

Instructor Notes:

Polymer

A powerful new platform for extending html and componentizing apps



Copyright © 2016 Capgemini. All rights reserved. No part of this publication shall be reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior written permission of Capgemini.

Capgemini considers information included in this document to be Confidential and Proprietary.

Instructor Notes:

Polymer

(v1.0)

June 29, 2016

Proprietary and Confidential

- 2 -



Instructor Notes:

Add instructor notes here.

Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
15/06/2016	1.0	Polymer v1.0	Karthik Muthukrishnan	

Instructor Notes:

Add instructor notes here.

Course Goals and Non Goals

➤ Course Goals

- Understand Web Components fundamentals
- Understanding the Polymer library fundamentals
- Building responsive web applications by componentizing app



➤ Course Non Goals

- Using Polymer in Node Environment
- Unit Testing
- Deploying Polymer web applications

Instructor Notes:

Add instructor notes here.

Pre-requisites

- HTML, JavaScript & CSS

Instructor Notes:

Add instructor notes here.

Intended Audience

- Web application developers



June 29, 2016

Proprietary and Confidential

+ 6 +



Instructor Notes:

Add instructor notes here.

Day Wise Schedule

➤ Day 1

- Lesson 1: Introduction to Web Components

➤ Day 2

- Lesson 2: Getting started with polymer
- Lesson 3: Data Binding

➤ Day 3

- Lesson 3: Data Binding(cont.)
- Lesson 4: Polymer Elements

Instructor Notes:

Add instructor notes here.

Table of Contents

- **Lesson 1: Introduction to Web Components**
 - Challenges faced in modern web development
 - Web Components - Introduction to the future web
 - Web Components – Technologies
 - HTML Templates
 - Using HTML Templates
 - Custom Elements
 - Building Custom Elements
 - Custom Element - Lifecycle callback methods
 - Custom Element DOM categories
 - Shadow DOM – Terminologies
 - <content> and <shadow> insertion points

Instructor Notes:

Add instructor notes here.

Table of Contents

- **Lesson 1: Introduction to Web Components(cont.)**
 - Event Re-Targeting
 - Shadow DOM – CSS and Styling
 - HTML imports
 - Web Components - Benefits

Instructor Notes:

Add instructor notes here.

Table of Contents

- **Lesson 2: Getting started with polymer**
 - Polymer Library Introduction
 - Polymer Official Website
 - Polymer Installation
 - Polymer Application – Folder Structure
 - Registering a custom element
 - Polymer Library Introduction
 - Polymer Official Website
 - Polymer Installation
 - Polymer Application – Folder Structure
 - Registering a custom element
 - Creating Custom Element

Instructor Notes:

Add instructor notes here.

Table of Contents

- **Lesson 2: Getting started with polymer (cont.)**
 - Plunker and Polygit
 - Local DOM
 - Local DOM template
 - Shady DOM
 - Declared Properties
 - Usage of Declared Properties
 - Lifecycle callbacks
 - DOM API
 - Behaviors
 - Extending Native Elements
 - Event handling
 - CSS and Styling

June 29, 2016

Proprietary and Confidential

- 11 -



Instructor Notes:

Add instructor notes here.

Table of Contents

- **Lesson 3: Data binding**
 - Databinding in Polymer
 - Binding to text content
 - Compound bindings
 - Binding to objects
 - Binding to array items

Instructor Notes:

Add instructor notes here.

Table of Contents

- **Lesson 4: Polymer Elements**
 - Introduction to Polymer Elements
 - Polymer Element Catalog
 - Installing Elements
 - Using Elements
 - Paper Elements
 - Iron Elements
 - Neon Elements
 - Gold Elements
 - App Elements
 - Molecules
 - [customelements.io](#)

June 29, 2016

Proprietary and Confidential

+ 13 +



Instructor Notes:

Add instructor notes here.

References

- www.html5rocks.com
- <https://www.polymer-project.org/1.0/>



Instructor Notes:

Add instructor notes here.

Other Parallel Technology Areas

- **X-TAG**: X-Tag is a small JavaScript library, created and supported by Mozilla
- **BOSONIC**: To build Web Components to support not-so-modern browsers like IE9.
- **SKATEJS**: SkateJS is a superset of the web component specs

Instructor Notes:

Add instructor notes here.

Polymer

[Lesson 01: Introduction to Web Components](#)

Instructor Notes:

Add instructor notes here.

Lesson Objectives

- At the end of this module you will be able to:
 - Understand the importance of Web components for the future web
 - Create and use HTML Templates
 - Build Custom Elements
 - Encapsulate Markup and Styles using Shadow DOM
 - Bundle HTML, JS and CSS in a single file and import it using HTML5 imports



2

Instructor Notes:

Add instructor notes here.

Challenges faced in modern web development



Undescriptive Markup

It's hard to read and navigate `<div>` hierarchy. Div tags are generic containers that doesn't convey the nature of the data inside. Using HTML 5 Semantics may helps us to convey the meaning of the content.

Style and JS Conflicts

Need to write complex, prefixed CSS selectors to avoid conflicts. !important need to be used to force styles, but still there is not guarantee that another style won't conflict.

```
#splId {  
    color:red;  
}  
.splClass{  
    color:blue!important; /*will have high importance than id selector*/  
}  
<div id="splId" class="splClass">Hi</div>
```

No markup encapsulation i.e. JavaScript can also be accidentally manipulate as well.

No Native Templates

There is no natural way to store HTML for templates. Approaches followed now create confusion, lead to XSS vulnerabilities, or aren't truly inert. For instance handlebars deliver a template to the browser by including `type="text/x-handlebars-template"` it in a `<script>` tag.

No Bundling

There's no simple way to import a component with a single line of code. Need to reference the necessary JS and CSS in the right order and add proprietary markup.

No Component Standard

Switching between today's proprietary component solutions is painful. jQueryUI, KendoUI, Angular, etc. all have their own style.

Instructor Notes:

Add instructor notes here.

Web Components - Introduction to the future web

- W3C aims to address the challenges faced in modern web development by introducing Web Components.
- It is a collection of specifications that enable web developers to create web applications as a set of reusable components.
- It dramatically redefine the way web apps are built. It allows us to create rich, powerful, and reusable components using the well known technologies HTML5, JavaScript, and CSS.
- Each web component lives in its self-defined encapsulated unit with corresponding style and behavior logic.
- These components not only to be shared across a single web application; it can be also distributed on the web for others to use.

4

Instructor Notes:

Add instructor notes here.

Web Components - Technologies



HTML Templates : *Inert, reusable markup*

Custom Elements : *Defining own elements*

Shadow DOM : *Encapsulated markup & Styling*

HTML Imports : *Bundle HTML, JS & CSS*

5

Web Components consists of four technologies : Custom Elements, HTML Templates, Shadow DOM and HTML Imports.

Custom Elements : Enable web developers to create their own elements that are relevant to their design as part of the DOM structure with the ability to style/script them like any other HTML tag.

HTML Templates : Allows to define fragments of markup that stay consistent across web pages with the ability to inject dynamic content using JavaScript.

Shadow DOM : Abstract all the complexities from the markup by defining functional boundaries between the DOM tree and the subtrees hidden behind a shadow root.

HTML Imports : Similar to importing CSS files; It allows to include and reuse HTML documents in other HTML documents.

Each of these pieces is useful individually. But when combined as a whole package gives:

Composability : Being able to create whole sites and apps by putting different elements together.

Encapsulation : Isolating markup, style, and behavior logic so they don't leak into the rest of the page.

Reusability : Extending existing elements to create new elements, allowing you to stop reinventing the wheel.

Instructor Notes:

Add instructor notes here.

HTML Templates

- The concept of templating is not new to web development, frameworks use different techniques to share the markup for rendering presentation layers like given below :
- **HTML in Script Tags :** Templating engines like handlebars used to store markup as string not as DOM elements, which can be easily injected using innerHTML, but it can lead to XSS (Cross Side Scripting) vulnerability.
- **Hidden DOM Elements :** Stuffing templates in a hidden DOM elements, since it is DOM element it can be easily cloned but everything inside still runs which leads to 404's, scripts can be executed etc.
- The new HTML template element <template> is a mechanism for holding client-side content that is not to be rendered when a page is loaded but may subsequently be instantiated during runtime using JavaScript.

Handlebars delivers a template to the browser by including it in a <script> tag.

```
<script id="entry-template" type="text/x-handlebars-template">
<div class="entry">
<h1>{{title}}</h1>
<div class="body">
{{body}}
</div>
</div>
</script>
```

Template Characteristics :

Inert: Does nothing until cloned

Hidden from Selectors: We cannot locate the elements placed within the template tag using document.getElementById

Flexible placement : Template tag can be placed anywhere on the page <head>, <body> etc.

```
<template>
<p id="para">Paragraph element inside the template</p>
<script> alert("Won't be executed"); </script>
</template>
```

Instructor Notes:

Add instructor notes here.

Using HTML Templates

Get reference to the template

Use `document.importNode` to clone the template's content

Append element to page

Change the target element with in the template, if required

Instructor Notes:

Add instructor notes here.

Demo

- Using-HTML-Template



5

Instructor Notes:

Add instructor notes here.

Custom Elements

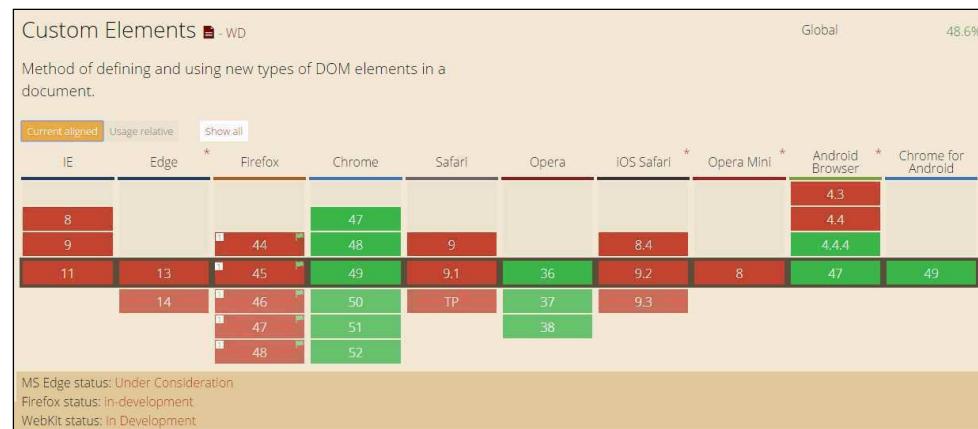
- Custom Elements allow web developers to define new types of HTML elements.
- It provides the following features
 - Defining new HTML/DOM elements
 - Create elements that extend from other elements
 - Logically bundle together custom functionality into a single tag
 - Extend the API of existing DOM elements
- Custom Elements must be named with a dash (-) by prefixing with a unique namespace to avoid naming conflicts.
 - **Valid Custom Elements:** cg-login-form, capgemini-login-form, cg-submit-button
 - **Invalid Custom Elements:** mycomponent, my_component, myComponent, MyComponent

Custom Elements enable developers to create their own custom HTML tags and allow them to use those tags in their sites and apps, and enable easier component reuse.

Naming rules :

It should have at least one '-' inside the name of a custom element. Before the dash (-) prefix custom element with Company Name, Application Name, Brand Name which makes the developer to understand the usage of Custom element and avoid naming conflicts.

Supported browsers : At present, custom Elements are supported by Chrome, Firefox and Android browsers. For polyfilling other browsers, we can use **webcomponents.js**



Source : <http://caniuse.com/>

Instructor Notes:

Add instructor notes here.

Building Custom Element

- Custom elements are created using `document.registerElement()`. By default, custom elements inherit from `HTMLElement`, we can inherit from other HTML elements also. For instance : `Object.create(HTMLButtonElement.prototype)`

```
var CgTab = Object.create(HTMLElement.prototype); //Create a prototype
//Add Js properties and Methods here...
document.registerElement('cg-tab'); // Register the new element
document.body.appendChild(new CgTab()); // Add to DOM
```

- Custom Element can be Instantiated using Markup, new Operator, createElement & innerHTML.
- Instantiating type extension-style custom elements is strikingly close to custom tags.

10

Creating Custom element and adding JS methods and Properties to it.

```
var XFooProto = Object.create(HTMLElement.prototype);
// 1. Give x-foo a foo() method.
XFooProto.foo = function() { alert('foo() called'); }

// 2. Define a property read-only "bar".
Object.defineProperty(XFooProto, "bar", {value: 5});

// 3. Register x-foo's definition.
var XFoo = document.registerElement('x-foo', {prototype: XFooProto});

// 4. Instantiate an x-foo.
var xfoo = document.createElement('x-foo');

// 5. Add it to the page.
document.body.appendChild(xfoo);
```

Instantiating Custom Element

- Markup :** `<x-foo>`
- new Operator :** `var el = new Xfoo();`
- createElement :** `var el = document.createElement('x-foo');`
- innerHTML :** `el.innerHTML = '<x-foo />'`

Instantiating Extended Element

- Markup :** `<button is="spl-button"/>`
- JavaScript :** `var button = document.createElement('button','spl-button');`
- New Operator :** `document.body.appendChild(new SplButton());`

Instructor Notes:

Add instructor notes here.

Custom Element - Lifecycle callback methods

- Functions can be defined to be called when certain events happened on Custom Elements which are called as "lifecycle callbacks".

Method	Called When
createdCallback	an instance of the element is created
attachedCallback	an instance was inserted into the document
detachedCallback	an instance was removed from the document
attributeChangedCallback	Attributes are added, removed and updated

11

Instructor Notes:

Add instructor notes here.

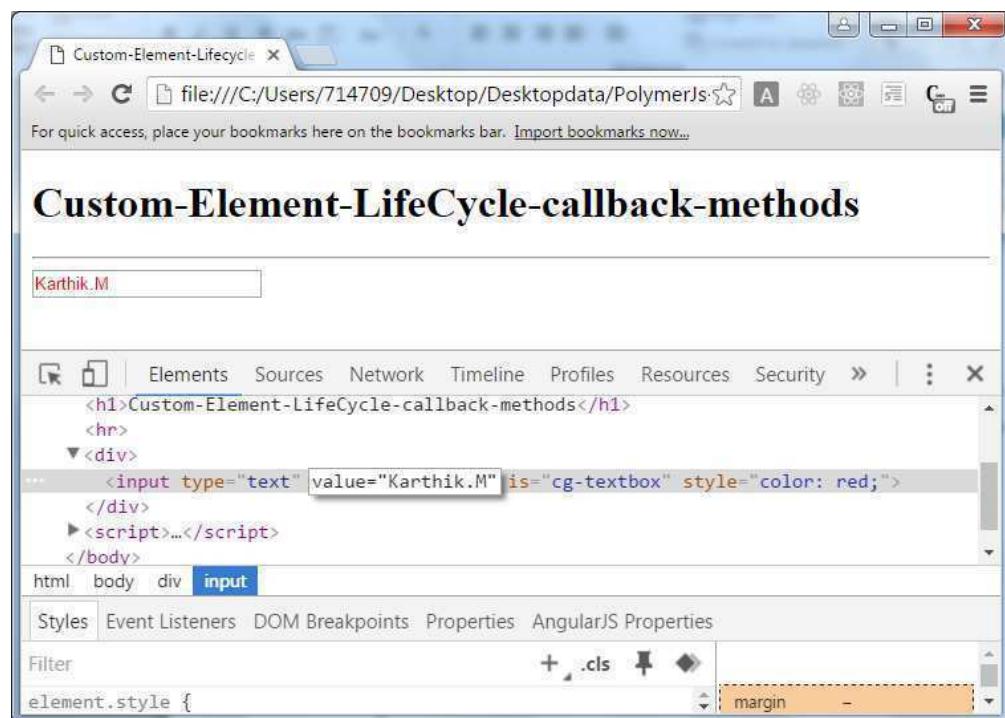
Demo

- Using-Custom-Element
- Custom-Element-Lifecycle-callback-methods



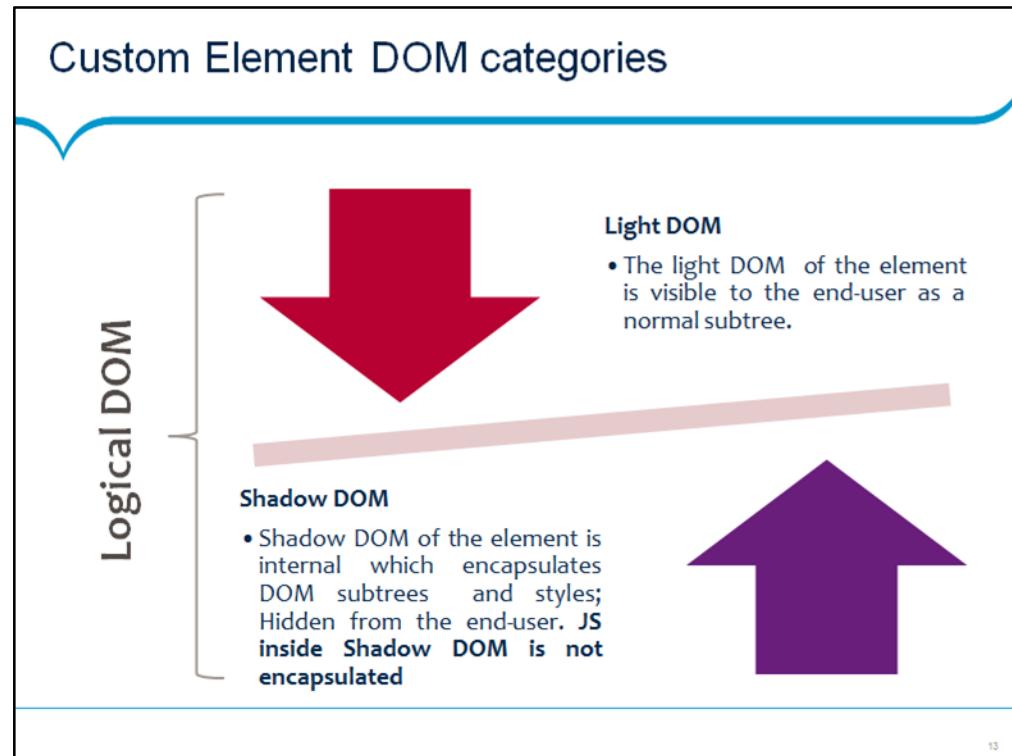
12

Changing the value for the value attribute using Chrome Developer Tool



Instructor Notes:

Add instructor notes here.

**Light DOM**

Custom element supplies the light DOM. The light DOM of `<my-custom-element>` is visible to the end-user of the element as a normal subtree. They can access `.childNodes`, `.children`, `.innerHTML`, or any other property or method that gives information about a node's subtree.

```
<my-custom-element>
  <q>Hello World</q> <!-- part of my-custom-element's light DOM -->
</my-custom-element>
```

Shadow DOM

`<my-custom-element>` may define shadow DOM by attaching a shadow root to itself. Shadow DOM is internal to the element and hidden from the end-user. Its nodes are not children of `<my-custom-element>`

```
#shadow-root
<!-- everything in here is my-custom-element's shadow DOM -->
<span>People say: <content></content></span>
<footer>sometimes</footer>
```

Together, the light DOM and shadow DOM are referred to as the logical DOM. This is the DOM that the developer interacts with. The composed DOM is what the browser sees and uses to render the pixels on the screen.

Instructor Notes:

Add instructor notes here.

Shadow DOM - Terminologies

▪ Shadow DOM

- Shadow DOM addresses the lack of true DOM tree encapsulation when building components. It allows multiple DOM subtrees to be composed into one larger tree.

▪ Shadow Host

- DOM element which is hosting the Shadow DOM subtree or it is the DOM node which contains the Shadow Root.

▪ Shadow Root

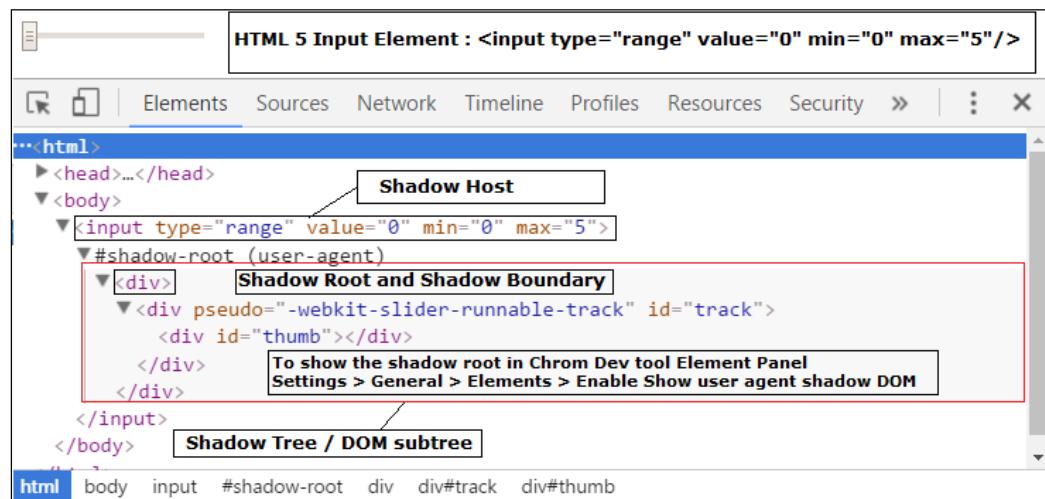
- Root of the DOM subtree containing the shadow DOM nodes. It is a special node, which creates the boundary between the normal DOM nodes and the Shadow DOM nodes. It is this boundary, which encapsulates the Shadow DOM nodes from any HTML or CSS code on the consuming page.

Shadow Boundary :

A Barrier which separates Light DOM from the Shadow DOM. It Encapsulates DOM sub tree. It enhances the readability and avoids accidental styling.

CSS Selectors inside a shadow root only match / style the Shadow DOM.

JavaScript in the Shadow DOM is not encapsulated. We need to use module patterns if encapsulation required.



Instructor Notes:

Add instructor notes here.

<content> and <shadow> insertion points

- <content></content> : Serves as a placeholder for the host content in shadow DOM.
- <shadow></shadow> : They are hosts for other shadow trees.

```
<div id="shadowHost">Light DOM</div>
<script>
var container = document.querySelector('# shadowHost ');
var root1 = container.createShadowRoot();
var root2 = container.createShadowRoot();
root1.innerHTML = '<div>Root 1</div><content></content>';
root2.innerHTML = '<div>Root 2</div><shadow></shadow>';
</script>
```

Output :

```
ROOT 2
ROOT1
Light DOM
```

15

Greedy Insertion Points : Below mentioned <content> placeholders are considered a wildcard selection and it will grab any content in the shadow host that is left over

```
<content></content>
<content select=""></content>
<content select="*"></content>
```

Shadow trees added to a host are stacked in the order they're added, starting with the most recent first. The last one added is the one that renders.

If multiple <shadow> insertion points exist in a shadow tree, only the first is recognized. The rest are ignored.

Instructor Notes:

Add instructor notes here.

Event Re-Targeting

- If an event originates from one of the nodes in Shadow DOM crosses the Shadow Boundary, then it is re-targeted to refer to the Shadow Host in order to maintain encapsulation.
- Some events cross the shadow boundary and some do not. The following events never cross the shadow boundary : *abort, error, select, change, load, reset, resize, scroll, selectstart*

Instructor Notes:

Add instructor notes here.

Shadow DOM – CSS and Styling

- CSS styles defined inside Shadow DOM are scoped to the ShadowRoot.
- Following selectors can be used to apply CSS and styling to shadow DOM.
 - `:host` : Selects a shadow host element. May contain additional identifiers in parenthesis. It has low specificity, i.e. Most specific selectors in the Light DOM may override.
 - `:host-context` : Selects a shadow host based on a matching parent element.
 - `::content` : Selects distributed nodes inside the element
- We can expose a custom property (a.k.a. CSS variable) to be styled from the outside.

17

CSS Specificity Rules:

Traditional CSS specificity rules apply for `:host`

From most to least specific:

- | | |
|-----------------------------------|--|
| 1. Inline Style | <code><p style="color:red"></code> |
| 2. ID | <code>#target{color:red}</code> |
| 3. Class, pseudo-class, attribute | <code>.spl{ color : red}</code> |
| 4. Elements | <code>p {color: red}</code> |

Instructor Notes:

Add instructor notes here.

Demo

- ShadowDOM
- InsertionPoints
- Event-Re-Targeting
- ShadowDOM-Styling



15

Instructor Notes:

Add instructor notes here.

HTML imports

- HTML imports provides an easy way to include HTML documents in other HTML documents.
- HTML imports can also include CSS and JavaScript which makes imports a fantastic tool for loading related HTML/CSS/JS.

<!-- It must be inside head tag. HTML is initially inert. JS and CSS run and apply immediately. Imports from separate domain must support CORS -->

```
<link rel="import" href="source.html">
```

- `document.currentScript.ownerDocument` is used to references the imported document
- Using Sub-imports import can include another one, which helps in reusing or extending another component.

19

HTML Imports allow bundling HTML/CSS/JS as a single resource

An import's mimetype is `text/html`.

Resources from other origins need to be CORS-enabled.

Imports from the same URL are retrieved and parsed once. That means script in an import is only executed the first time the import is seen.

Scripts in an import are processed in order, but do not block the main document parsing.

Instructor Notes:

Add instructor notes here.

Demo

- HTML-Imports

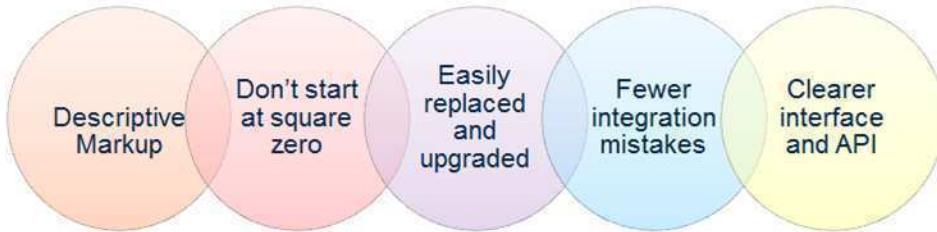


20

Instructor Notes:

Add instructor notes here.

Web Components - Benefits



Descriptive markup : Web components makes the markup easier to read, navigate and understand. It helps to replace the generic tags like div and span tags with higher level and more descriptive markup, which can be navigate and maintained easily.

Don't start at square zero : No need to start from the beginning on each new project, we can easily leverage the components created in the previous projects or use the components created by the community.

Easily replaced and upgraded : Components can be easily replaced and upgraded when bugs have been found; since it is properly bundled, encapsulated and centralized.

Fewer Integration mistakes : HTML imports makes it very easy to import bundle of dependencies, which leads to have less integration problems in projects. Just we need to add single line at the top of file and get everything we need to run the component.

Clearer interface and API : API for interacting with the components will be much clear than the current frameworks. Components will feel native , which greatly reduces the learning curve when using the new components created by other developers.

21

Instructor Notes:

Add instructor notes here.

Summary

- Web components are a collection of specifications that enable developers to create their web applications as a set of reusable components
- HTML Templates provides a native way to declare templates. All the contents inside does nothing until activated via cloning.
- Custom Elements lets us to create rich, reusable and highly descriptive markup by defining our own html element.
- Shadow DOM provides encapsulation for DOM and styles but JS is not encapsulated.
- Like <template> tag any HTML in HTML5 import is inert initially.
- HTML imports bundle up HTML, JS, CSS and deliver the components.
- webcomponents.js is a set of polyfills built on top of the Web Components specifications. It makes it possible for developers to use these standards today across all modern browsers.



22

Instructor Notes:

Add instructor notes here.

Polymer

[Lesson 02: Getting started with polymer](#)

Instructor Notes:

Add instructor notes here.

Lesson Objectives

- At the end of this module you will be able to:
 - Understand the basic concepts of polymer.js
 - Create custom element
 - Explain the concept of Local DOM, Shadow DOM and Shady DOM
 - Create Local DOM using <dom-module> and the <template> element with in it.
 - Create properties for custom element.
 - Understand Polymer components Lifecycle.
 - Use Polymer DOM API
 - Create and use Behaviors
 - Understand Event handling In Polymer
 - Style Polymer components using CSS



2

Instructor Notes:

Add instructor notes here.

Polymer Library Introduction

- Web components provide a new standard for building web UI components, and Polymer makes it easy to build reusable components for your web applications.
- Polymer is a new type of library for the web, built on the top of web components, and designed to leverage the evolving web platform on modern browsers.

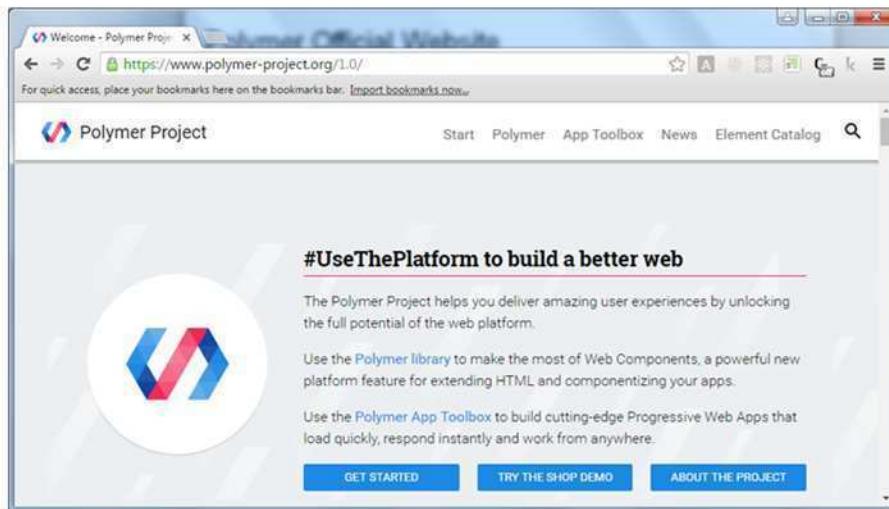


3

Instructor Notes:

Add instructor notes here.

Polymer Official Website



Instructor Notes:

Add instructor notes here.

Polymer Installation

- Polymer library can be installed in two ways
 - Installing with Bower (Requires node setup)
 - Installing from ZIP files
- **Installing from Zip files :** When Polymer downloaded as a ZIP file, we will get all of the dependencies bundled into a single archive.
- It's a great way to get started there is no need to install any additional tools.

Installing from ZIP files

Click the button to download Polymer 1.0 as a ZIP file.

[DOWNLOAD ZIP](#)

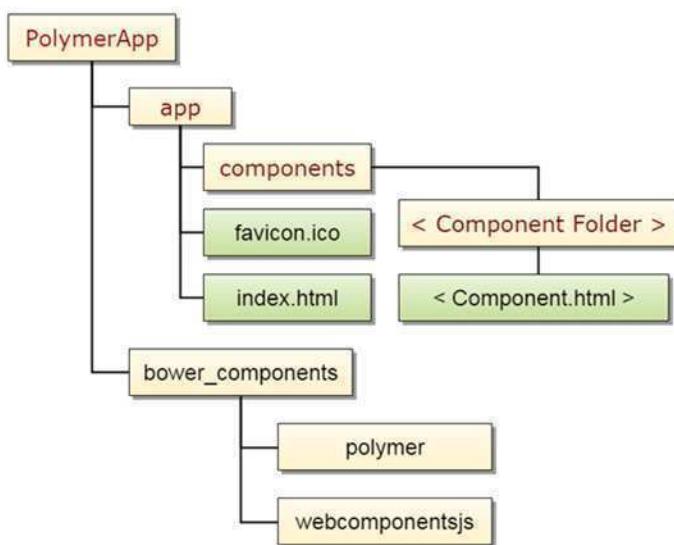
Expand the ZIP file in your project directory to create a `bower_components` folder.
▼ my-project
 ▼ bower_components
 ➤ polymer
 ➤ webcomponentsjs

5

Instructor Notes:

Add instructor notes here.

Polymer Application – Folder Structure



components folder hold our custom components

5

Instructor Notes:

Add instructor notes here.

Registering a custom element

- To register a custom element, use the Polymer function, and pass in the prototype for the new element.
- The prototype must have an *is* property that specifies the HTML tag name for custom element.
- By specification, the custom element's name must contain a dash (-).

```
// To register a custom element
var MyElement = Polymer({
  is: 'my-element',
});
// create an instance with createElement:
var el1 = document.createElement('my-element');

// ... or with the constructor:
var el2 = new MyElement();
```

7

Instructor Notes:

Add instructor notes here.

Creating Custom Element

```
PolymerApp\app\index.html
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <title>First Polymer element</title>
    <script src="../bower_components/webcomponentsjs/webcomponents-lite.js"></script>
    <link rel="import" href="components/cg-element/cg-element.html"/>
  </head>
  <body>
    <cg-element></cg-element>
  </body>
</html>

PolymerApp\app\components\cg-element\cg-element.html
<link rel="import" href="../../bower_components/polymer/polymer.html"/>
<dom-module id="cg-element">
  <template>
    My First Polymer Element
  </template>
</dom-module>

<script>
  Polymer({
    is:"cg-element"
  });
</script>
```

5

To run the program in Internet Explorer

- <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
- IE=edge means IE should use the latest (edge) version of its rendering engine
 - chrome=1 means IE should use the Chrome rendering engine if installed

Polyfill for the old browsers which doesn't support rel="import" attribute

<script src="../bower_components/webcomponentsjs/webcomponents-lite.js"></script>

Including the element declaration

<link rel="import" href="components/cg-element/cg-element.html"/>

Adding custom element

<cg-element></cg-element>

importing polymer

<link rel="import" href="../../bower_components/polymer/polymer.html"/>

dom module element is used to declare the element

<dom-module id="cg-element">

details in the template tag will be used as a content for the element

<template> My First Polymer Element </template>
</dom-module>

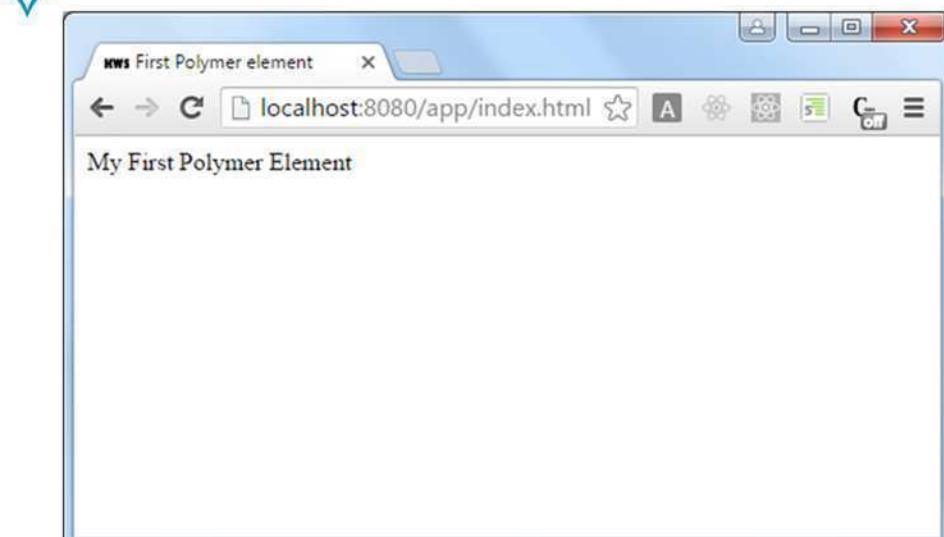
<script>

Adding the element to the polymer scope which will parse the dom module
Polymer({ is:"cg-element" });
</script>

Instructor Notes:

Add instructor notes here.

Creating Custom Element

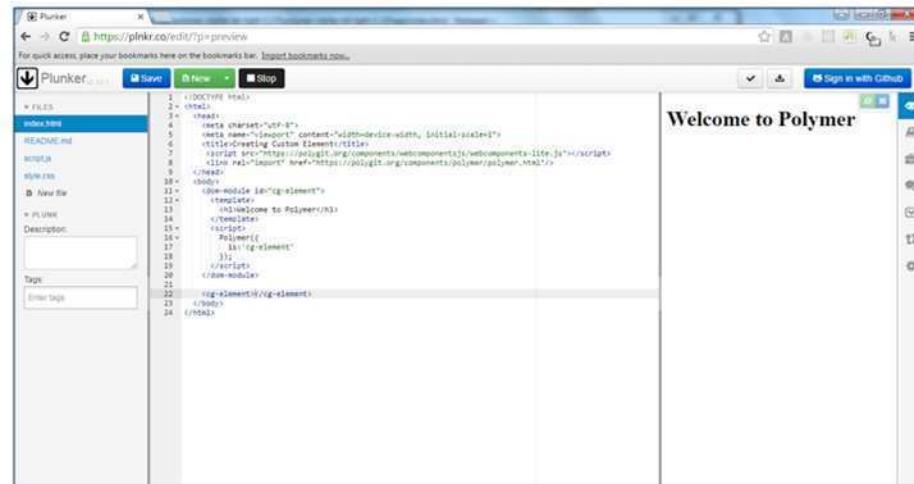


19

Instructor Notes:

Add instructor notes here.

Plunker and Polygit



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
    <title>Creating Custom Element</title>
    <script src="https://polygit.org/components/webcomponentsjs/webcomponents-lite.js"></script>
    <link rel="import" href="https://polygit.org/components/polymer/polymer.html"/>
  </head>
  <body>
    <dom-module id="cg-element">
      <template>
        <h1>Welcome to Polymer</h1>
      </template>
      <script>
        Polymer({
          is:'cg-element'
        });
      </script>
    </dom-module>
    <cg-element></cg-element>
  </body>
</html>
```

Instructor Notes:

Add instructor notes here.

Demo

- Creating-Custom-Element



11

Instructor Notes:

Add instructor notes here.

Local DOM

- The DOM that an element creates and manages is called its local DOM.
- Polymer supports multiple local DOM implementations.
- In future browsers which support shadow DOM, will create the local DOM.
- Currently Polymer provides local DOM via a custom implementation called shady DOM which is inspired by shadow DOM.
- Shady DOM requires Polymer DOM API when manipulating DOM from JavaScript.
- Polymer DOM API covers most of the common DOM methods and properties, and is compatible with both shady DOM and native shadow DOM.
- Currently Polymer uses shady DOM by default on all browsers which can be configured to use shadow DOM.

12

Document-level global Polymer settings can be set by creating a Polymer object on window before importing the Polymer library

```
<html>
<head>
  <meta charset="utf-8">
  <script src="components/webcomponentsjs/webcomponents-lite.js"></script>
  <script>
    /* this script must run before Polymer is imported */
    window.Polymer = {
      dom: 'shadow', //shady (current default)
      lazyRegister: true
    };
  </script>
  <!-- import a component that relies on Polymer -->
  <link rel="import" href="elements/my-app.html">
</head>
<body>

  ...

```

Settings can also be switched on the URL query string: <http://example.com/test-app/index.html?dom=shadow>

shady: All local DOM is rendered using shady DOM, even where shadow DOM is supported (current default).

shadow: Local DOM is rendered using shadow DOM where supported (this will be the default in the future).

Instructor Notes:

Add instructor notes here.

Local DOM template

- <dom-module> element is used to specify DOM to be used for an element's local DOM.
 - ***id attribute***: matches its element's specified in the <script> is property
 - <template> : Polymer will automatically clone the <template> tag contents into the element's local DOM.
- Defining an element has an imperative and declarative portion. The imperative portion is the call to Polymer({...}), and the declarative portion is the <dom-module> element.

```
<dom-module id="x-foo">
  <template>I am x-foo!</template>
  <script>
    Polymer({
      is: 'x-foo'
    });
  </script>
</dom-module>
```

13

The imperative and declarative portions of an element's definition may be placed in the same html file or in separate files.

The <script> tag can be inside or outside of the <dom-module> element.

The element's template must be parsed before the call to Polymer.

The children specified within the element is called as **Light DOM**

```
<dom-module id="x-foo">
  <template>I am x-foo!</template>
  <script>
    Polymer({
      is: 'x-foo'
    });
  </script>
</dom-module>

<body>
  <x-foo>
    <p>Child element placed inside element</p><!-- Light DOM -->
  </x-foo>
</body>
```

Instructor Notes:

Add instructor notes here.

Shady DOM

- Web components require tree-scoping for proper encapsulation.
- Shadow DOM is the standard that implements tree-scoping, but it's not yet universally implemented.
- Polyfilling shadow DOM is hard, the robust polyfill is invasive and slow.
- Shady DOM is a super-fast shim for shadow DOM that provides tree-scoping, but has drawbacks. Most importantly, one must use the shady DOM APIs to work with scoped trees.
- Shady DOM provides enough tree-scoping for Polymer to act as if shadow DOM is available on all platforms, without compromising performance.
- Shady DOM API can optionally employ real shadow DOM where it's available.

14

Many projects cannot afford the Shadow DOM Polyfill, for the reasons given below.

- It's a lot of code.
- It's slow to indirect all the DOM API.
- Structures like NodeList can simply not be emulated.
- There are certain accessors that cannot be overwritten (for example, window.document, window.document.body).
- The polyfill returns objects that are not actually Nodes, but Node proxies, which can be very confusing.

```
> $0
< cg-element>
    My First Polymer Element
</cg-element>
> $0.textContent
< "My First Polymer Element"
" 
> Polymer.dom($0).textContent
< ""
```

Instructor Notes:

Add instructor notes here.

Demo

- Working-with-LocalDOM



15

Instructor Notes:

Add instructor notes here.

Declared Properties

- Properties can be declared on custom elements by adding them to the properties object on prototype.
- Adding a property to the properties object allows a user to configure the property from markup.
- A Property value can be bind with in the element's Local DOM using {{ }} or [[]]

```
Polymer({  
  is: 'x-custom',  
  properties: {  
    /*property keys and values*/  
  }  
});
```

16

Using properties we can define properties in element.

Instructor Notes:

Add instructor notes here.

Usage of Declared Properties

- With properties object we can specify :
 - Property type : Boolean, Date, Number, String, Array or Object
 - Default value : It can be Boolean, Number, String or function's return value.
 - Property change observer : Calls a method whenever the property value changes.
 - Read-only status : Prevents accidental changes to the property value.
 - Two-way data binding support : Fires an event whenever the property value changes.
 - Computed property : Dynamically calculates a value based on other properties.
 - Property reflection to attribute : Updates the corresponding attribute value when the property value changes.

17

The properties object supports the following keys for each property:

Key	Value
type	Used for deserializing from an attribute. It can be Boolean, Date, Number, String, Array or Object
value	Sets the Default value for the property(boolean, number, string or function). If value is a function, the function is invoked and the return value is used as the default value of the property. If the default value should be an array or object unique to the instance, create the array or object inside a function.
reflectToAttribute	Set to true to cause the corresponding attribute to be set on the host node when the property value changes. If the property value is Boolean, the attribute is created as a standard HTML boolean attribute
readOnly	If true, the property can't be set directly by assignment or data binding
notify	If true, the property is available for two-way data binding. In addition, an event, property-name-changed is fired whenever the property changes.
computed	The value is interpreted as a method name and argument list. The method is invoked to calculate the value whenever any of the argument values changes. Computed properties are always read-only.
observer	The value is interpreted as a method name to be invoked when the property value changes

Instructor Notes:

Add instructor notes here.

Demo

- Working-with-properties



15

Instructor Notes:

Add instructor notes here.

Lifecycle callbacks

- Polymer's Base prototype implements the standard Custom Element lifecycle callbacks to perform tasks necessary for Polymer's built-in features.
- Polymer adds an extra callback, ready, which is invoked when Polymer has finished creating and initializing the element's local DOM.

Method	Called When
created	an instance of the element is created
attached	an instance was inserted into the document
detached	an instance was removed from the document
attributeChanged	Attributes are added, removed and updated
ready	Use for one-time configuration of component after local DOM is initialized

19

The properties object supports the following keys for each property:

Key	Value
type	Used for deserializing from an attribute. It can be Boolean, Date, Number, String, Array or Object
value	Sets the Default value for the property(boolean, number, string or function). If value is a function, the function is invoked and the return value is used as the default value of the property. If the default value should be an array or object unique to the instance, create the array or object inside a function.
reflectToAttribute	Set to true to cause the corresponding attribute to be set on the host node when the property value changes. If the property value is Boolean, the attribute is created as a standard HTML boolean attribute
readOnly	If true, the property can't be set directly by assignment or data binding
notify	If true, the property is available for two-way data binding. In addition, an event, property-name-changed is fired whenever the property changes.
computed	The value is interpreted as a method name and argument list. The method is invoked to calculate the value whenever any of the argument values changes. Computed properties are always read-only.
observer	The value is interpreted as a method name to be invoked when the property value changes

Instructor Notes:

Add instructor notes here.

Demo

- Lifecycle callbacks



20

Instructor Notes:

Add instructor notes here.

DOM API

- Polymer provides a custom API for manipulating DOM such that local DOM and light DOM trees are properly maintained.
- All DOM manipulation must use this API, as opposed to DOM API directly on nodes.
- These methods and properties have the same signatures as their standard DOM equivalents, with the following exceptions:
 - Properties and methods that return a list of nodes return an Array, not a NodeList.
 - Use the root property to access a Polymer element's local DOM root.
 - Insert, append, and remove operations are transacted lazily in certain cases for performance. In order to interrogate the DOM (for example, offsetHeight, getComputedStyle, etc.) immediately after one of these operations, call `Polymer.dom.flush()` first.

21

Use `Polymer.dom(this)` to access the light dom and `Polymer.dom(this.root)` to access element's local dom.

The following methods and properties are provided.

Adding and removing children:

```
Polymer.dom(parent).appendChild(node)  
Polymer.dom(parent).insertBefore(node, beforeNode)  
Polymer.dom(parent).removeChild(node)  
Polymer.dom.flush()
```

Calling `append`/`insertBefore` adds the node to parent's light DOM. In order to insert/append into the local DOM of a custom element, use a node in the local DOM as a parent (or `this.root`, which is the root of the local DOM).

Parent and child APIs:

```
Polymer.dom(parent).childNodes  
Polymer.dom(parent).children  
Polymer.dom(node).parentNode  
Polymer.dom(node).firstChild  
Polymer.dom(node).lastChild  
Polymer.dom(node).firstElementChild  
Polymer.dom(node).lastElementChild  
Polymer.dom(node).previousSibling  
Polymer.dom(node).nextSibling  
Polymer.dom(node).textContent  
Polymer.dom(node).innerHTML
```

Instructor Notes:

Add instructor notes here.

DOM API

- To support composition of an element's light DOM with its local DOM, Polymer supports the <content> element.
 - The <content> element provides an insertion point at which an element's light DOM is combined with its local DOM.
 - The <content> element supports a select attribute which filters nodes via a simple selector.
- Every Polymer element has a this.root property which is the root of its local DOM tree.
- Nodes in the Local DOM using querySelector, querySelectorAll, or the \$\$ utility method.
- Any node specified in the element's template with an id is stored on the `this.$`

22

Query selector:

```
Polymer.dom(parent).querySelector(selector)  
Polymer.dom(parent).querySelectorAll(selector)
```

Content APIs:

```
Polymer.dom(contentElement).getDistributedNodes()  
Polymer.dom(node).getDestinationInsertionPoints()
```

Node mutation APIs:

```
Polymer.dom(node).setAttribute(attribute,value)  
Polymer.dom(node).removeAttribute(attribute)  
Polymer.dom(node).classList
```

Instructor Notes:

Add instructor notes here.

DOM API

- **dom-repeat**: `dom-repeat` accepts an `items` property, and one instance of the template is stamped for each item into the DOM at the location of the `dom-repeat` element.
 - The `item` property will be set on each instance's binding scope, thus templates should bind to sub-properties of `item`.
 - Sorting and Filtering can be applied.
- **dom-if**: used to stamps the template if the property is true.
- **dom-bind** : Polymer data binding is only available in templates that are managed by Polymer. i.e. data binding works inside an element's local DOM template, but not for elements placed in the main document.
 - `dom-bind` element is used to perform polymer bindings without defining a new `customElement`

23

Instructor Notes:

Add instructor notes here.

Demo

- Polymer-DOM-API
- Polymer-Api
- InsertionPoints



24

Instructor Notes:

Add instructor notes here.

Behaviors

- Polymer supports extending custom element prototypes with shared code modules called behaviors. It is used to share the functionalities between two elements.
- A behavior is an object that looks similar to a typical Polymer prototype. A behavior can define lifecycle callbacks, declared properties, default attributes, observers, and listeners.
- Behavior can be extended from another Behavior.
- To add a behavior to a Polymer element definition, include it in a behaviors array on the prototype.

```
Polymer({  
  is: 'super-element',  
  behaviors: [SuperBehavior]  
});
```

25

For lifecycle events, the lifecycle callback is called for each behavior in the order given in the behaviors array, followed by the callback on the prototype.

Any non-lifecycle functions on the behavior object are mixed into the base prototype, unless the prototype already defines a function of the same name. If multiple behaviors define the same function, the last behavior in the behaviors array takes precedence over other behaviors.

To extend a behavior, or create a behavior that includes an existing behavior, define a behavior as an array of behaviors.

Instructor Notes:

Add instructor notes here.

Demo

■ Behavior



26

Instructor Notes:

Add instructor notes here.

Extending Native Elements

- Polymer supports to inherit from native HTML elements instead of creating the element from the scratch.
- Extending native element solves accessibility issues.

```
<dom-module id="spl-button">
  <style>
    :host{
      background:red;
    }
    :host:hover{
      background:yellow;
    }
  </style>
  <template>
    <content></content>
  </template>
  <script>
    Polymer({
      is: 'spl-button',
      extends: 'button'
    });
  </script>
</dom-module>
<button is="spl-button">Go!</button>
```

27

For lifecycle events, the lifecycle callback is called for each behavior in the order given in the behaviors array, followed by the callback on the prototype.

Any non-lifecycle functions on the behavior object are mixed into the base prototype, unless the prototype already defines a function of the same name. If multiple behaviors define the same function, the last behavior in the behaviors array takes precedence over other behaviors.

To extend a behavior, or create a behavior that includes an existing behavior, define a behavior as an array of behaviors.

Instructor Notes:

Add instructor notes here.

Demo

- Extending-Native-Element



28

Instructor Notes:

Add instructor notes here.

Event Handling

- In polymer event listeners can be added to the host element by providing a listeners object that maps events to event handler function names.
- Because the event name is specified using an HTML attribute, the event name is always converted to lowercase.
- To add event listeners to local DOM children, use on-event annotations in the template, it eliminates the need to give an element an id solely for the purpose of binding an event listener.
- Listeners can be added and removed imperatively by listen and unlisten methods using automatic node finding.
- To fire a custom event from the host element use the fire method

29

If listeners are added imperatively, then it needs to be removed imperatively alone. This is commonly done in the attached and detached callbacks. If the listeners are used in object or annotated event listeners, Polymer automatically adds and removes the event listeners.

Event retargeting:

Shadow DOM has a feature called "event retargeting" which changes an event's target as it bubbles up, such that target is always in the same scope as the receiving element.

Shady DOM doesn't do event retargeting for events as they bubble, because the performance cost would be prohibitive.

Use Polymer.dom(event) to get a normalized event object that provides equivalent target data on both shady DOM and shadow DOM. Specifically, the normalized event has the following properties:

rootTarget: The original or root target before shadow retargeting (equivalent to event.path[0] under shadow DOM or event.target under shady DOM).

localTarget: Retargeted event target (equivalent to event.target under shadow DOM). This node is always in the same scope as the node where the listener was added.

path: Array of nodes through which event will pass (equivalent to event.path under shadow DOM).

Instructor Notes:

Add instructor notes here.

Demo

- Styling-and-Theming



30

Instructor Notes:

Add instructor notes here.

CSS and Styling

- Polymer uses Shadow DOM styling rules for providing scoped styling of the element's local DOM.
- Scoped styles should be provided via `<style>` tags placed inside the element's local DOM `<template>`.
- Polymer provides a `<style is="custom-style">` custom element for defining styles in the main document.
- Rather than exposing the details of an element's internal implementation for theming, Polymer uses CSS Properties.
- Polymer supports CSS Mixins which allow all properties in the set to be applied to a specific CSS rule in an element's local DOM

31

Document styles defined in a custom-style are shimmed to ensure they do not leak into local DOM when running on browsers without native Shadow DOM.

Styling	
<code>:host</code>	style host element
<code>:host ::content</code>	style distributed content
<code>--custom-property: value</code>	create css property 'custom-property' and give it a 'value'
<code>-- mixin-name: { mixin-contents }</code>	create mixin 'mixin-name' with contents 'mixin contents'
<code>color: var(--my-color, red)</code>	apply custom property 'my color', set default to red
<code>color: @apply(--mixin-name)</code>	apply mixin 'mixin-name' to a property 'color'

Instructor Notes:

Add instructor notes here.

Demo

- Styling-and-Theming



32

Instructor Notes:

Add instructor notes here.

Summary

- Polymer is a new type of library for the web, built on the top of web components.
- By specification, the custom element's name must contain a dash (-).
- The DOM that an element creates and manages is called its local DOM.
- Shady DOM requires Polymer DOM API when manipulating DOM from JavaScript.
- `<dom-module>` element is used to specify DOM to be used for an element's local DOM.
- Property value's type can be a Boolean, Date, Number, String, Array or Object.
- A Property value can be bind with in the element's Local DOM using `{}{}` or `[[[]]]`



33

Instructor Notes:

Add instructor notes here.

Polymer

[Lesson 03: Data Binding](#)

Instructor Notes:

Add instructor notes here.

Lesson Objectives

- At the end of this module you will be able to:
 - Understand Databinding concepts and Binding Annotations
 - Explain the concepts of One Way and Two Way Binding
 - Bind with Objects
 - Bind with Array Items



2

Instructor Notes:

Add instructor notes here.

Databinding in Polymer

- Data binding binds a property or sub-property of a custom element (the host element) to a property or attribute of an element in its local DOM (the child or target element).
- A binding is created with a binding annotation in the host element's local DOM template
- A binding annotation consists of a property name or sub-property name enclosed in curly brackets ({{}}) or square brackets ([[]]).
- Square brackets ([[]]) create one-way bindings. Data flow is downward, host-to-child, and the binding never modifies the host property.
- Curly brackets ({{}}) create automatic bindings. Data flow is one-way or two-way, depending whether the target property is configured for two-way binding.

To bind to a child property, specify the attribute name that corresponds to the property, with an annotation as the attribute value:

```
<child-element name="{{myName}}"></child-element>
```

The above example binds the child element's name property to the host element's myName property.

Property name to attribute name mapping:

For data binding, deserializing properties from attributes, and reflecting properties back to attributes, Polymer maps attribute names to property names and the reverse.

When mapping attribute names to property names:

- Attribute names are converted to lowercase property names. For example, the attribute firstName maps to firstname.
- Attribute names with dashes are converted to camelCase property names by capitalizing the character following each dash, then removing the dashes. For example, the attribute first-name maps to firstName.

The same mappings happen in reverse when converting property names to attribute names (for example, if a property is defined using reflectToAttribute:true.)

3

Instructor Notes:

Add instructor notes here.

Binding to text content

- To bind to a child element's.textContent, just include the binding annotation inside the child element.
- Binding to text content is always one-way, host-to-child even though {{ }} is used.

```
<dom-module id="cg-foo">
  <template>
    First Name : {{firstName}} Last Name : {{lastName}}
  </template>
  <script>
    Polymer({
      is: 'cg-foo',
      properties: {
        firstName: String,
        lastName: String
      }
    });
  </script>
</dom-module>

<cg-foo first-name="Karthik" last-name="Muthukrishnan"></cg-foo>
```

When binding to style, href, class, for or data-* attributes, it is recommended to use attribute binding syntax.

To bind to an attribute, use \$= rather than =. This results in a call to:

element.setAttribute(attr,value);

As opposed to:

element.property = value;

How configuration and binding annotation affect direction of data binding:

Child property config	notify: false readOnly: false	notify: true* readOnly: false	notify: true* readOnly: true	notify: false readOnly: true
Binding				
{}{{curly}}	host --> child	host <--> child	host <-- child	no binding
[[square]]	host --> child	host --> child	silly	no binding

So using curly braces only takes effect when the child's property is notify: true, and setting the child to notify: true only takes effect when the host uses curly braces.

Instructor Notes:

Add instructor notes here.

Compound bindings

- String literals and bindings can be combined in a single property binding or text content binding
- Compound bindings are re-evaluated whenever the value of any of the individual bindings changes.
- Undefined values are interpolated as empty strings.
- Compound bindings are always one-way i.e. host-to-target.

```
<dom-module id="og-foo">
<template>
  <span>Full Name: {{firstName}} {{lastName}}</span>
  <br/>
  
</template>
<script>
  Polymer({
    is: 'og-foo',
    properties: {firstName: String, lastName: String}
  });
</script>
</dom-module>

<og-foo first-name="Karthik" last-name="Muthukrishnan"></og-foo>
```

5

Instructor Notes:

Add instructor notes here.

Two-way binding to native elements

- Polymer uses an event naming convention to achieve two-way binding.

- `target-prop="{{hostProp::target-change-event}}"`

```
<dom-module id="cg-foo">
  <template>
    <div>
      Name : <input type="text" value="{{name::input}}"/> <span>{{name}}</span><br/>
      Marital Status : <input type="checkbox" checked="{{maritalStatus::change}}"/>
      <span>{{maritalStatus}}</span>
    </div>
  </template>
  <script>
    Polymer({
      is: 'cg-foo',
      properties: {
        name: String,
        maritalStatus:{type:Boolean, value:true}
      }
    });
  </script>
</dom-module>
```

5

To avoid two-way binding, use “square-brace” syntax ([[property]]), which results in only one-way (downward, host-to-child) data-binding.

To summarize, two-way data-binding is achieved when both the host and the child agree to participate, satisfying these three conditions:

- The host must use curly-brace {{property}} syntax. (Square-brace [[property]] syntax results in one-way downward binding, regardless of how the child property is configured.)
- The child property being bound to must be configured with the notify flag set to true (or otherwise send a <property>-changed custom event). (If the property being bound does not have the notify flag set, only one-way (downward) binding will occur.)
- The child property being bound to must not be configured with the readOnly flag set to true. (If the child property is notify: true and readOnly: true, and the host binding uses curly-brace syntax, the binding is one-way, upward (child-to-host).)

Instructor Notes:

Add instructor notes here.

Demo

- OneWay-Compound-TwoWay
- OneWay-TwoWay-Pasing-values



7

Instructor Notes:

Add instructor notes here.

How Binding Annotation works inside Polymer

Look for the nodes with Binding Annotation

Create Property Effects and setter for the property

Performs Dirty checking with in the setter

If Old Value is not equal to New Value

Loop overs the property effects and updates the nodes in the DOM

8

```
<div>{{foo}}</div>
<paper-input value="{{foo}}">
</paper-input>

Polymer({
  is:'cg-binding',
  property :{
    foo:{
      observer:'fooChanged',
      notify:true
    }
  },
  set foo(val){
    /*Dirty checking*/
    var old = this.__data__.foo;
    if(old !== val){
      this.__data__.foo = val;
      //Loops over the property effects
      //and execute their actions
      this.fooChanged();
      this.fire('foo-changed', {value:val})
    }
  }
});
```

Instructor Notes:

Add instructor notes here.

Demo

▪ Observers



19

Instructor Notes:

Add instructor notes here.

Binding to Objects

- Sub-properties of the host can be binded by specifying a path inside the binding annotation.
- If a path is changed automatically by a binding, the change propagates normally to other bindings to that path (and its sub-paths).
- If a path is changed imperatively, Polymer provides two methods that allow such changes to be notified to the system: `notifyPath(path, value)` and `set(path, value)`, where path is a string identifying the path (relative to the host element).
- `notifyPath` is called directly after an assignment, so a convenience function `set` is provided that performs both actions

10

Path change notification

Two-way data-binding and observation of paths in Polymer is achieved using a similar strategy to 2-way property binding:

When a sub-property of a property configured with type: Object changes, an element fires a non-bubbling `<property>-changed` DOM event with a `detail.path` value indicating the path on the object that changed.

Elements that have registered interest in that object (either via binding or change handler) may then take the appropriate action.

Finally, those elements will forward the notification on to any children they have bound the object to, and will also fire a new `<property>-changed` event where property is the root object, to notify any hosts that may have bound root object down.

Instructor Notes:

Add instructor notes here.

Demo

- Binding-Object



11

Instructor Notes:

Add instructor notes here.

Binding to Array

- Bindings to array items by index isn't supported in polymer e.g. `users[0].name`
- Computed binding is used to bind to a specific array item, or to a subproperty of an array item e.g. `array[index].name`
- The computing function needs to be called if the subproperty value changes, or if the array itself is mutated, so the binding uses a wildcard path, `myArray.*`.
- Computing function takes 3 arguments
 - First argument is the change record for the array change, `change.base` is the array specified in the binding
 - Second is the index position
 - Third is the value for a path relative to a root object

12

To avoid two-way binding, use “square-brace” syntax (`[[property]]`), which results in only one-way (downward, host-to-child) data-binding.

To summarize, two-way data-binding is achieved when both the host and the child agree to participate, satisfying these three conditions:

- The host must use curly-brace `{{property}}` syntax. (Square-brace `[[property]]` syntax results in one-way downward binding, regardless of how the child property is configured.)
- The child property being bound to must be configured with the `notify` flag set to true (or otherwise send a `<property>-changed` custom event). (If the property being bound does not have the `notify` flag set, only one-way (downward) binding will occur.)
- The child property being bound to must not be configured with the `readOnly` flag set to true. (If the child property is `notify: true` and `readOnly: true`, and the host binding uses curly-brace syntax, the binding is one-way, upward (child-to-host).)

Instructor Notes:

Add instructor notes here.

Demo

- Binding-Array



13

Instructor Notes:

Add instructor notes here.

Summary

- Polymer supports one way and two way binding using the binding annotation `[[]]` and `{{ }}` respectively.
- When binding to style, href, class, for or data-* attributes, it is recommended that you use attribute binding syntax.
- Two-way bind to native elements specifies a custom change event name in the binding annotation.
- Sub-properties of the host can be bound by specifying a path inside the binding annotation.
- Paths do not support array access notation (such as `users[2]`). String keys (such as `users[bob]`) can be replaced with dotted paths (`users.bob`).



14

Instructor Notes:

Add instructor notes here.

Polymer

[Lesson 04: Polymer Elements](#)

Instructor Notes:

Add instructor notes here.

Lesson Objectives

- At the end of this module you will be able to:
 - Use the Polymer elements given in the element catalog



2

Instructor Notes:

Add instructor notes here.

Introduction to Polymer Elements

- The Polymer element set provide elements that can be used in web pages and apps.
- These elements are built with the Polymer library.
- No need to use Polymer directly to use these elements. However, using Polymer can give advantage of special features such as data binding.

Instructor Notes:

Add instructor notes here.

Polymer Element Catalog

The Polymer Element Catalog displays eight primary collections of elements arranged in a 2x4 grid:

- App** (0.9.0) - App Elements: App elements
- Fe** (1.0.10) - Iron Elements: Polymer core elements
- Md** (1.0.7) - Paper Elements: Material design elements
- Go** (1.0.1) - Google Web Components: Components for Google's APIs and services
- Au** (1.0.1) - Gold Elements: Ecommerce Elements
- Ne** (1.0.0) - Neon Elements: Animation and Special Effects
- Pt** (2.0.0) - Platinum Elements: Offline, push, and more
- Mo** (1.0.0) - Molecules: Wrappers for third-party libraries

Polymer contains a few primary collections of elements:

App Elements : It enables building full web apps out of modular custom elements.

Iron elements : A set of utility elements including generic UI elements (such as icons, input and layout components), as well as non-UI elements providing features like AJAX, signaling and storage.

Paper elements : A set of UI elements that implement the material design system.

Gold elements : Form elements for ecommerce.

Neon elements : Animation-related elements.

Platinum elements : Elements for app-like features, like push notifications, offline caching and bluetooth.

Instructor Notes:

Add instructor notes here.

Installing Elements

- Polymer elements can be installed one at a time, or install a whole collection of elements.
- Element can be added / removed from the download collection by clicking  from the polymer catalog list



- Selected components can be download either as bower.json file or ZIP file by click  and selecting the download tab.
- When component is downloaded as a ZIP file, all of its dependencies bundled into a single archive, no need to install any additional tools.

Instructor Notes:

Add instructor notes here.

Using Elements

- To use elements, first load the web components polyfill library, `webcomponents-lite.min.js` which provides polyfill support for the browsers which haven't yet implemented the web components APIs.
- Load the element file using an HTML Import
- Application need to be run from a web server for the HTML Imports polyfill to work properly. This requirement goes away when the API is available natively.

Be sure to include `webcomponents-lite.min.js` before any code that touches the DOM.

To use Polymer elements effectively, it's important to be familiar with a few of the common Polymer element APIs.

The Polymer element docs list element properties, but not element attributes.

When mapping attribute names to property names:

- Attribute names are converted to lowercase property names.
- Attribute names with dashes are converted to camelCase property names by capitalizing the character following each dash, then removing the dashes.

HTML attributes are string values, but many Polymer elements understand attribute values that are a JSON-serialized object or array.

Polymer elements also support data binding but only In the local DOM template of a Polymer element or Inside an autobinding template.

Polymer element, need to use the `Polymer.dom` APIs to maintain the correct DOM structure. This is particularly important when adding or removing children from a Polymer element

Many of the Polymer elements can be styled using custom CSS properties and mixins.

Property and mixin values can be specified:

- Inside a Polymer element's `<style>` tag to specify values scoped to that element.
- Inside a custom-style element to specify values in the document scope.

Instructor Notes:

Add instructor notes here.

Demo

- Using-Elements



7

Instructor Notes:

Add instructor notes here.

Paper Elements

- Paper elements are a set of visual elements that implement Google's Material Design.



8

The Paper Elements is a group of Material Design elements.

Material Design is Google design language to makes user interface and experience across Google platforms both the Web and Android apps more visually consistent.

Some elements that are unique to Material Design are Paper and Floating Action Button (FAB).

Instructor Notes:

Add instructor notes here.

Demo

- Paper-Elements



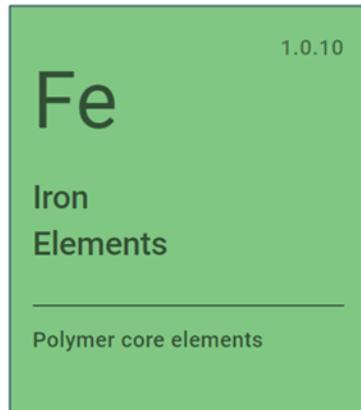
19

Instructor Notes:

Add instructor notes here.

Iron Elements

- A set of visual and non-visual utility elements. Includes elements for working with layout, user input, selection, and scaffolding apps.



10

Iron Elements is a collection of basic elements. These basic elements are what we normally use to build a web page such as an input, form and image. The difference is Polymer adds some extra powers to these elements.

Instructor Notes:

Add instructor notes here.

Demo

- Iron-Elements
- dom-bind-demo
- iron-ajax-demo
- iron-iconset-svg-demo



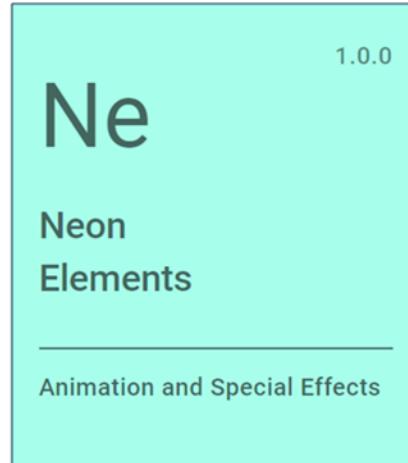
11

Instructor Notes:

Add instructor notes here.

Neon Elements

- Neon elements implements animation and special effects.



12

Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

Instructor Notes:

Add instructor notes here.

Demo

■ Neon-Elements



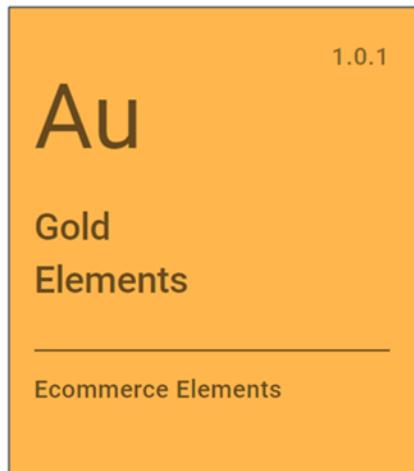
13

Instructor Notes:

Add instructor notes here.

Gold Elements

- The gold elements are built for e-commerce use-cases like checkout flows.



14

Be sure to include `webcomponents-lite.min.js` before any code that touches the DOM.

Instructor Notes:

Add instructor notes here.

Demo

▪ Gold-elements



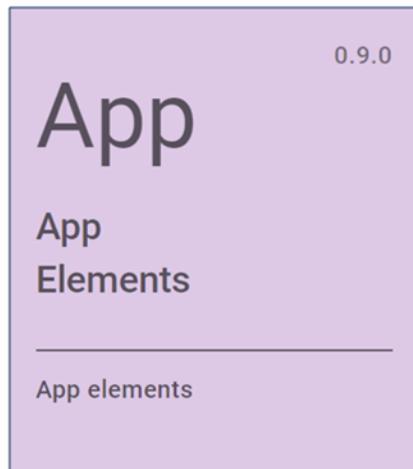
15

Instructor Notes:

Add instructor notes here.

App Elements

- The app elements enable building full web apps out of modular custom elements.



16

Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

Instructor Notes:

Add instructor notes here.

Demo

- App-elements
- Routing



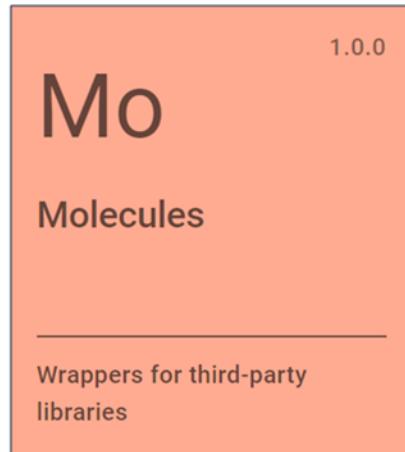
17

Instructor Notes:

Add instructor notes here.

Molecules

- Paper elements are a set of visual elements that implement Google's Material Design.



18

Be sure to include `webcomponents-lite.min.js` before any code that touches the DOM.

Instructor Notes:

Add instructor notes here.

Demo

▪ Molecules



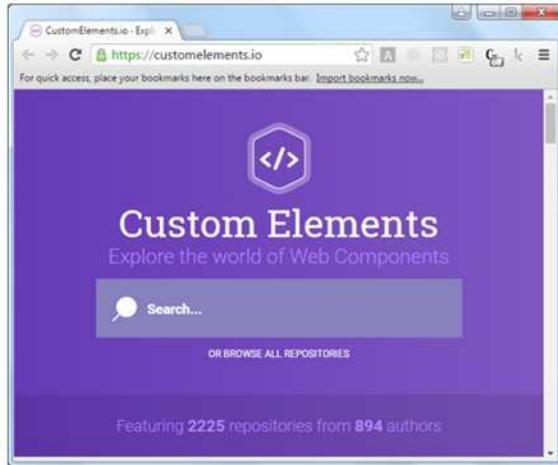
19

Instructor Notes:

Add instructor notes here.

customelements.io

- A gallery to display Web Components created by the community.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

20

Instructor Notes:

Add instructor notes here.

Summary

- Material design UI elements implemented using Polymer Paper elements.
- Iron Elements is a collection of basic elements.
- Gold Elements are the elements designed specifically for e-commerce apps.
- Neon elements are used for animation and special effects.
- Platinum elements for offline and push notifications.
- Google Web Components are special elements that cope with Google APIs and services such as Google Maps, Youtube, as well as Google Feed.
- CustomElements.io is a hub where we can find custom elements built with Web Components.



21