ISMLA Session 7 - GWT

Björn Rudzewitz

Tübingen University

December 4, 2017

Plan

1. Motivation
2. Components
3. Project Structure
   - Module Descriptor
   - Entry Point
   - src
   - war directory
   - WEB-INF directory
4. Developing with GWT – Clientside
   - RootPanel
   - Widgets

Motivation

"Industrial-Strength Multilingual Language Analysis"

- interpretations of "industrial-strength":
  1. robust, scalable software
  2. product/application for end users
- second part of the technical strand: learn technologies for building applications to make functionality available to end users

Motivation

- so far: building NLP applications to analyze language
- runnable on local machine by developers via command line or IDE
- How to make the technology accessible to end users ?

  $\rightarrow$ application

Motivation

Application building:

- creating a platform-specific application, e.g. Android app
- creating an application in principle supported by most platforms, e.g. a Java runnable jar
- creating a web service independent of a specific platform or installed runtime

Motivation

Application building:

- creating a platform-specific application, e.g. Android app
- creating an application in principle supported by most platforms, e.g. a Java runnable jar desktop app
- creating a web service independent of a specific platform or installed runtime

→ GWT: develop platform-independent web applications in Java *like* you would develop desktop applications

Motivation

- browsers interpret HTML, JavaScript, CSS
- client code of web applications needs to be written* in these languages

Motivation

- browsers interpret HTML, JavaScript, CSS
- client code of web applications needs to be written* in these languages
- * or compiled to
- GWT as a Java-to-JavaScript/HTML/CSS compiler

Why GWT ?

- allows to use the full range of Java development tools like IDEs, debuggers, or unit tests for development to generate JavaScript optimized for every browser
- allows to make use of the power of the Java programming language (strict types, inheritance, . . .)
- complete code base of AJAX application inluding client and server code in one language developed jointly
- no need to learn JavaScript, no need to translate between the different languages/technologies from the client and the server side
- parts to assemble the application can be used out-of-the-box

GWT Components/Tools

1. compiler
   compile a system version for every browser
2. user interface
   cross-browser UI layer/components
3. RPC
   (de)serialization of objects between client and server and asynchronous response handling
4. further utilities
   history system, native code functions, test server, GWT shell...

cf. [Cooper and Collins, 2008]

GWT Components/Structure

Module

- module: corresponds to Java project
- components:
  1. XML configuration (compilation/inheritance/entry point)
  2. Java code (client/server/shared)
  3. CSS
  4. HTML host page with link to compiled JavaScript source

Starting a GWT project

Installing the GWT plugin

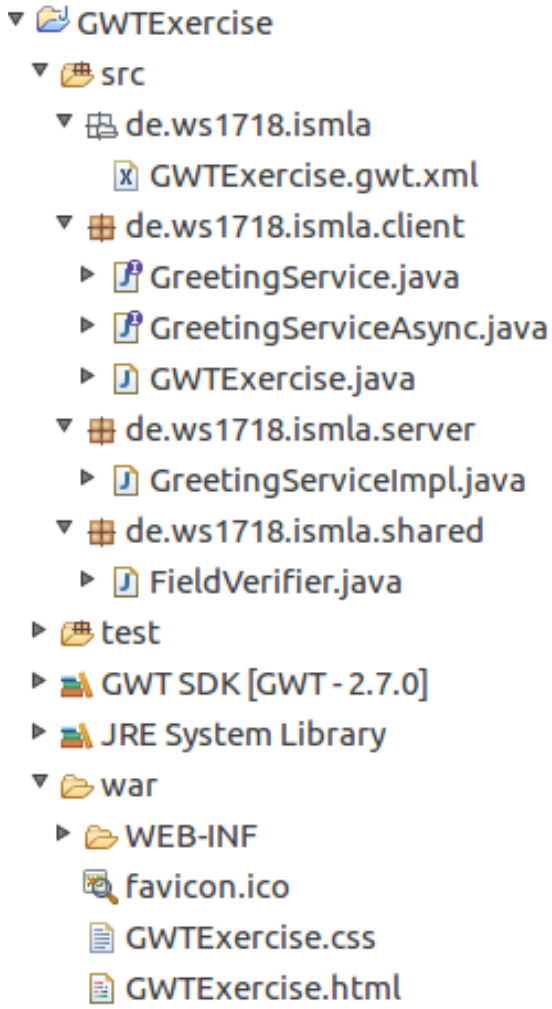
- Open the window *Help* → *Install New Software . . .*
- Select the correct or closest matching variant of the GWT plugin from the distribution page
  https://developers.google.com/eclipse/docs/download
- Install the software (ignore the Android tools)

Starting a GWT project

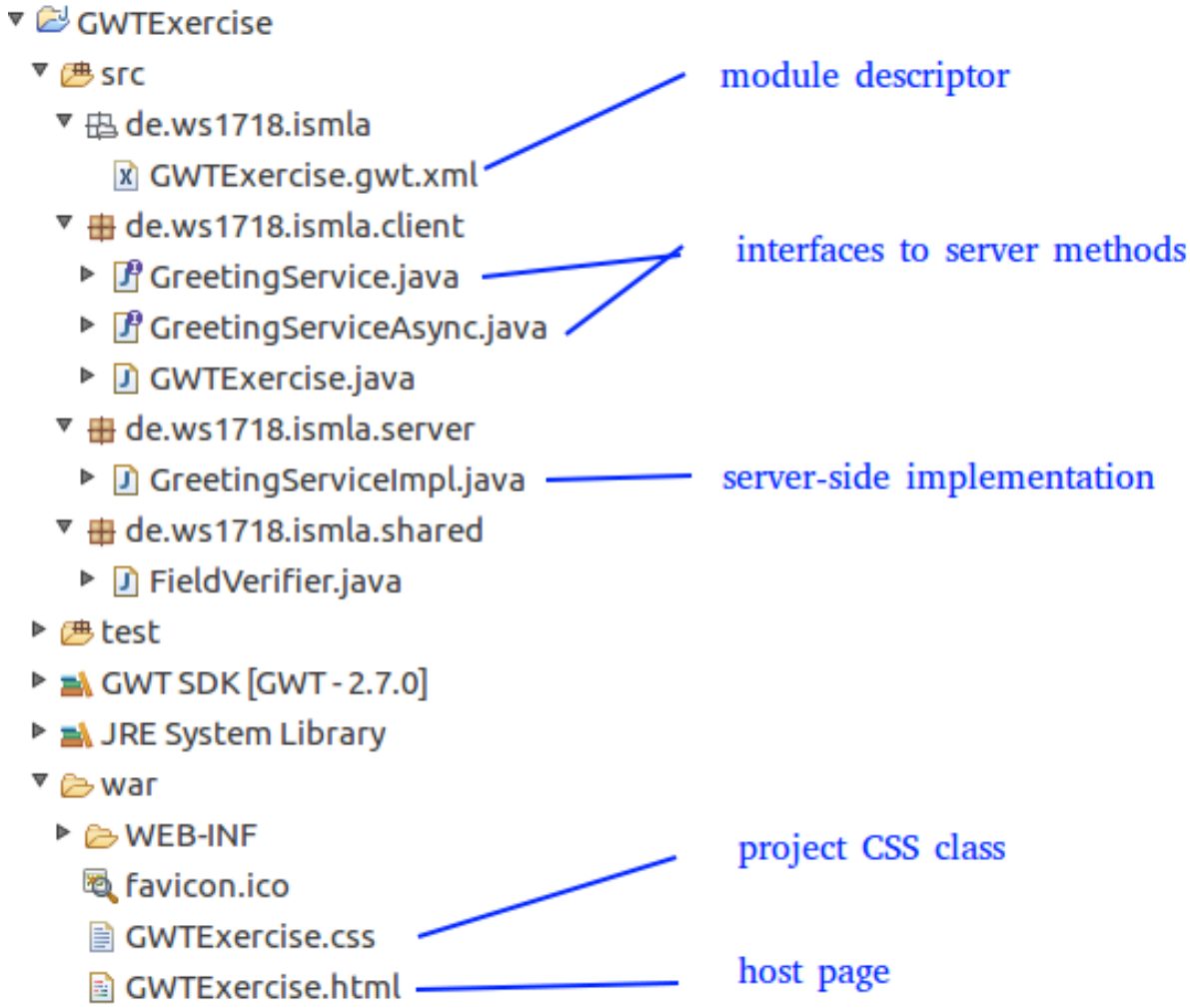
Creating a base project

1. *File* $\rightarrow$ *New* $\rightarrow$ *Google* $\rightarrow$ *Web Application Project*
2. enter project name and namespace
3. uncheck the option to use Google App Engine
4. leave the option to generate sample code checked

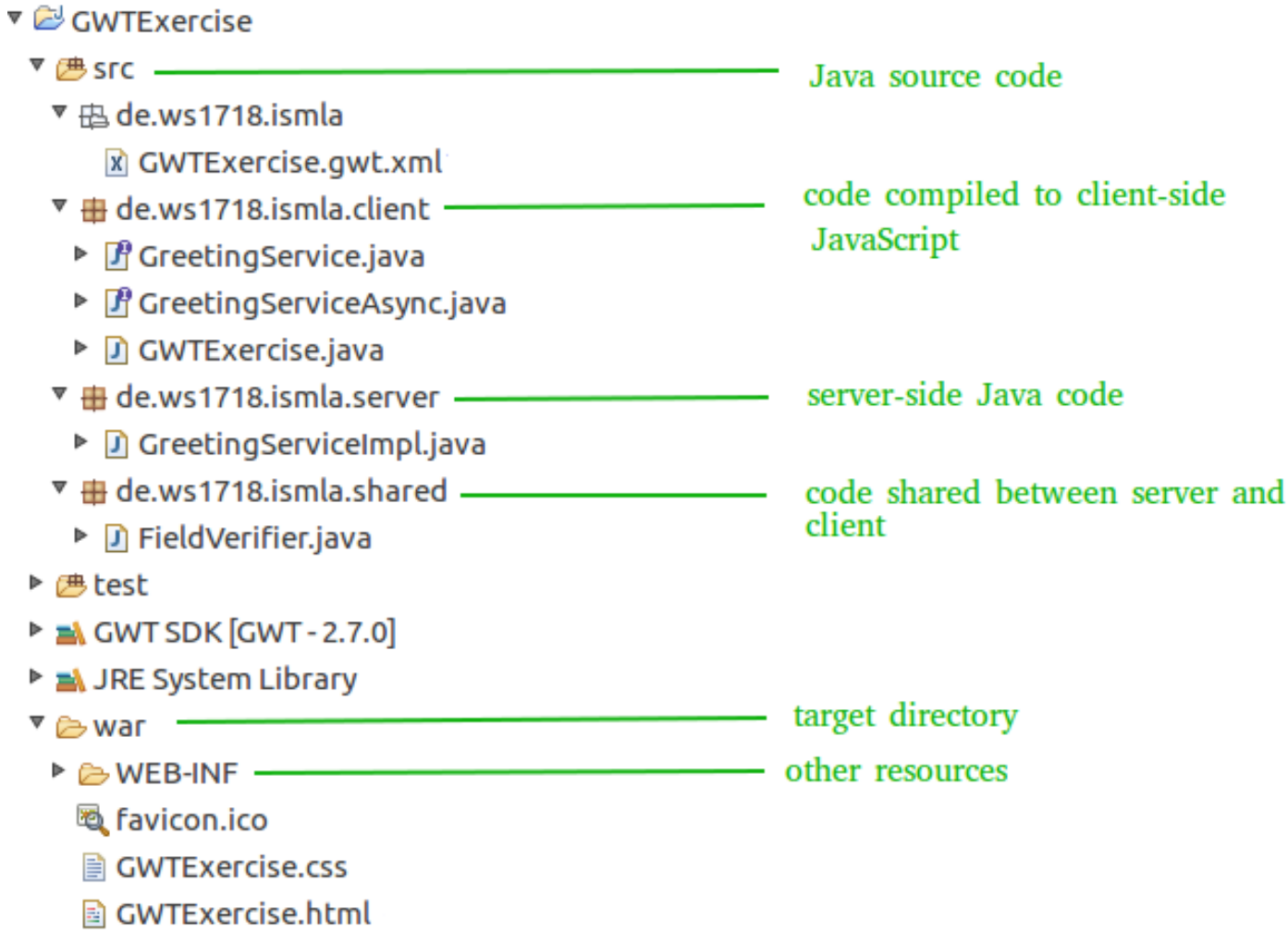
Project Structure



Project Structure



Project Structure



Module Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  When updating your version of GWT, you should also update this DTD reference,
  so that your app can take advantage of the latest GWT module capabilities.
-->
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.7.0//EN"
  "http://gwtproject.org/doctype/2.7.0/gwt-module.dtd">
<module rename-to='gwtexercise'>
  <!-- Inherit the core Web Toolkit stuff.                        -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.  You can change       -->
  <!-- the theme of your GWT application by uncommenting          -->
  <!-- any one of the following lines.                            -->
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>
  <!-- <inherits name='com.google.gwt.user.theme.standard.Standard'/> -->
  <!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->
  <!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>     -->

  <!-- Other module inherits                                      -->

  <!-- Specify the app entry point class.                         -->
  <entry-point class='de.ws1718.ismla.client.GWTExercise'/>

  <!-- Specify the paths for translatable code                    -->
  <source path='client'/>
  <source path='shared'/>

  <!-- allow Super Dev Mode -->
  <add-linker name="xsiframe"/>
</module>
```

Module Descriptor

- name of the application/module
- possibility of inheriting/including other modules
- path to source code packages for compilation
- entry point class: class first executed when client accesses the web application

Entry Point

- Java entry point class
- defines the code executed when a client opens the URL
- implements `com.google.gwt.core.client.EntryPoint`
- defines the function `public void onModuleLoad()`

src

src folder divided into several main packages

1. "default" package:
   contains the module descriptor
2. client
   compiled to JavaScript; User Interface, application logic behind interface; Entry Point
3. server
   server-side Java code, servlet logic for reacting to client requests
4. shared
   compiled to JavaScript; code shared between server and client

war directory

- contains the compiled version of the project
- compile: *Right-Click* $\rightarrow$ *Google* $\rightarrow$ *GWT compile*
- contains the WEB-INF directory with potential additional resources
- for deploying the application on a server it has to be compiled (*Right-click* $\rightarrow$ *GWT Compile*) and the content of the war file needs to be packaged into a .war file and deployed in a tomcat server

WEB-INF directory

- contains potential additional resources like images or libraries
- contains the web.xml file defining the services/servlets and the host page

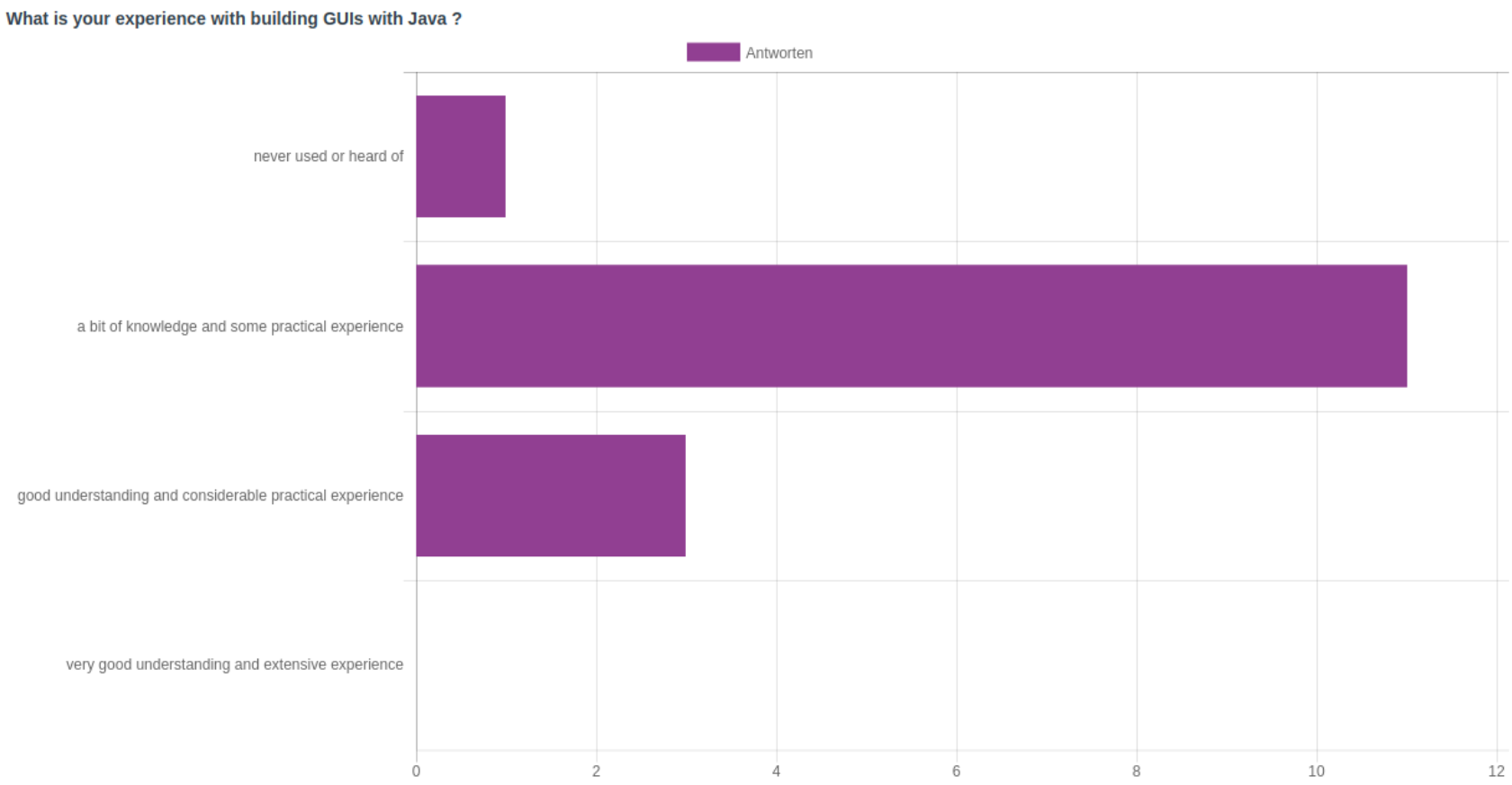
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
              http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5"
         xmlns="http://java.sun.com/xml/ns/javaee">

  <!-- Servlets -->
  <servlet>
    <servlet-name>greetServlet</servlet-name>
    <servlet-class>de.ws1718.ismla.server.GreetingServiceImpl</servlet-class>
  </servlet>

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

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

</web-app>
```

Running the GWT Project

- *Right-Click* $\rightarrow$ *Run as* $\rightarrow$ *Web Application (GWT Super Dev Mode)*
- copy URL into browser (or set default browser in Eclipse)

Developing with GWT – Clientside



Grafikdaten verbergen

	Antworten
never used or heard of	1 (6,67 %)
a bit of knowledge and some practical experience	11 (73,33 %)
good understanding and considerable practical experience	3 (20,00 %)
very good understanding and extensive experience	0

Developing with GWT – Clientside

- GWT allows to build client–side interfaces like desktop interfaces
- design patterns and design elements similar to other Java GUIs like Swing

Developing with GWT – Clientside

Basic UI elements:

- panels for basic layout,
  e.g. FlowPanel or HorizontalPanel
- widgets for special interaction functions,
  e.g. Button or TextBox
- event handlers attached to elements,
  e.g. ClickHandler or KeyListener

Developing with GWT – Clientside

- most fundamental UI element: `RootPanel`
- `RootPanel` is a static (i.e. always accessible) panel to which *all* other UI elements are added
- minimal example:

```
public void onModuleLoad() {
   // creating a web page with one button
   Button button = new Button("Push me");
   RootPanel.get().add(button);
}
```

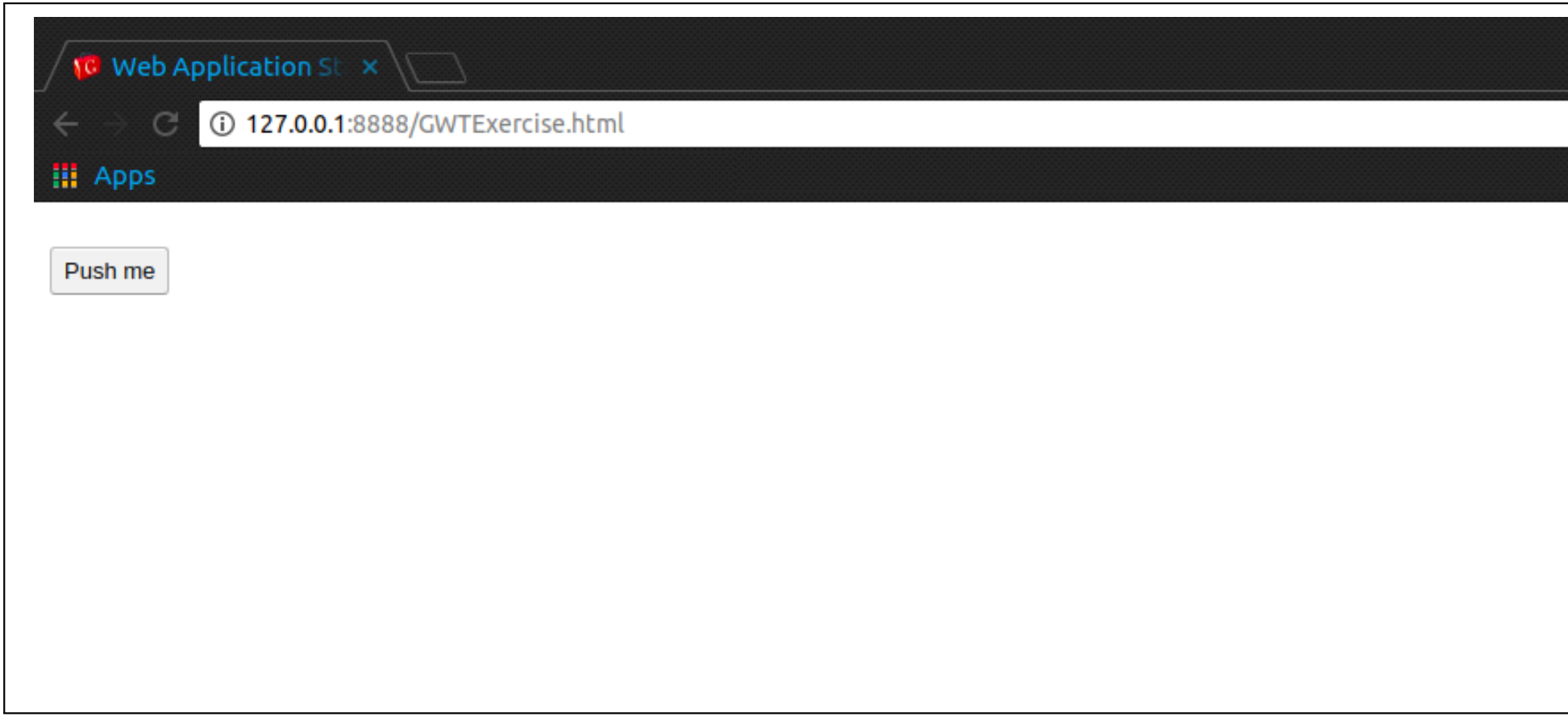
Developing with GWT – Clientside

Minimal example translates to:

```
<html>
▶<head>…</head>
  <!--                                                -->
  <!-- The body can have arbitrary html, or           -->
  <!-- you can leave the body empty if you want       -->
  <!-- to create a completely dynamic UI.             -->
  <!--                                                -->
▼<body>
    <!-- RECOMMENDED if your web app will not function without JavaScript
    enabled -->
  ▶<noscript>…</noscript>
    <h1>Web Application Starter Project</h1>
  ▶<table align="center">…</table>
  ▼<iframe src="javascript:"" id="gwtexercise" tabindex="-1" style=
  "position: absolute; width: 0px; height: 0px; border: none; left: -1000px;
  top: -1000px;">
    ▼#document
        <!DOCTYPE html>
      ▼<html>
          <head></head>
        ▼<body>
            <script src="http://127.0.0.1:9876/gwtexercise/
            F1A3611326F01F3AC2CDC8748B951909.cache.js"></script>
          </body>
        </html>
    </iframe>
    <button type="button" class="gwt-Button">Push me</button> == $0
  </body>
</html>
```

Developing with GWT – Clientside

Minimal example translates to:



Widgets

Selected Widgets I

- HTML
- Image
- Anchor
- Button
- CheckBox
- RadioButton

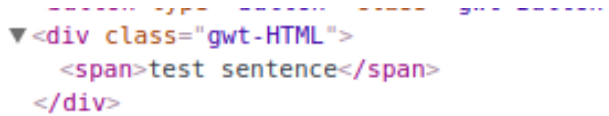
HTML

- translates to a *div* element with the specified string interpreted as HTML
- i.e. the *span* elements declared as a string are translated to elements
- in contrast, the *Label* widget interprets string as text, not HTML

HTML

- translates to a *div* element with the specified string interpreted as HTML
- i.e. the *span* elements declared as a string are translated to elements
- in contrast, the *Label* widget interprets string as text, not HTML

```
// HTML element
HTML html = new HTML("<span>test sentence</span>");
RootPanel.get().add(html);
```



Image

- creates an *img* element with the given URL as the value of the src atttribute

```
Image img = new Image("/img/20170703145537.jpg");
img.setWidth("200px");
img.setHeight("300px");
RootPanel.get().add(img);
```

Image

- attention: paths to images in the project are relative to the war folder
- i.e. in the given example the image lies in an img directory below war

```
<img src="/img/20170703145537.jpg" class="gwt-Image" style="width: 200px;
height: 300px;">
```

Anchor

- translates to an *a* element
- default behavior/constructor: do nothing; i.e. in the example below only for the second anchor the user is forwarded

```
Anchor anchor = new Anchor("a link");
RootPanel.get().add(anchor);

Anchor workingAnchor = new Anchor("another link", true,
"http://www.sfs.uni-tuebingen.de", "_blank");
RootPanel.get().add(workingAnchor);
```

Button

- translates to button element with the string given as argument as HTML (cf. example)

```
Button btnHtml = new Button(
"<span style=\"color:green;\">Push</span>");
```

CheckBox

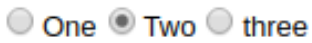
- creates an *input* element of type checkbox
- value (checked/unchecked) can be read out via `cb.getValue();`

```
CheckBox cb = new CheckBox("checkbox label");
```

RadioButton

- translates to *input* elements of type radio

```
// radio button
RadioButton rb1 = new RadioButton("group1", "One");
RadioButton rb2 = new RadioButton("group1", "Two");
RadioButton rb3 = new RadioButton("group1", "three");
```




```
▼<span class="gwt-RadioButton">
    <input type="radio" name="group1" value="on" id="gwt-uid-2" tabindex=
    "0">
    <label for="gwt-uid-2">One</label>
  </span>
▶<span class="gwt-RadioButton">…</span>
▶<span class="gwt-RadioButton">…</span>
```

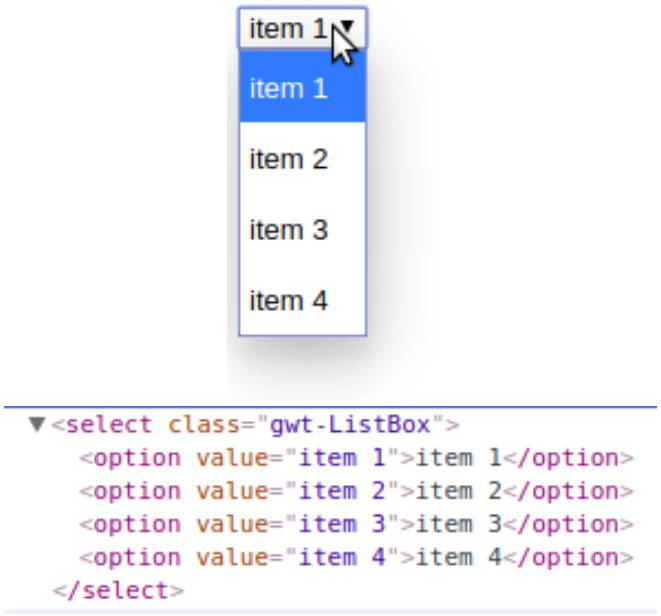
Widgets

Selected Widgets II

- ListBox
- TextBox
- TextArea

ListBox

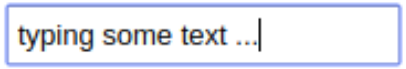
- list with values to select
- translated to *select* and *option* element
- can be changed from a dropdown list to a list with more visible value via `setVisibleItemCount`



TextBox and TextArea

- one-line input field for plain text
- compiled to *input* element of type text
- *TextArea*: like *TextBox* but with multiple lines and resizable

```
TextBox tb = new TextBox();
RootPanel.get().add(tb);
```



Widgets

Selected Widgets III

- Tree
- MenuBar
- DatePicker

Tree

```
Tree tree = new Tree();
TreeItem root = new TreeItem();
root.setText("root node");
root.addTextItem("item 1");
root.addTextItem("item 2");
TreeItem ti = new TreeItem(new HTML(
"<span style=\"color:green\">item 3</span>"));
ti.addTextItem("item 4");
ti.addTextItem("item 5");
root.addItem(ti);
TreeItem ti2 = new TreeItem(new Label("item 6"));
ti2.addTextItem("item 7");
root.addItem(ti2);
tree.addItem(root);
RootPanel.get().add(tree);
```

Tree

- generates a collabsible tree widget
- possibility to use other widgets as tree items (cf. green html)

MenuBar

- generates a menu bar with possible sub menus
- translates to rather complex HTML


```
▼<div tabindex="0" role="menubar" class="gwt-MenuBar gwt-MenuBar-
horizontal" hidefocus="true" style="outline: 0px;" aria-activedescendant=
"gwt-uid-18">
   <input type="text" tabindex="-1" role="presentation" style="opacity:
   0; height: 1px; width: 1px; z-index: -1; overflow: hidden; position:
   absolute;">
  ▼<table>
    ▼<tbody>
      ▼<tr>
          <td class="gwt-MenuItem" id="gwt-uid-14" role="menuitem">item 1
          </td>
          <td class="gwt-MenuItem" id="gwt-uid-17" role="menuitem" aria-
          haspopup="true">item 2</td>
          <td class="gwt-MenuItem" id="gwt-uid-18" role="menuitem">item 5
```

MenuBar

```
MenuBar mb = new MenuBar();
Command c = new Command() {
@Override
public void execute() {
        Window.alert("Menu item clicked");
}};

mb.addItem("item 1", c);

MenuBar mb2 = new MenuBar(true);
mb2.addItem("item 3", c);
mb2.addItem("item 4", c);
mb.addItem("item 2", mb2);

mb.addItem("item 5", c);
RootPanel.get().add(mb);
```

MenuBar

- generates a calendar widget
- date selected by user retrieved via `getValue()`

```
DatePicker dp = new DatePicker();
RootPanel.get().add(dp);
```

Widgets

Selected Widgets IV

- CellTree
- CellList
- CellTable

Cell Widgets

- *CellTree*, *CellList*, *CellTable* follow similar design patterns to quickly display large amounts of arbitrary data
- selection model determines handling of actions
- data provider to update data, changes in data are rendered automatically to the frontend

explanations, examples, design patterns:
http://www.gwtproject.org/doc/latest/DevGuideUiCellWidgets.html

Widgets

Selected Panels

- FlowPanel
- HorizontalPanel
- VerticalPanel
- HTMPanel
- ScrollPanel
- PopupPanel
- DialogBox

Panels

- different panels for pre-structuring of layout
- *HorizontalPanel* allows for very quick layout in horizontal manner
- but: *HorizontalPanel* translates to *table* element, making fine-grained styling very hard due to invariable table structure
- for layout going beyond basic style: preferably use *FlowPanel* or *HTMLPanel* and set style via CSS rules (not elements)

Panels

```
FlowPanel fp = new FlowPanel();
fp.add(new Button("Button 1"));
fp.add(new Button("Button 2"));
fp.add(new Button("Button 3"));
RootPanel.get().add(fp);

VerticalPanel vp = new VerticalPanel();
vp.add(new Button("Button 1"));
vp.add(new Button("Button 2"));
vp.add(new Button("Button 3"));
RootPanel.get().add(vp);
```

Panels

Note the difference in complexity in the translated code for equally complex Java source code:


```
▼<div>
    <button type="button" class="gwt-Button">Button 1</button>
    <button type="button" class="gwt-Button">Button 2</button>
    <button type="button" class="gwt-Button">Button 3</button>
  </div>
▼<table cellspacing="0" cellpadding="0">
  ▼<tbody>
    ▼<tr>
      ▶<td align="left" style="vertical-align: top;">…</td>
      </tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    </tbody>
  </table>
</body>
```

Popups

- *PopupPanel* and *DialogBox* can be extended for poups
- general design pattern:
  1. create one instance of a popup
  2. hide and show the one instance
  3. update the variable content of the popup

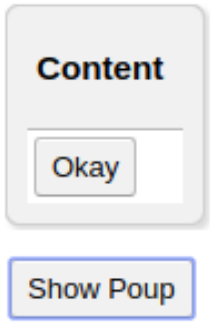
Popups

```
private class DialogPopup extends DialogBox {

 public DialogPopup(String htmLContent) {

        setHTML(htmLContent);
        Button ok = new Button("Okay");
        ok.addClickHandler(new ClickHandler() {

        @Override
        public void onClick(ClickEvent event) {
                hide();
        }
        });
        add(ok);
  }
}
```

Popups

```
final DialogPopup dlp = new DialogPopup("<h3>Content</h3>");
Button btnPopup = new Button("Show Poup");
btnPopup.addClickHandler(new ClickHandler() {

        @Override
        public void onClick(ClickEvent event) {
                dlp.show();
        }
});
RootPanel.get().add(btnPopup);
```

Popups



Exercise

- learn how to programmatically assemble GWT widgets to create a client–side frontend
- see handout

References

GWT books used: [Tacy et al., 2013], [Cooper and Collins, 2008], [Kereki, 2010]

excellent tutorial from which part of the structure of the presentation has been inspired: https://www.tutorialspoint.com/gwt/index.htm

Robert Cooper and Charlie Collins. *GWT in Practice*. Manning Publications Co., 2008.

Federico Kereki. *Essential GWT: building for the web with Google Web toolkit 2*. Pearson Education, 2010.

Adam Tacy, Robert Hanson, Jason Essington, and Anna Tokke. *GWT in Action*. Manning Publications Co., 2013.