BASIC SOY TEMPLATE SYNTAX

WHAT ARE SOY TEMPLATES?

- We talked about Soy Templates earlier in the course, but let's review.
- Soy Templates, also known as Google Closure Templates, are a client-side and server-side templating system that helps you dynamically build reusable HTML and UI elements.
- The syntax is simple, and you can customize it to fit your application's needs.
- Soy Templates are implemented for both JavaScript and Java, so you can use the same templates on both the server-side and the client-side.
- This means you can render pages on the server before serving to the client, benefiting the user experience on first load.

HOW DO SOY TEMPLATES RELATE TO METAL.JS?

- Soy Templates are one of the template types compatible with Metal.js.
  - Soy Template support, along with other template options, is still in its initial steps and will be continually improved in the future.
- Using Soy Templates and Metal.js together is simple and makes it easy to create solid, lightweight, and flexible UI components.
- A Soy Portlet is a Liferay Portlet that uses Soy Templates and Metal.js as its front-end.

WHAT ARE THE BENEFITS OF SOY PORTLETS?

- Using Soy Templates as the template of your portlet gives you all the benefits of using Metal.js.
- You get the advantage of a framework that's built from the ground up with performance in mind.
- It's a versatile build system that you can leverage in a number of different ways.
- Your application would be written with future-ready *ECMAScript 2015* code, making it clean and easy to read.

WHAT DOES A SOY PORTLET LOOK LIKE?

- The file structure of a Soy Portlet would look similar to a regular application.
- Let's look at an example we'll call `MySoyPortlet`:

```
- my-soy-portlet
  - .lfrbuild-portal
  - build.gradle
  - bnd.bnd
  - package.json
  - src/main/
    - java/com/liferay/frontend/my/soy/portlet/web/internal/portlet
      - MySoyPortlet.java
    - resources/META-INF/resources/
      - MyComponent.es.js
      - MyComponent.soy
      - MyComponent.scss
```

EXTENDING THE SOYPORTLET CLASS

- `MySoyPortlet` can extend from the `SoyPortlet` class like this:

```
public class MySoyPortlet extends SoyPortlet {
    @Override
    public void render(RenderRequest renderRequest, RenderResponse
    renderResponse) {
        //do things here
    }
}
```

SOY TEMPLATE MAGIC

- And here is what the Soy Template magic looks like:

```
{namespace MyComponent}

/**
 * This renders the main content of the `MyComponent` component.
 * @param? content
 */
{template .render}
    <div class="my-component">
        <div class="my-component-content">
            <button type="button" class="close" data-onclick="hide">
                <span>×</span>
            </button>
            <h4>{$content ?: ''}</h4>
        </div>
    </div>
{/template}
```

METAL.JS JAVASCRIPT CODE WITH THE COMPONENT

- And the Metal.js JavaScript code to go along with this component:

```
'use strict';
import templates from './MyComponent.soy';
import Component from 'metal-component';
import Soy from 'metal-soy';
class MyComponent extends Component {
    hide() {
        // All Metal.js components already have a `visible` state which sets t
        // main element's `display` to "none" when set to false.
        this.visible = false;
    }
}
// This line is declaring that `MyComponent` will be using the given
soy templates for
// rendering itself.
Soy.register(MyComponent, templates);

export default MyComponent;
```

SOY PORTLET EXAMPLES

- **Hello Soy Portlet:** A Hello World portlet built with Soy Templates:
  - *https://github.com/liferay/liferay-portal/tree/master/modules/apps/foundation/hello-soy/hello-soy-web*
- **Image Editor Portlet:** A portlet for editing images built with Soy Templates and Metal.js:
  - *https://github.com/liferay/liferay-portal/tree/master/modules/apps/foundation/frontend-image-editor/frontend-image-editor-web*

SOY TEMPLATE SYNTAX AND CONCEPTS

- The SoyPortlet provides the Soy Template file structure and Soy Template data.
- Let's take a closer look at the basic concepts and syntax.
  1. Template syntax such as:
     - Comments
     - Text
  2. Command
  3. Expression
  4. Functions

COMMENTS SYNTAX

- Comments in Soy Templates are similar to Java and JavaScript.
- If preceded by a white space, // begins a rest of line comment.
- Using /* */ delimits any text between /* and */ as a comment.

```
{template .soyComments}
  I Love Liferay<br>  // What a great comment
  /* you can use
  multiline comments */
  // URL syntax is not interpreted as comment.
  http://www.google.com<br>
{/template}
```

TEXT SYNTAX

- Any character in a template that does not appear between braces {} is raw text.
- The compiler does not parse raw text, except for joining lines and removing indentation.
- This allows developers to create nicely formatted templates with proper indentation and readability.

```
{template ..soyTextSyntax}
  // These two lines will be joined by adding a space.
  First
  second.<br>
  // When either HTML or Soy tag border the join location the
  //lines will be joined without adding a space.
  <i>First</i>
  second.<br>
  First
  {''}second.<br>
{/template}
```

TEXT SYNTAX: SPECIAL CHARACTERS COMMANDS

- Soy Templates also provide a number of commands to generate raw text.
- This includes special character commands.
  - `{sp}`: A space
  - `{nil}`: An empty string
  - `{\r}`: Carriage return
  - `{\n}`: New line or line feed
  - `{\t}`: Tab
  - `{lb}`: Left brace
  - `{rb}`: Right brace

TEXT SYNTAX: COMMANDS

- There are a few other commands that can be used to generate text.
- The `literal` command allows you to include a block of raw text, as no processing happens between literal blocks and output is rendered as it appears in the template.

```
{template .soyLiteral}
  // Note: Lines are not joined and indentation is not stripped
  Literal : {literal}AA BB { CC DD } EE {sp}{\n}{rb} FF{/literal}
  </pre>
{/template}
```

- The `print` command will output the results of an expression:

```
{template .soyPrint}
  {print 'Boo!'}<br>  // print a string
  {'Boo!'}<br>  // the command name 'print' is implied
  {1 + 2}<br>  // print the result of an expression
  {$boo}<br>  // print a data value
  {1 + $two}<br>  // print the result of an expression that uses a data value
{/template}
```

COMMANDS

- Beyond the special character and literal and print commands, there are many other commands available in Soy Templates.
- Commands are instructions provided to the compiler to create templates and add custom logic.
- Command syntax in Soy Templates is delimited by braces {}. The command name can be followed by additional tokens to form the command text.
- Let's look at a few common commands and their syntax.

COMMANDS: BASIC

- The {call} command can be used to call another template and return its output. You can also provide parameters to the call template.

```
{template .soyCaller}
  {call .soyCallee}
    {param firstName: $instructor.firstName /}
    {param lastName: $instructor.lastName /}
  {/call}
{/template}
{template .soyCallee}
  Hi $firstName{sp}$lastName, welcome to Liferay Training
{/template}
```

- Use the {let} command to define an intermediate value.

```
  {let $isAbeforeB: $aaa < $bbb /}
```

COMMANDS: CONTROL FLOW

- The {if} commands can be used for conditional output.

```
{if $instructor.name == 'Nick'}
  <h1>Dude rocks</h1>
{elseif $instructor.name == 'Jon'}
  <h1>Dude rocks more</h1>
{else}
  <h1>We all rock</h1>
{/if}
```

- Similarly, the {switch} command can be used for conditional output.

```
{switch $instructor.name}
  {case 'Nick'}
    <h1>Dude rocks</h1>
  {case 'Jon'}
    <h1>Dude rocks more</h1>
  {default}
    <h1>We all rock</h1>
{/switch}
```

COMMANDS: LOOPS

- The {foreach} command iterates a list.

```
{foreach $instructor in $instructors}
   <h1>{$instructor.name} rocks</h1>
{ifempty}
  <h1>No teachers let's party</h1>
{/foreach}
```

- The {for} command is used for a numerical loop.

```
{for $i in range($instructors.size)}
  $i little instructor went to the market.<br>
{/for}
```

- Note: The range function can take one, two, or three arguments to allow the loop to control initial value, limit, and increment.

FUNCTIONS

- Beyond the `range()` function in the `{for}` command, there are a number of other functions available.
  - `isFirst(var)`, `isLast(var)`, and `index(var)` functions can be used with the `{foreach}` command.
  - `isNonnull(value)` function will return true if the given value is both defined and not the value null.
  - There are functions that operate on a list or map such as `length(list)`, `keys(map)`, `augmentMap(baseMap, additionalMap)`, and `quoteKeysIfJs(mapLiteral)`.

FUNCTION OPERATORS

- There are also functions that provide operations on numbers and text.
  - `round(number)` and `round(number, numDigitsAfterDecimalPoint)` will round to an integer or to a significant digit.
  - `floor(number)` returns the floor of the number.
  - `ceiling(number)` returns the ceiling of the number.
  - `min(number, number)` returns the min of the two numbers.
  - `max(number, number)` returns the max of the two numbers.
  - `randomInt(rangeArg)` A random integer between `0` and the rangeArg
  - `strContains(str, subStr)` checks whether a string contains a substring.

REFERENCE

- You can find more references and samples at:
  1. *https://developers.google.com/closure/templates/docs/commands*
  2. *https://github.com/google/closure-templates/blob/master/examples/features.soy*
  3. Liferay Source: *liferay-portal/modules/apps/foundation/hello-soy/hello-soy-web/*

Notes: