More powerful and standards-compliant web browsers have allowed the development of so-called “Rich Internet Applications” (RIAs) that provide the same user experience as desktop applications. Google Web Toolkit hides many of the complexities of RIA development behind a Java programming model that is akin to developing a desktop application.

Google Web Toolkit

- Rich Internet Applications
- Google Web Toolkit
- Widgets, containers, and event handling
Rich Internet Applications

Traditionally, there have been significant differences between applications delivered on the desktop and those delivered on the web.

<table>
<thead>
<tr>
<th>Desktop</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensive widget set</td>
<td>HTML forms (button, text fields, etc.)</td>
</tr>
<tr>
<td>Operates on local machine</td>
<td>Accesses data on server</td>
</tr>
<tr>
<td>Difficult to distribute updates</td>
<td>Always the latest version</td>
</tr>
<tr>
<td>Runs as fast as the machine’s processor</td>
<td>Possibly slow network</td>
</tr>
<tr>
<td>Trusted software that can access local disk</td>
<td>“Someone else’s” software</td>
</tr>
</tbody>
</table>

Now, web browsers are sophisticated enough (and networks fast enough) to provide an user experience that approaches the desktop experience.
Rich Internet Applications

RIAs leverage a number of web technologies

- **HTML** represent the visual components of an application
- **CSS** (Cascading Style Sheets) provide coloring and positioning
- Web browsers model pages using **DOM**, the Document Object Model
- **JavaScript** is used to implement application logic
- A special object in the browser, called the **XMLHttpRequest** object (XHR), that lets a page communicate with the server without reloading
  - AJAX (Asynchronous JavaScript and XML) requests XML data form the server which is rendered in the page
Rich Internet Applications

There were a number of reasons that it took so long for RIAs to appear

- Browsers offered different levels of support for JavaScript and CSS
  - Had to be knowledgeable about and test on every browser
  - Lots of folklore (not well-documented)

- JavaScript wasn’t viewed as a real programming language
  - Lots of bad examples out there
  - Different enough from mainstream languages (C, Java, Visual Basic) that server-side programmers shied away
  - Support for the language was spotty

- Applications on the web were aimed at the lowest common denominator
  - Old browsers, slow connections, naive users
Rich Internet Applications

Meanwhile, things were happening...

- JavaScript libraries were developed to provide widgets and APIs that could be used across all browsers
  - Prototype, script.aculo.us, Dojo, YUI, jQuery, etc.
  - Showed developers what JavaScript can do
- Browsers become more standards compliant
- Web applications branched out from e-commerce
  - “Web 2.0” social networking applications have a different (more tech-savvy) user base
  - Google applications (Maps, World, Docs, etc.) raised the bar

But it still took a lot of technologies and knowledge to develop RIAs
Google Web Toolkit

Google Web Toolkit (GWT) allows RIAs to be developed using only Java

http://code.google.com/webtoolkit

- Provides a suite of Java classes that model UI components
- Development resembles Desktop UI development (Java Swing)
  - Create and configure widget objects (buttons, labels, tables, etc.)
  - Lay out the widgets in a container
  - Handle events generated by the user
- GWT compiles the Java into HTML and JavaScript that is downloaded on the client
  - Familiar to Java programmers
  - Can leverage Java tools (IDEs, debuggers, code analyzers, etc.)
  - JavaScript can be highly optimized by GWT compiler

*And some CSS for styling
Java code translated for the client

Some portion of a GWT application will be compiled into HTML and JavaScript for the client

- User Interface layout, event handling, and display logic

- By convention, code for the client side resides in a client package:
  - edu.pdx.cs410J.gwt.client

- Uses code from GWT “client” packages and a subset of standard Java classes (*JRE Emulation*)
  - java.lang: “wrapper” classes, Math, String, StringBuilder
  - java.util: Date, ArrayList, HashMap, Comparator
Java code translated for the client

Some Java concepts cannot be accurately represent in JavaScript

- Numbers in JavaScript are always 64-bit floating point (affects math operations, overflow, integer division, etc.)
- Cannot get stack traces for exceptions
- Threading-related operations (synchronized, wait/notify) are ignored
- Regular expression support is very similar, be sure to use the overlapping subset

GWT offers alternatives to some Java APIs that would be too slow if implemented in JavaScript

- DateTimeFormat, NumberFormat, Timer

Classes that implement java.io.Serializable (or GWT’s IsSerializable interface) will also be translated to JavaScript
GWT Project Layout

GWT recommends the following directory and package conventions:

```
+- com/
  `-- mycompany/
    `-- myapp/  (root package)
      `-- MyApp.gwt.xml  (module conf file)
  `-- client/  (Client-side source code)
    `-- MyApp.java  (entry point for app)
    `-- DataService.java  (RPC client class)
  `-- server/  (Server-side source code)
    `-- DataServerImpl.java  (RPC server impl)
  `-- public/  (Static content)
    `-- MyApp.html  (HTML file for app)
    `-- MyApp.css  (Style sheet)
    `-- MyLogo.png  (Image file)
```
GWT Modules

A module specifies configuration for a GWT application or library

- Configuration resides in a XML file with the .gwt.xml suffix
  - Module name is kind of a like a class name: com.mycompany.myapp.MyApp defined in MyApp.gwt.xml

- Modules may inherit from other modules
  - Most often a GWT-provided mode like com.google.gwt.user.User that lets your app use the widgets in the com.google.gwt.user.client package

- Renaming a module ensures that the URLs look nice

- A GWT application begins at an entry point class (kind like a main class)

A basic module file might look like this:

```xml
<module rename-to="examples">
  <inherits name="com.google.gwt.user.User"/>
  <entry-point class="edu.--.gwt.client.Examples"/>
  <entry-point class="edu.--.gwt.client.Hello"/>
</module>
```
GWT HTML File

A GWT application is rendered in the contents of a HTML host page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" 
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Hello World in GWT</title>
</head>
<body>
  <script language="javascript"
    src="examples.nocache.js"/>
</body>
</html>
```

The body of the HTML file is available to the GWT code as the RootPanel

- Individual elements of the host page can be obtained via their id using RootPanel.get(String id)
A simple GWT Application

This GWT application displays a button

```java
package edu.pdx.cs410J.gwt.hello.client;

import com.google.gwt.core.client EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.RootPanel;

public class Hello implements EntryPoint {
    public void onModuleLoad() {
        Button button = new Button("Hello World");
        RootPanel.get().add(button);
    }
}
```

Application execution begins with the `onModuleLoad` method of an `EntryPoint` instance

- A module may have multiple entry points
GWT Execution Environment

The are two execution environments for GWT applications: *hosted mode* and *web mode*

Hosted mode is used during development

- The application runs inside a Java program that simulates the browser
- Client code is not translated into JavaScript
- Can attach with a debugger
- “Refreshing” the application will pick up any client-side changes
  - Quickly see how changes effect the user interface
- Error messages are written to a log window
  - App can write to this log using `GWT.log(String, Exception)`
- Uses an embedded Jetty web container instance that serves static content and runs the server-side application components
  - You can configure hosted mode to run against a different server
GWT Web Mode

From hosted mode, client-side Java code can be translated into JavaScript and deployed in web mode

- Launches the application in a real web browser
- Same as actually deploying the application
- Allows you to verify functionality, look, and performance on multiple browsers
GWT with Maven

The GWT tools available from Google work well, but there is a Maven plugin* that makes things easier

http://mojo.codehaus.org/gwt-maven-plugin

The plugin downloads GWT, compiles your app, launches hosted mode, and assembles a war file.

Both the GWT libraries and the codehaus plugin are found in well-known repositories that are automatically searched by Maven

It’s probably a good idea to define your version of GWT once in your pom.xml

<properties>
   <gwtVersion>2.4.0</gwtVersion>
</properties>

*The codehaus plugin merged with the far-superior “totsp” plugin in version 1.1.
Creating a new GWT project

The GWT plugin has an archetype for creating GWT projects

$ mvn archetype:generate \
-DarchetypeGroupId=org.codehaus.mojo \
-DarchetypeArtifactId=gwt-maven-plugin \
-DarchetypeVersion=1.1 \
-DgroupId=edu.pdx.cs410J.whitlock \
-DartifactId=gwtProject

+- gwtProject/
  +- pom.xml
  +- src/
    +- main/
      +- java/
        +- edu/pdx/cs410J/whitlock/
          +- Application.gwt.xml
          +- client
            +- Application.java
        +- resources/
          +- edu/pdx/cs410J/whitlock/
            +- public
              +- Application.css
              +- Application.html
        +- webapp/
          +- index.html
          +- WEB-INF/web.xml
GWT with Maven

Make the project depend on GWT:

```xml
<dependency>
  <groupId>com.google.gwt</groupId>
  <artifactId>gwt-servlet</artifactId>
  <version>${gwt.version}</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.google.gwt</groupId>
  <artifactId>gwt-user</artifactId>
  <version>${gwt.version}</version>
  <scope>provided</scope>
</dependency>
```

GWT runs out of an exploded war directory. The Maven plugin places its artifacts into a directory named war. Compiled Java code should also code into the war directory.

```xml
<build>
  <outputDirectory>war/WEB-INF/classes</outputDirectory>
```
In the `build` section of the POM, configure the plugin:

```xml
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>gwt-maven-plugin</artifactId>
  <version>1.1-SNAPSHOT</version>
  <configuration>
    <runTarget>edu.pdx.cs410J.gwt.Examples/Hello.html</runTarget>
    <modules>
      <module>edu.pdx.cs410J.gwt.Examples</module>
    </modules>
    <extraJvmArgs>-Xmx512m</extraJvmArgs>
    <style>DETAILED</style>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>test</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```
GWT with Maven

The Maven plugin provides several Maven goals:

- **gwt:run** Builds the project and runs hosted mode
- **gwt:debug** Builds and runs the project in hosted mode, will wait for a debugger to attach
- **gwt:test** Runs GWT unit tests that subclass GWTTestCase

If your POM configures the `<goals>` as shown on the previous slide, then all you need to use is **gwt:run** and **gwt:debug**
You need to configure the war and Jetty plugin to look for files in a different place

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <warSourceDirectory>${basedir}/war</warSourceDirectory>
    <webappDirectory>${basedir}/war</webappDirectory>
    <webXml>${basedir}/src/main/webapp/WEB-INF/web.xml</webXml>
  </configuration>
</plugin>

<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>maven-jetty-plugin</artifactId>
  <configuration>
    <webAppSourceDirectory>${basedir}/war</webAppSourceDirectory>
    <connectors>
      <connector
        implementation="org.mortbay.jetty.nio.SelectChannelConnector">
        <port>8080</port>
      </connector>
    </connectors>
  </configuration>
</plugin>
```
GWT Widgets

The `com.google.gwt.user.client.ui` package contains classes for the user interface widgets.
A `Label` is a piece of plain text that usually describes what some other widget is or does

- The `HTML` widgets can contain markup to be rendered
- The `wordWrap` can be controlled

```java
class LabelExample extends Example {

    public LabelExample() {
        super("Labels");

        add(new Label("This is a Label"));
        add(new HTML("This label has <i>HTML</i>"));
        String longString =
        "This label is very long and has word wrap enabled";
        add(new Label(longString, true));
        String longString2 =
        "This label is very long and does not have word
        wrap enabled";
        add(new Label(longString2, false));
    }
}
```
**ListBox**

A ListBox contains a list of strings

- Strings are added with the `addItem` method
- Can be a drop down (default) or a multi-select
- The `getSelectedItems` and `getSelectedIndex` return the selected string or its index
- Can also specify the number of items that are visible
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.user.client.ui.ListBox;

public class ListBoxExample extends Example {
    public ListBoxExample() {
        super("List Box");

        ListBox box = new ListBox();
        box.addItem("Monday");
        box.addItem("Tuesday");
        box.addItem("Wednesday");
        box.addItem("Thursday");
        box.addItem("Friday");

        add(box);

        ListBox multi = new ListBox(true);
        multi.addItem("Chocolate");
        multi.addItem("Vanilla");
        multi.addItem("Strawberry");

        add(multi);
    }
}

ListBox
TextBox

A TextBox lets the user enter a single line of text

- Has a VisibleLength and a MaxLength
- getText returns the text entered by the user

```java
package edu.pdx.cs410J.gwt.client;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.PasswordTextBox;

public class TextBoxExample extends Example {
    public TextBoxExample() {
        super("Text Boxes");
        TextBox text = new TextBox();
        text.setVisibleLength(15);
        text.setMaxLength(10);
        add(text);
        TextBox readOnly = new TextBox();
        readOnly.setReadOnly(true);
        readOnly.setVisibleLength(15);
        readOnly.setText("Read only text");
        add(readOnly);
        PasswordTextBox password = new PasswordTextBox();
        password.setVisibleLength(15);
        add(password);
    }
}
```
**TextArea and RichTextArea**

A `TextArea` allows the user to input multiple lines of text

- Can set the width (`CharacterWidth`) and height (`VisibleLines`) in characters

A `RichTextArea` allows a user to edit formatted text

- Content can be accessed with `getHTML` and `setHTML`

- The `BasicFormatter` configures
  - `BackColor`, `ForeColor`, `Bold`, `Italic`, etc.

- The `ExtendedFormatter` lets you
  - `insertHorizontalRule` (`<hr/>`), `insertImage` URL, `insertOrderedList` and `insertUnorderedList`

- If the browser does not support the formatting, the formatter will be `null`
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.user.client.ui.TextArea;
import com.google.gwt.user.client.ui.RichTextArea;

public class TextAreaExample extends Example {
    public TextAreaExample() {
        super("Text Area");

        TextArea text = new TextArea();
        text.setCharacterWidth(30);
        text.setVisibleLines(5);
        add(text);

        RichTextArea rich = new RichTextArea();
        rich.setHTML("Rich <i>text</i> <b>area</b>");
        add(rich);
    }
}

TextArea and RichTextArea
GWT offers several kinds of buttons

- **Button**: click to perform some action

- **CheckBox**: A check box

- **PushButton**: Can be “up” or “down”
  - Add style for up, down, hovering, and disabled
  - Has an `isDown` method

- **ToggleButton**: Like a PushButton, but stays in the down state when clicked
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.CheckBox;
import com.google.gwt.user.client.ui.PushButton;
import com.google.gwt.user.client.ui.ToggleButton;

public class ButtonsExample extends Example {
   public ButtonsExample() {
      super("Buttons");

      Button button = new Button("Disabled button");
      button.setEnabled(false);
      add(button);

      CheckBox check = new CheckBox("Check box");
      check.setChecked(true);
      add(check);

      PushButton push =
         new PushButton("Click me", "Clicked");
      add(push);

      ToggleButton toggle =
         new ToggleButton("Up", "Down");
      add(toggle);
   }
}
Window

While not strictly a widget, the Window class provides information about the browser window in which the application is rendered via static methods

- alert, confirm, prompt pop up modal dialog
- Register CloseHandler and ResizeHandler
- print the contents of the page
- setTitle and setStatus
GWT Containers

Some widgets are responsible for containing and arranging (laying out) other widgets
Horizontal and Vertical Layouts

HorizontalPanel and VerticalPanel layout widgets in a single column or row

class AxisPanelExample extends Example {
    public AxisPanelExample() {
        super("Axis Panels");
        HorizontalPanel horiz = new HorizontalPanel();
        horiz.add(new Button("A"));
        horiz.add(new Button("B"));
        horiz.add(new Button("C"));
        add(horiz);

        VerticalPanel vert = new VerticalPanel();
        vert.add(new Label("1"));
        vert.add(new Label("2"));
        vert.add(new Label("3"));
        add(vert);
    }
}
DockPanel

DockPanel has five compartments: north, south, east, west, and center

package edu.pdx.cs410J.gwt.client;

import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.Button;

public class DockPanelExample extends Example {
    public DockPanelExample() {
        super("Dock Panel");
        DockPanel dock = new DockPanel();
        dock.add(new Button("North"), DockPanel.NORTH);
        dock.add(new Button("South"), DockPanel.SOUTH);
        dock.add(new Button("East"), DockPanel.EAST);
        dock.add(new Button("West"), DockPanel.WEST);
        dock.add(new Button("Center"), DockPanel.CENTER);
    }
}

**DeckPanel**

**DeckPanel** contains multiple widgets, but only displays one of them at a time

- Like a deck of cards

- Contained widgets are referenced by their index in the **DeckPanel**
  - `getWidgetIndex(Widget)`, `showWidget(int index)`, `getVisibleWidget()`

`edu.pdx.cs410J.gwt.client.DeckPanelExample`
StackPanel

StackPanel stacks widgets in a single column and only reveals one at a time

```java
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.user.client.ui.StackPanel;
import com.google.gwt.user.client.ui.Button;

public class StackPanelExample extends Example {
    public StackPanelExample() {
        super("Stack Panel");

        StackPanel stack = new StackPanel();
        stack.add(new Button("One"), "One");
        stack.add(new Button("Two"), "Two");
        stack.add(new Button("Three"), "Three");
        stack.add(new Button("Four"), "Four");
        add(stack);
    }
}
```
TabPanel

TabPanel displays only one of its children at a time and has a row of tabs at its top

```java
class TabPanelExample extends Example {
    public TabPanelExample() {
        super("Tab Panel");

        TabPanel tabs = new TabPanel();
tabs.add(new Button("One"), "One");
tabs.add(new Button("Two"), "Two");
tabs.add(new Button("Three"), "Three");
tabs.add(new Button("Four"), "Four");
tabs.selectTab(0);
        add(tabs);
    }
}
```
Handling user events

When the user interacts with a widget, an event is generated

- Events are handled by handlers registered on the widget on which the user acts

- Handlers are implemented as an interface containing one method that handles an event*
  - An event object contains relevant information such as the widget that received the event and other pertinent information (x/y coordinates of mouse click, e.g.)

- Handlers are often implemented using anonymous inner classes
  - The implementation only makes sense with respect to the one widget
  - If you’re worried about performance, you might want to handle multiple events (and events for multiple widgets) in a single class

*Prior to GWT 1.5, events were handled by “listener” interfaces with methods that took one or two arguments
Events and Event Handlers

Every event handler interface as a corresponding “Has” interface that is implemented by widgets that generate the event

- HasClickHandlers is implemented by to all FocusWidgets: buttons, Image, Label, text widgets

- The “Has” interface “add” method returns a HandlerRegistration that as a removeHandler method

Event objects are subclasses of GWTEvent

- The getSource method returns the widget that fired the event

- Most subclasses add more state
  - ResizeEvent has getHeight and getWidth
Handling mouse clicks

This program pops up an alert dialog when a button is clicked

```java
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;

public class ClickHandlerExample extends Example {
    public ClickHandlerExample() {
        super("Click Handler");

        Button button = new Button("Click me");
        button.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent clickEvent) {
                Window.alert("I was clicked");
            }
        });
        add(button);
    }
}
```
Dialog boxes

A DialogBox is shown in response to some action

- A dialog box can be automatically hidden when the user clicks outside of it

- A modal dialog box prevents the user from interacting with the rest of the UI while the dialog is displayed

- A DialogBox has a title and contains another widget for its content

From edu.pdx.cs410J.gwt.client.DialogBoxExample

```java
private DialogBox createDialogBox(boolean autoHide, boolean modal) {
    final DialogBox box = new DialogBox(autoHide, modal);
    box.setText("Auto-hide dialog");

    DockPanel panel = new DockPanel();
    panel.add(new Label("Click outside me"), DockPanel.CENTER);
    panel.add(new Button("Close", new ClickHandler() {
        public void onClick(ClickEvent event) {
            box.hide();
        }
    }), DockPanel.SOUTH);
    box.setWidget(panel);
    box.center();
    return box; }
```
Types of Event Handlers

A ChangeHandler’s onChange method is invoked when the contents of a widget changes

- Applies to ListBox and the text widgets (TextBox, TextArea, etc.)

A FocusHandler’s onFocus method is invoked when a widget gets keyboard focus

- Applies to all FocusWidgets: buttons, Image, Label, text widgets

- BlurHandler’s onBlur method is invoked when a widget looses keyboard focus

There are three handlers for keyboard events: KeyUpHandler, KeyDownHandler, and KeyPressedHandler

- KeyPressEvent has a getCharCode method that returns the char that was pressed

- KeyUpEvent and KeyDownEvent have getNativeKeyCode that also represents keys like “Home” and “Delete” (see KeyCodes)
Mouse events

There are several kinds of these events fired in response to mouse operations in a widget:

- `MouseDownEvent`, `MouseUpEvent`, `MouseMoveEvent`, `MouseOutEvent`, `MouseOverEvent`, and `MouseWheelEvent`

- Applies to `FocusPanel`, `HTML`, `Image`, `Label`, and `RichTextArea`

- `MouseWheelEvent` gets a `MouseWheelVelocity` that specifies the direction (north or south) and a measurement of velocity (`getDeltaY` that the wheel was scrolled)

```java
edu.pdx.cs410J.gwt.client.FocusPanelExample
```
GWT Application Implementation

So, most GWT applications follow these steps

1. Create and configure widgets

2. Arrange widgets in a container Panel

3. Register event handlers to handle user interaction
Remote Procedure Calls

GWT provides a framework for invoking service methods whose implementation lies on the server

The service’s contract is specified by an interface that extends RemoteService

```java
package edu.pdx.cs410J.gwt.client;

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

@RemoteServiceRelativePath("division")
public interface DivisionService extends RemoteService {
  public int divide(int dividend, int divisor);
}
```

Note that the service interface resides in the client package
Remote Procedure Calls

The service is implemented by a class in the server package that extends RemoteServiceServlet

```java
package edu.pdx.cs410J.gwt.server;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import edu.pdx.cs410J.gwt.client.DivisionService;

public class DivisionServiceImpl
    extends RemoteServiceServlet
    implements DivisionService {

    public int divide(int dividend, int divisor) {
        return dividend / divisor;
    }
}
```

The service is actually a servlet that is deployed in a web application

```xml
<servlet>
    <servlet-name>divisionServlet</servlet-name>
    <servlet-class>edu.pdx.cs410J.gwt.server.DivisionServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>divisionServlet</servlet-name>
    <url-pattern>/examples/division</url-pattern>
</servlet-mapping>
```
Remote Procedure Calls

All calls to the server in GWT are asynchronous (the remote call immediately returns and the application is not blocked)

- The result of the remote call is delivered to either the onSuccess or the onFailure method of a AsyncCallback object

- The client code works with an async version of the remote service interface

```java
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface DivisionServiceAsync {
    void divide(int dividend, int divisor, AsyncCallback async);
}
```
Remote Procedure Calls

The client uses the magical GWT class to create an instance of the remote service

- The URL of the service (ServiceDefTarget) must be configured in the client

```java
DivisionServiceAsync service =
    GWT.create(DivisionService.class);
```

The @RemoteServiceRelativePath annotation on the service interfaces tells the client code how to find the service on the server

- Prior to GWT 1.5, the client had to explicitly register the async service with the GWT runtime
Remote Procedure Calls

Finally, when invoking the remote service, a AsyncCallback instance is used to handle the response

```java
int d1 = Integer.parseInt(dividend.getText());
int d2 = Integer.parseInt(divisor.getText());
service.divide(d1, d2, new AsyncCallback() {
    public void onFailure(Throwable ex) {
        Window.alert(ex.toString());
    }

    public void onSuccess(Object o) {
        int result = (Integer) o;
        quotient.setText(String.valueOf(result));
    }
});
```


Localization

Modern applications (especially web applications) should support multiple locales

- You want your application to be familiar to your users
- Numbers, currencies, dates, and times formatted like they expect
- Text and messages displayed in their native language

The com.google.gwt.i18n.client package contains facilities for localizing data

- DateTimeFormat, NumberFormat

GWT also takes a unique approach to localizing strings: Constants and Messages
Localizing String Constants

Translating the text of your application after it has been written is laborious

GWT allows you define Java interfaces whose methods return string constants

- Your interface must extend
  com.google.gwt.i18n.client.Constants

- The @Constants.DefaultStringValue specifies the default value of a string constant

- Constants can also have a type of int, boolean, double, float, string array, or string map

Your program gets an instance of the constant interface using GWT.create

- The GWT compiler generates efficient code for accessing the string constants
Localizing String Constants

```java
package edu.pdx.cs410J.gwt.client;

import com.google.gwt.i18n.client.Constants;

public interface DateLocalizationStrings
        extends Constants {

    @DefaultStringValue( "Input your date of birth" )
    public String selectString();

    @DefaultStringValue( "How many days have you been alive?" )
    public String defaultDaysMessage();
}

From DateLocalizationExample:

DateLocalizationStrings strings =
        GWT.create(DateLocalizationStrings.class);
this.label = new Label();
this.label.setText(strings.defaultDaysMessage());
this.add(new Label(strings.selectString()));
```
Localizing Messages

Often an application will issue messages that have variable content

- An error message that contains malformed data inputted by the user

Messages can be localized using the Messages interface

```java
default package edu.pdx.cs410J.gwt.client;
import com.google.gwt.i18n.client.Messages;
public interface DateLocalizationMessages extends Messages {
    @DefaultMessage("You have been alive for \{0,number\} days")
    String daysMessage(long days);
}
```

Using an interface provides type-safe message formatting

- The GWT compiler will complain if the message string doesn’t match the signature of the interface method
Localizing Text

Messages and constants are localized in properties files that reside in the same package as the interfaces:

In edu/pdx/cs410J/gwt/client

From DateLocalizationMessages_es.properties:

daysMessage = Usted ha estado vivo durante \{0,number} d\u00edas

From DateLocalizationStrings_es.properties

selectString = Ingrese su fecha de nacimiento
defaultDaysMessage = \u00bfCu\u00e1ntos \d\u00edas has vivido?

Note that the key in the property file must match the name of the interface method

• There is no “get” convention

• Unlike the JDK, GWT handles properties files that are UTF-8 encoded

*In a GWT Maven project, I like to put my properties file in src/main/resources
Configuring for Localization

You need to let the GWT module know which locales are supported

From Examples.gwt.xml

```xml
<module rename-to="examples">
  <inherits name="com.google.gwt.user.User"/>
  <inherits name="com.google.gwt.i18n.I18N"/>
  <entry-point
    class="edu.pdx.cs410J.gwt.client.Examples"/>
  
  <extend-property name="locale" values="es"/>
</module>
```

For maximum efficiency, the GWT compiler outputs another “permutation” of the module that has its strings localized for Spanish

- Minimizes the number of runtime calls to get the localized strings
- Lets code be inlined, etc.
Running in a Locale

There are a couple of ways to run a GWT application in a specific Locale

- Add the `locale=es` to the URL

- Add `<meta name="gwt:property" content="locale=es">` to the applications HTML `<head>`
  - Get the locale from the `Accept-Language` HTTP request header
  - Something like a JSP can be used to set the `meta` tag dynamically
**Browser History**

In an RIA, what happens when the user presses the “back” button?

- Browser will go back to the previous page (probably eBay, right?)

- When user presses the “forward” button, the GWT app reloads
  - All of the UI state is lost (unless you add a lot of logic to your app)

GWT has support for maintaining application state so that the back/forward buttons can be meaningful

- State is kept in the URL fragment (anchor)
  - Navigating to an anchor doesn’t reload the page

- Also let’s an application have URLs that can be bookmarked
History in GWT

The `com.google.gwt.user.client.History` class handles browser history.

It relies on an invisible `iframe` in the application’s HTML file to handle the previous/next behavior:

```html
<iframe src="javascript:''"
    id="__gwt_historyFrame"
    style="width:0;height:0;border:0"></iframe>
```

When your application displays a “place” which is worth bookmarking or navigating back to, it can push a history “token” on the stack with `History.newItem()`

```java
tabs.addSelectionHandler(new SelectionHandler<Integer>() {
    public void onSelection(SelectionEvent<Integer> event) {
        ExamplePanel panel = (ExamplePanel) tabs.getWidget(event.getSelectedItem());
        History.newItem(panel.getTitle(),
            false /* Don’t fire change event */);
    }
});
```
History in GWT

When the user navigates to a URL, a `ValueChangeEvent` is delivered to the `History` framework

- The event’s `value` is the URL fragment

```java
History.addValueChangeHandler(
    new ValueChangeHandler<String>() {
        public void onValueChange(
            ValueChangeEvent<String> event) {
            String token = event.getValue();

            // Parse relevant information from token
            // Display the desired application state
        }
    })
)
```

In your application’s `onModuleLoad`, you’ll want to invoke `History.fireCurrentHistoryState()` to put your app into the desired initial state
Testing GWT code

Server-side GWT code is just Java and can be tested with JUnit, mockito, etc.

Client-side GWT code needs some special consideration

- Plain-old Java business logic can be tested using plain-old JUnit

- UI (Widget) code is rendered with “native” JavaScript that accesses DOM, calls GWT-RPC services, only uses JRE emulation APIs, etc.

- The GWT class does magic things in the browser

GWTTestCase provides framework for writing JUnit tests that are executed in development mode

- Uses the HtmlUnit library to simulate a browser environment

- The getModuleName method returns which GWT modules is being test

- Run with a plain-old JUnit test runner which logs output, etc.
Testing GWT code

Because they may invoke GWT-RPC services, GWT tests are often asynchronous

- Test has to wait for something to happen asynchronously before verifying test state

- But java.util.Thread isn’t supported by JRE Emulation

GWTTestCase provides methods to support asynchrony

- finishTest tells GWTTestCase that the test has validated its state

- delayTestFinish tells GWTTestCase to wait a given number of milliseconds for finishTest to be called
  - delayTestFinish puts the test into “asynchronous mode”
  - If finishTest isn’t called within the timeout, the test fails with a TimeoutException

- GWT’s Timer class can be used to wait a little bit for validating state
GWT Module for testing

GWT tests are more like “integration” tests than unit tests

- Depend on development mode and GWT server
- You may want to define a GWT module just for these tests*

```xml
<module rename-to='it'>
  <inherits name='edu.pdx.cs410J.gwt.Examples'/>

  <source path="client">
    <include name="**/*.java"/>
    <!-- Exclude plain-old JUnit tests that may use non-emulation APIs -->
  </source>

  <!-- Need to define the servlet in .gwt.xml in order to use service in a GwtTest -->
  <servlet path="/division" class="edu.----.DivisionServiceImpl"/>

</module>
```

*The GWT Maven plugin recommends that you run GWT tests are part of the integration-test phase.*
package edu.pdx.cs410J.gwt.client;

public class DivisionServiceExampleGwtTest  
   extends GWTTestCase {

   @Override
   public String getModuleName() {
      return "edu.pdx.cs410J.gwt.IntegrationTests";
   }

   @Test
   public void testDivisionUI() {
      final DivisionServiceExample ui =
         new DivisionServiceExample();
      int dividend = 6;
      int divisor = 3;
      final int quotient = 2;

      ui.dividend.setText(String.valueOf(dividend));
      ui.divisor.setText(String.valueOf(divisor));
      click(ui.equals);

   }

   In order to test it, DivisionServiceExample exposes its UI components (dividend, divisor, etc.)
   
   • Test class is in the same package so fields can be package-private
Writing a GWT test

Clicking buttons in a GWT test is surprisingly difficult

- Button.click() doesn’t work

- Have to create a native browser event yourself

```java
private void click(Button button) {
    NativeEvent event = Document.get().createClickEvent(0, 0, 0, 0, 0, false, false, false, false);
    DomEvent.fireNativeEvent(event, button);
}
```

Lame.
Writing a GWT test

Since clicking the equals button invokes a GWT-RPC service, we have to wait until the service returns before verifying

```java
Timer verify = new Timer() {
    @Override
    public void run() {
        assertEquals(String.valueOf(quotient),
                    ui.quotient.getText());
        finishTest();
    }
};

// Wait for the RPC call to return
verify.schedule(500);

delayTestFinish(1000);
```
More efficient GWT tests

Each GWT test has a lot of overhead

- Launching development mode, compiling Java into JavaScript, etc.

A GWTTestSuite contains a bunch of GWTTestCaseS

- Analyze test cases and only launches one development mode “shell” per module being tested

- All test cases for a module are run in the same shell

```java
public class ExamplesGwtTestSuite
    extends TestCase {

    public static Test suite() {
        GWTTestSuite suite =
            new GWTTestSuite("GWT Examples");

        suite.addTestSuite(DivisionServiceGwtTest.class);
        suite.addTestSuite(DivisionServiceExampleGwtTest.class);

        return suite;
    }
}
```
Summary

Google Web Toolkit allows Rich Internet Applications to be developed using exclusively the Java programming language

- Java source code is compiled into JavaScript that is run in the web browser

- Programming model closely resembles developing desktop applications
  - Classes that represent user interface widgets, containers that lay out widgets, and listeners that handle user events
  - “Hosted mode” allows for easy debugging and quick turnaround of changes
  - Localizable strings and messages are modeled using Java interfaces to provide type safety

- The client communicates asynchronously with the service via a Remote Procedure Call framework

- GWT code can be tested like any other Java code
  - However, testing Widget code requires special consideration