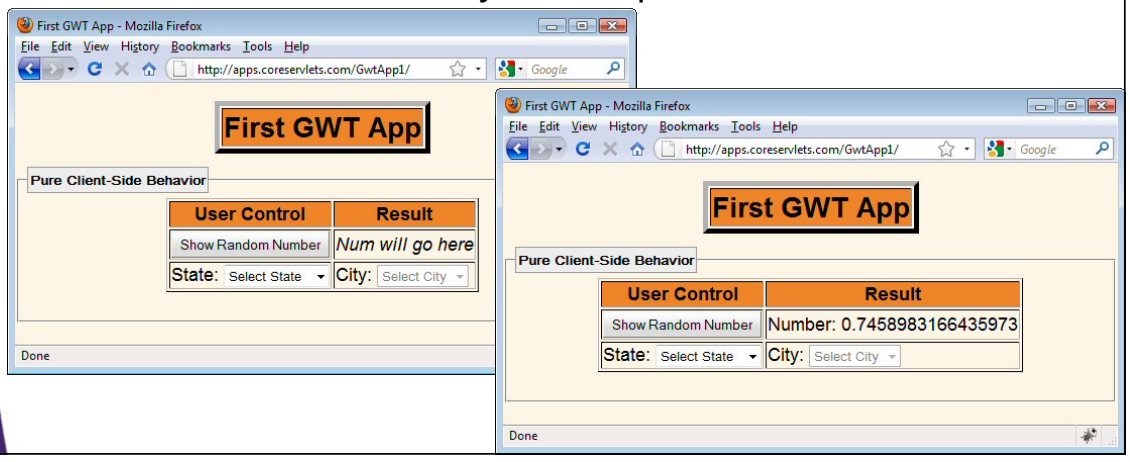


Testing in Deployed Mode

- **Idea**
  - Deployed mode is similar to production mode
    - Running in a regular browser with all client-side Java code converted to JavaScript
    - However, uses normal Web App structure running on a Java server of your choice on any machine
- **Steps**
  - Run the application in production mode
  - R-click project, Google → GWT Compile
    - Do this *immediately* before deploying so you have JavaScript files corresponding to most recent Java code
  - Find the "war" folder on file system
    - Copy/rename and deploy as exploded WAR
    - Or, copy contents (not "war" folder itself) into file.zip, then rename file.zip to file.war and deploy as WAR file

Testing in Deployed Mode

- **Copied workspace/GwtApp1/war folder**
  - Renamed to GwtApp1 to keep similar-looking URL
    - GwtApp1.html is welcome-file, so name can be omitted
  - Deployed to Apache Tomcat on apps.coreservlets.com
    - Could have used any Java-capable server



Example: Linked Comboboxes

Big Ideas

- **Use regular Java for client-side code**
  - Normal Java 1.6 syntax
    - Classes, methods, constructors, loops, control statements, etc.
  - Supporting classes
    - Static methods in the Math class and many common data structures
      - String, array, List (ArrayList, LinkedList), Map (HashMap and others), Date, etc.
      - For full list, see http://code.google.com/webtoolkit/doc/latest/RefJreEmulation.html
  - Custom GWT classes
    - Class defined for each type of standard HTML control
      - Button, Checkbox, Image, Hyperlink, RadioButton, etc.
    - GWT also provides classes for groups of HTML controls
      - SuggestBox, SimplePanel, TabPanel, ScrollPanel, Tree, FlexTable, PopupPanel, etc.
- **Java gets translated into JavaScript**
  - Google → GWT Compile to generate JavaScript
  - But you develop and test using Java *only*
- **Restrictions**
  - Not all collections or classes supported
    - But relatively clear warning message when you try to use unsupported feature
  - Custom classes used by client code must be placed in "...client" package.

Example: Linked Comboboxes
(Choosing State Results in List of Associated Cities)

- **Eclipse Project: GwtApp1**
  - Same project as in previous example
- **HTML: GwtApp1.html**
  - Need regions with ids stateListHolder and cityListHolder
- **Java:**
  - GwtApp1.java: main application class
    - Defines two listboxes
    - Attaches event handler to listbox that stores state names
  - StateInfo.java (must be in ...client package)
    - Associates state name with array of cities
    - Defines static method with list of common states
- **Note**
  - In AjaxTags tutorial, we matched cities to states using server-side code. Using custom Java classes greatly simplifies client-side code.

Editing Auto-Generated Files

- **Edit GwtApp1.html and give ids to sections**
  - Give ids to sections where buttons, textfields, and output will go

```
<tr><td>State: <span id="stateListHolder"></span></td>
    <td>City: <span id="cityListHolder"></span></td></tr>
```

- **Edit GwtApp1.java**
  - Code called by auto-generated onModuleLoad method

1. Create controls

```
stateList = new ListBox();
cityList = new ListBox();
```

2. Define event handlers

```
stateList.addChangeHandler(new StateHandler());
```

3. Insert in HTML page

```
RootPanel.get("stateListHolder").add(stateList);
RootPanel.get("cityListHolder").add(cityList);
```

Auto-Generated HTML File

- **Same file as in earlier example**
  - war/GwtAppl.html
  - Needs auto-generated script tag in the head section
- **Body**

```
<fieldset>
<legend>Client-Side Data</legend>
<table border="1">
  <tr><th>User Control</th>
      <th>Result</th></tr>
  <tr><td id="randomNumberButton"></td>
      <td id="randomNumberResult"></td></tr>
  <tr><td>State: <span id="stateListHolder"></span></td>
      <td>City:  <span id="cityListHolder"></span></td></tr>
</table><br/>
</fieldset>
```


Auto-Generated Java Class

- **Same file as in earlier example**
  - Called src/coreservlets.client.GwtApp1.java
  - Has three sets of import statements
  - Has auto-generated features
    - implements EntryPoint
    - onModuleLoad method

```
package coreservlets.client;

import com.google.gwt.core.client.*;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.event.dom.client.*;

public class GwtApp1 implements EntryPoint {
  public void onModuleLoad() {  // Main entry point
```

Auto-Generated Java Class: Continued

```
public class GwtApp1 implements EntryPoint {
  public void onModuleLoad() {  // Main entry point
    buttonSetup();
    listSetup();
  }
  // buttonSetup shown earlier

  private void listSetup() {
    ListBox stateList = new ListBox();
    populateStateList(stateList);
    stateList.setVisibleItemCount(1);
    ListBox cityList = new ListBox();
    cityList.addItem("Select City");
    cityList.setVisibleItemCount(1);
    cityList.setEnabled(false);
    stateList.addChangeHandler(new StateHandler(stateList,
                                                cityList));
    RootPanel.get("stateListHolder").add(stateList);
    RootPanel.get("cityListHolder").add(cityList);
  }
```

ListBox is builtin GWT class. All standard HTML controls have corresponding GWT Swing-like classes. GWT also defines many extended components that are combinations of HTML elements.

Matches ids in the HTML

Auto-Generated Java Class: Continued

```
private void populateStateList(ListBox stateList) {
   stateList.addItem("Select State");
   StateInfo[] nearbyStates =
     StateInfo.getNearbyStates();
   for(StateInfo state: nearbyStates) {
     stateList.addItem(state.getStateName());
   }
 }
```

Unlike early GWT versions, you can use Java 5 and 6 features (e.g., new-style for/each loop) for client-side code. However, you are limited in what classes are supported for client-side code. For a complete list, see http://code.google.com/webtoolkit/doc/latest/RefJreEmulation.html. For server-side code (see later tutorial section), you can use any features of the Java version your server is using.

Auto-Generated Java Class: Continued

```
private class StateHandler implements ChangeHandler {
   private ListBox stateList, cityList;

   public StateHandler(ListBox stateList, ListBox cityList) {
     this.stateList = stateList;
     this.cityList = cityList;
   }

   public void onChange(ChangeEvent event) {
     int index = stateList.getSelectedIndex();
     String state = stateList.getItemText(index);
     StateInfo[] nearbyStates =
       StateInfo.getNearbyStates();
     String[] cities =
       StateInfo.findCities(nearbyStates, state);
     cityList.clear();
     for(String city: cities) {
       cityList.addItem(city);
     }
     cityList.setEnabled(true);
   }
 }
```


Helper Class (StateInfo)

```
package coreservlets.client;

public class StateInfo {
  private String stateName;
  private String[] cities;

  public StateInfo(String stateName, String[] cities) {
    this.stateName = stateName;
    this.cities = cities;
  }

  public String getStateName() {
    return(stateName);
  }

  public String[] getCities() {
    return(cities);
  }
```

Must be in same package as AppName.java. (...client). Only classes in this package (or a subpackage) are translated into JavaScript.

Helper Class (StateInfo, Continued)

```
private static String[] mdCities =
  {"Baltimore", "Frederick","Gaithersburg","Rockville"};
private static String[] vaCities =
  {"Virginia Beach","Norfolk","Chesapeake","Arlington"};
private static String[] paCities =
  {"Philadelphia","Pittsburgh","Allentown","Erie"};
private static String[] njCities =
  {"Newark", "Jersey City","Paterson","Elizabeth"};
private static String[] nyCities =
  {"New York", "Buffalo","Rochester","Yonkers"};
private static StateInfo[] nearbyStates =
  { new StateInfo("Maryland", mdCities),
    new StateInfo("Virginia", vaCities),
    new StateInfo("Pennsylvania", paCities),
    new StateInfo("New Jersey", njCities),
    new StateInfo("New York", nyCities)
  };
```

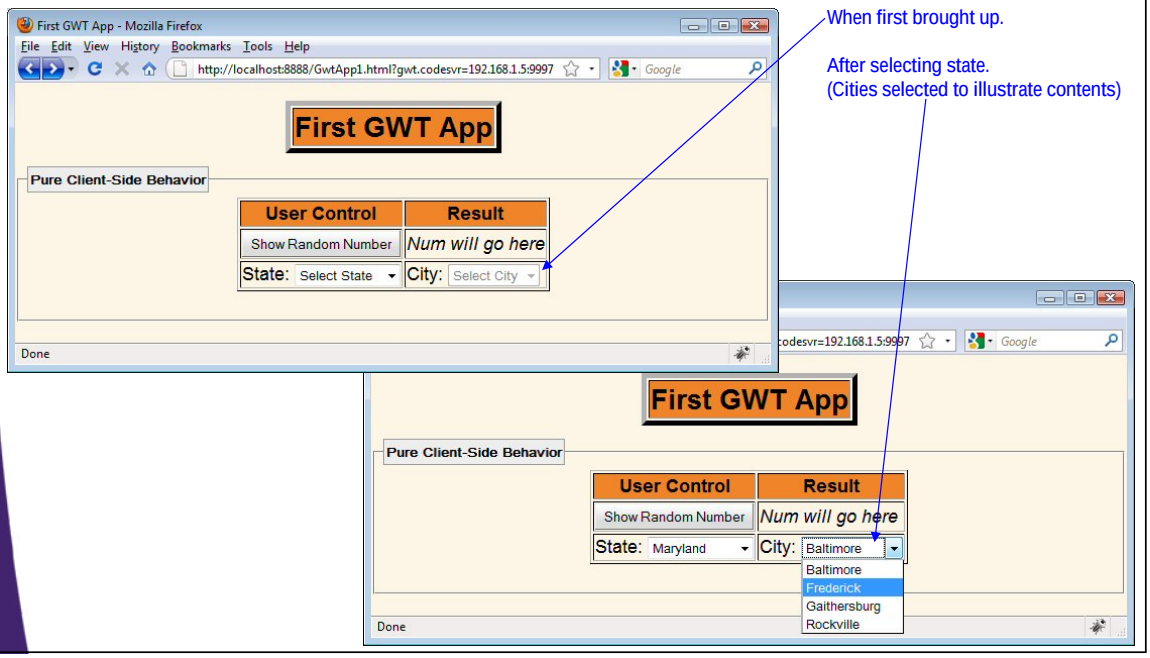
Can use arrays, ArrayList, HashMap, String, Date, custom classes, etc. GWT does not support every standard Java class on client, but a large set of them. For full list, see http://code.google.com/webtoolkit/doc/latest/RefJreEmulation.html

Helper Class (StateInfo, Continued)

```
public static StateInfo[] getNearbyStates() {
  return(nearbyStates);
}

public static String[] findCities(StateInfo[] states,
                                  String stateName) {
  for(StateInfo state: states) {
    if(state.getStateName().equals(stateName)) {
      return(state.getCities());
    }
  }
  String[] unknown = {"Unknown state"};
  return(unknown);
}
}
```

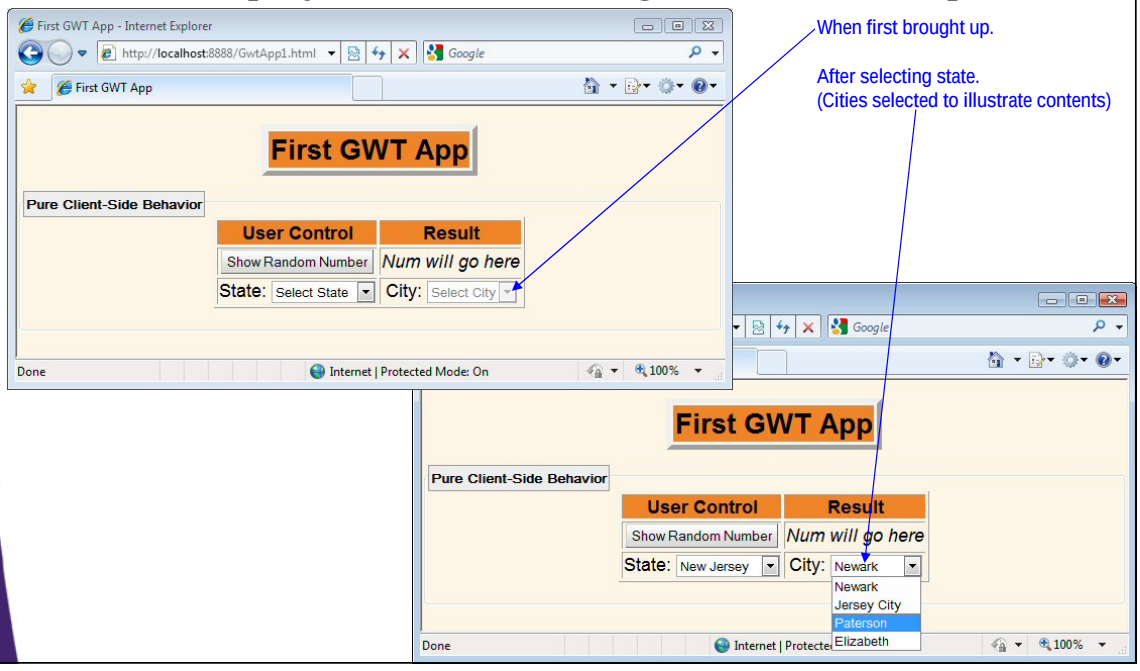
Testing in Development Mode

– R-click project name and Run As → Web Application



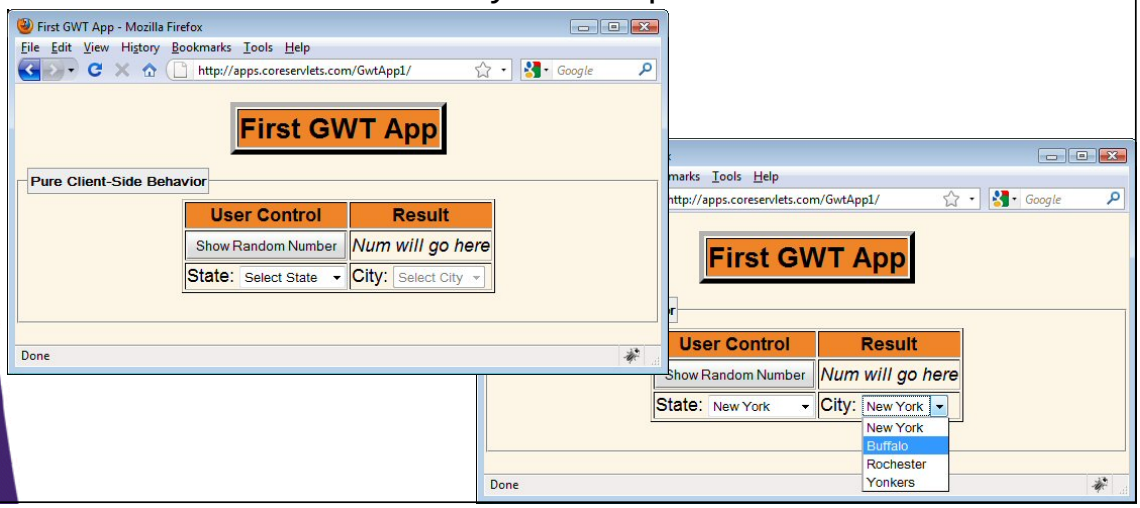
Testing in Production Mode

– R-click project name and Google → GWT Compile



Testing in Deployed Mode

- **workspace/GwtApp1/war**
  - Renamed to GwtApp1 to keep similar-looking URL
  - Deployed to Apache Tomcat on apps.coreservlets.com
    - Could have used any Java-capable server



Overriding Standard CSS Styles

Auto-Loaded Style Sheet in Auto-Generated HTML File

- **Idea**
  - The JavaScript file automatically inserts <link rel="stylesheet" href="…/gwt/clean/clean.css"> at *end* of head section.
- **Good news**
  - There are styles such as gwt-Button and gwt-HTML that let you customize the look of all GWT widgets
  - There are four predefined themes (clean, standard, dark, chrome) that you can switch among.
    - Edit src/mainPackage/AppName.gwt.xml to change
- **Bad news**
  - The automatic stylesheet is loaded after yours. So, it overrides your entries, not vice versa.
  - It gives styles for body, td, select, pre, and a.
  - Cannot edit it: gets rebuilt when you do GWT Compile

Simple Usage: Overriding Standard Styles

- **Mark your styles with !important**
  - End your entries with !important so that they won't be overridden by clean.css.
  - Only needed for body, td, select, pre, and a.
    - body { background-color: red !important; … }
      td { font-size: 103px !important; … }
- **Or, use explicit CSS class names**
  - Don't give styles to those five HTML element names, but instead use CSS class names
    - CSS:
      .mybodystyle { background-color: red; … }
      .bigcell { font-size: 103px; …
    - HTML:
      <body class="mybodystyle">
      <td class="bigcell">

Advanced Usage: Loading Stylesheets Manually

- **Edit src/packageName/AppName.gwt.xml**
  - Change <inherits name='… .Clean'/> to <inherits name='… .CleanResources'/>
    - This causes stylesheet to be output under war folder, but not loaded
- **Load the GWT stylesheet first**
  - <link rel="stylesheet" href="./appname/gwt/clean/clean.css">
- **Load your stylesheet second**
  - <link rel="stylesheet" href="./css/mystyles.css">

styles.css from GwtApp1

```
body { background-color: #fdf5e6 !important }
td { font-size: 18px !important;
     font-family: Arial, Helvetica, sans-serif !important;
}
…
th { background-color: #ef8429;
     font-size: 18px;
     font-family: Arial, Helvetica, sans-serif;
}
legend {
  font-weight: bold;
  color: black;
  background-color: #eeeeee;
  border: 1px solid #999999;
  padding: 3px 2px;
}
```

Entries for body and td marked as !important because body and td defined in standard.css. This approach is recommended for GWT beginners.

Entries for other elements not marked with !important because they are not defined in standard.css

Advanced Usage: Loading Stylesheets Manually

- **src/packageName/GwtApp1.gwt.xml**
  - Change
    - <inherits name='com.google.gwt.user.theme.clean.Clean'/>
  - to
    - <inherits name='com.google.gwt.user.theme.clean.CleanResources'/>
- **Auto-generated HTML file**
  - Load clean.css and custom stylesheet. If you load your custom stylesheet second, its entries override earlier ones.
    - <link rel="stylesheet"
      href="./gwtapp1/gwt/clean/clean.css">
      <link rel="stylesheet"
      href="./css/mystyles.css">

Wrap-Up

Summary

- **Make a project**
  - File → New → Web Application Project
- **Delete extraneous files**
  - Most Java classes other than AppName.java
  - If no server code (e.g., when first practicing), remove "server" package and edit web.xml to remove servlet refs
- **Edit auto-generated HTML file**
  - Preserve link to load JavaScript file
  - Create HTML elements with ID's for GWT to use
- **Edit auto-generated Java class**
  - Add client-side event handlers
  - Create and call custom Java classes

Questions?