Polymer

- Project began in July 2013
- 2013: 0.1 (Nov)
- 2014: 0.2 (Feb), 0.3 (May), 0.4 (Aug), 0.5 (Nov)
- 2015: 0.8 (Apr), 0.9 (May), 1.0 (May), 1.1 (Aug). 1.2 (Oct)
- 2016: 1.3 (Feb), 1.4 (Mar),1.5 (May), 1.6 (Jun), 1.7 (Sep)
- 2017: 1.8 (Feb), 1.9 (Apr), 2.0 (May), 2.1 (Sep)
- **Polymer 2.2 (Oct 18, 2017)**
- *Polymer 3.0 in May 2018?*

Prerequisites

1. Install Node.js from nodejs.org (LTS version 6.11.x)
   a. Node Package Manager (`npm`) # package manager for Node.js
2. Install Polymer Command Line Interface (CLI) & App
   a. `npm install -g bower` # package manager for the web app
   b. `npm install -g polymer-cli`
   c. `mkdir polymer-sample`
   d. `cd polymer-sample`
   e. `polymer init` (select Polymer 2 Application, choose a valid name for your element)
   f. `polymer serve --open`
   g. `[Open browser at http://localhost:NNNN]`

Demo #1: Setup and Run

Project Files

```
bower.json              # bower (package manager) configuration file
bower_components/       # dependencies for your app
index.html              # your app main entry point
manifest.json           # meta data about your app
polymer.json            # used by polymer CLI
src/                    # source directory of your app (HTML, JS, CSS)
test/                   # unit tests
```

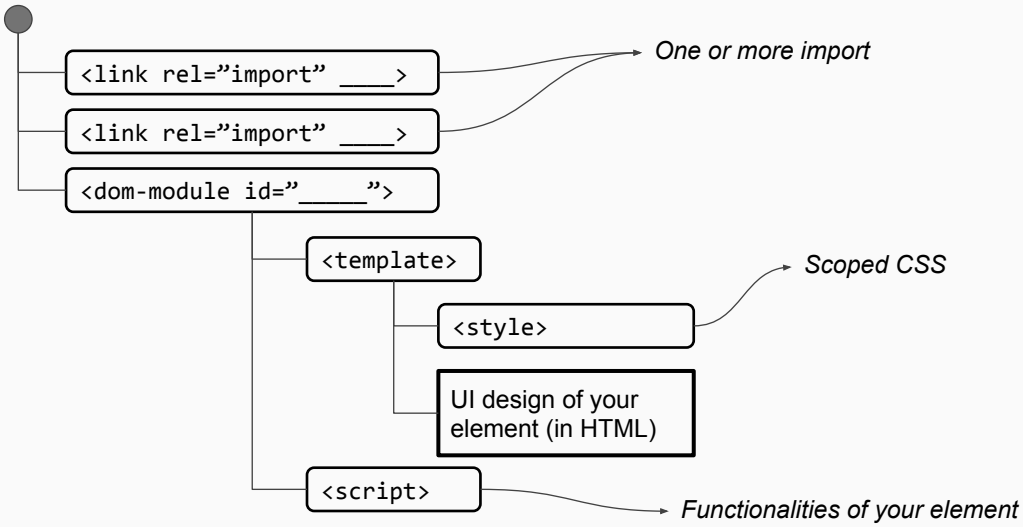
Generated Files

```
<!-- index.html -->
<html>
<head>
<link rel="import" href="src/demo-app.html">
</head>
<body>
<demo-app></demo-app>
</body>
```

- Whatever you put under `<template>` goes to a shadow DOM
- `[[__]]` is a syntax for data binding

```
<!-- demo-app.html -->
<link rel="import" href="____/polymer-element.html">
<dom-module id="demo-app">
  <template>
    <style> /* style of this element */ </style>
    <!-- HTML contents of this element -->
    <h2>Hello [[prop1]]</h2>
  </template>
  <script>
  class DemoApp extends Polymer.Element {
    static get is() { return 'demo-app'; }
    static get properties() {
      return {
        prop1: {
          type: String, value: 'demo-app'
        }
      };
    }
  }
  window.customElements (DemoApp.is, DemoApp);
  </script>
</dom-module>
```


Organization of a Polymer Element



Structure of a Polymer Element

```
<!-- my-elem.html -->
<dom-module id="my-elem">
  <template>
    <!-- your element UI goes here -->
  </template>

  <script>
    /* your element logic goes here */
  </script>
</dom-module>
```

```
<!-- my-elem.html -->
<dom-module id="my-elem">
  <template>
    <style>
      // UI style rules here
    </style>
    <!-- your element UI goes here -->
  </template>
  <script>
    /* your element logic goes here */
  </script>
</dom-module>
```

Android	: UI in XML, code in Java
iOS	: UI in XML, code in Swift
Polymer	: UI in XML, code in JavaScript

Structure of a Polymer Element

```
<dom-module id="my-elem">
  <template>
    <!-- UI of your element goes here -->
  </template>
  <script>
  class MyElement extends Polymer.Element {
    static get is() {
      return 'my-elem';
    }

    static get properties() {
      /* shown in a separate slide */
    }
    /* other methods */
  }
  customElements.define(MyElement.is, MyElement);
  </script>
</dom-module>
```

Every Polymer custom element must define the `is()` *function*

Custom Element Properties

- EcmaScript 6 (ES6) provides no direct declarative way for declaring "instance variables" like we know in Java
  - We can only declare methods
- But, custom elements may require data to work with Polymer data system
  - A Polymer element may define some attributes (like any other HTML elements)
  - A Polymer element may send notifications when its data changes
- Use *a function that returns an object* containing all the "data" used by a Polymer element
  - A getter function: `properties()`

Polymer Element Lifecycle

- Callbacks common to other custom elements
  - `constructor()`: when element is created
  - `connectedCallback()`: when element is inserted into the (local) DOM
  - `disconnectedCallback()`: when element is removed from the (local) DOM
  - `attributeChangedCallback()`: when one of element attributes is updated, ….
- Additional callback (Polymer specific)
  - `ready()`: when the element is inserted for **the first time** into the DOM
    - `super.ready()`: Polymer.Element initializes your template and properties
- **Remember** to call `super.`_____`()` in any of the functions above!

Demo #2: Click Counter

- Writing static getter function: `is()`
- Writing callback function: `ready()`
- Writing the getter function `properties()`
- Shadow DOM Styling (`:host`)
- Event listeners
- Data binding to HTML attributes
- Use bower to download external dependencies
  - http://webcomponents.org
  - http://material.io/icons

Custom Element Styles

```
<dom-module id="sam-ple">
  <template>
    <style>
    :host {                    // apply to the entire custom element
      font-size: 80%;
      display: inline-block;
    }
    :host(.warn) { font-style: bold; color: brickred; }

    h2 { border: 2px solid blue; } // apply only to <h2>s
    </style>

    <h2>Hello</h2>
    <span class="warn">Hi</span>

  </template>
  <script>_____</script>
</dom-module>
```

Data Binding to Element Attributes

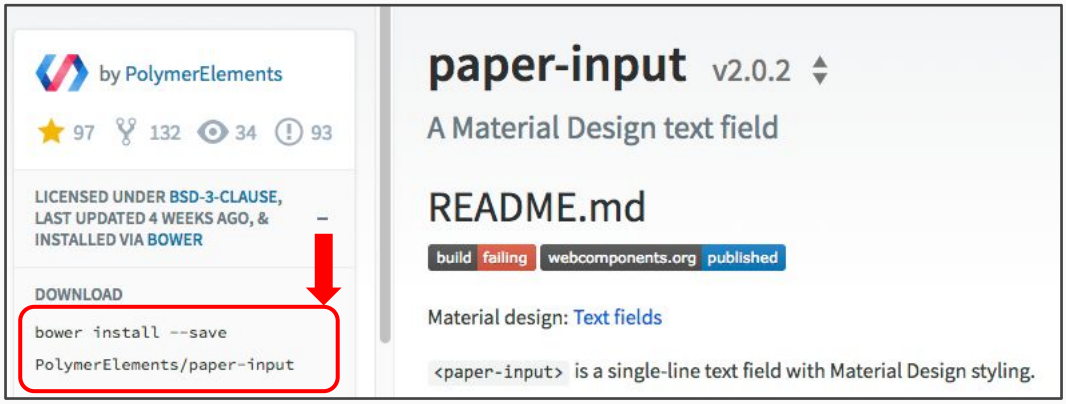
- Use `attr=[[property_name]]` when `attr` is a not a “native” HTML attribute
- Use `attr$=[[property_name]]` when `attr` is a native HTML attribute

Binding to Attributes

- src is a native HTML attribute, must append $ to the attribute name
- zip is a not a native HTML attribute

```
<dom-module id=”exam-ple”>
  <template>
    <img src$=”[[weatherIcon]]” _____>
    <local-weather zip=”[[userZip]]”></local-weather>
  </template>
  <script>
    class Example extends Polymer.Element {
      static get properties() {
        return { weatherIcon: String,
                 userZip: String   }
      }
    }
  </script>
</dom-module>
```

WebComponents.org: Search and download custom elements



Callback functions: Polymer 1.x and Polymer 2.x

Polymer 1.x	Polymer 2.x
`created()`	`constructor()`
`ready()`	`ready()`
`attached()`	`connectedCallback()`
`detached()`	`disconnectedCallback()`
`attributeChanged()`	`attributeChangedCallback()`

properties()

```
// two properties
// (simplest declaration)
static get properties() {
  return {
    isAuth: Boolean,
    uName: String
  }
}
```

```
// two properties, more elaborate
static get properties() {
  return {
    isAuth: {
      type : Boolean,
      readOnly: true,
      notify: true
    },
    uName: {
      type: String,
      value: “Anonymous”
    }
  }
}
```


HTML *attributes*

≠

Polymer *properties*

HTML Attributes vs Polymer Properties

```
<img src="snow.png" >
<local-forecast zip="49401"></local-forecast>
```

"zip" is an HTML attribute

"zip" is a Polymer property (in Forecast class)

```
<dom-module id="local-forecast">
  <template>
    <span>Condition at [[zip]]</span>
  </template>
  <script>
  class Forecast extends Polymer.Element {
    static get properties() {
      return {
        //zip : String
      }
    }
  }
  </script>
</dom-module>
```

- When the property "zip" is NOT defined, the will show only "`Condition at `"
- When the property "zip" is defined, the HTML attribute zip is automatically mapped into the property (*one-way data flow from the attribute to the property*) and the will show "`Condition at 49401`"

Additional reference

Property Configuration

```
static get properties() {
  return {
    myProperty: {
      type     : _____,        /* Number, Boolean, String, Array, Date */
      value    : _____,        /* initial value */

      readOnly: true,          /* can't be altered by direct assignment,
                                  must use setter function */
      notify  : true,          /* prepare for 2-way data binding */
      computed: "abc(x,y)",    /* the function "abc" is invoked to calculate the
                                  property value when x and y change */
      observer: "_____",       /* name of a function that gets invoked when value
                                  changes */
      reflectToAttribute: true  // allow 2-way property ⇔ attribute mapping
    }
  }
}
```

Attributes to Properties: name mapping

Attribute Name	Property Name
kebab-case	*camelCase*
`imageloaded`	`imageloaded`
`image-loaded`	`imageLoaded`
`user-image-loaded`	`userImageLoaded`

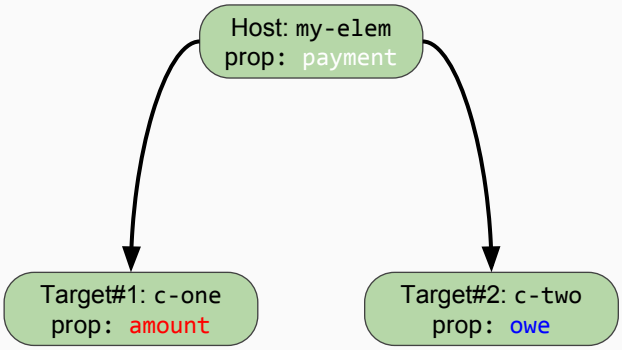
Polymer Data Binding

- Practical Usage
  - Binds data between a custom element to a (subordinate) element in its shadow DOM
  - Binds data between a custom element and another within the same shadow DOM
- Host: custom element defines a local DOM/shadow DOM
- Local DOM = target element
  - Shadow DOM is also referred to as “shadow host”
- ***One of the techniques for allowing two (or more) custom elements to communicate***

Data Binding: One-Way

```
<dom-module id="my-elem">
  <template>
    <c-one amount=[[payment]]></c-one>
    <c-two owe=[[payment]]></c-two>
  </template>
</body>
```

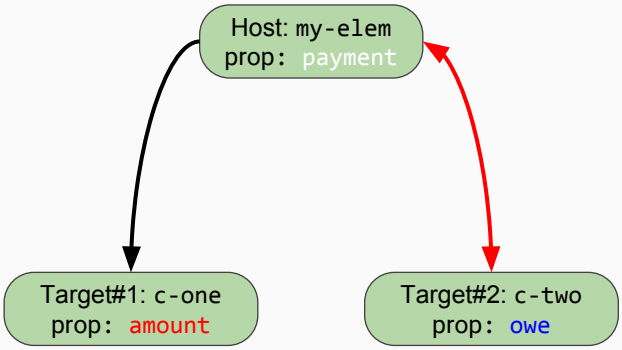
- Properties "amount" of <c-one> and "owe" of <c-two> are bound to property "payment" of <my-elem>
- Updates to payment by <my-elem> are **automatically observed** by both <c-one> and <c-two>
- Neither updates to "owe" by <c-two> nor "amount" by <c-one> are observed by <my-elem>.



Data Binding: Two-Way

```
<dom-module id="my-elem">
  <template>
    <c-one amount=[[payment]]></c-one>
    <c-two owe={{payment}}></c-two>
  </template>
</body>
```

- An example of **two-way** binding
- Properties "amount" of <c-one> and "owe" of <c-two> are bound to "payment" of <my-elem>
- Updates to payment by <my-elem> are **automatically observed** by both <c-one> and <c-two>
- Updates to "owe" by <c-two> can be observed by <my-elem>. Therefore, they ***can be observed*** by <c-one>



Demo #3: Data Binding

`[[one-way]] & {{two-way}}`

Data Flow in Data Binding

`notify`	`readOnly`	**Data Flow**
`false`	`false`	(default) One-way: host to target
`false`	`true`	No data flow
`true`	`false`	Two-way
`true`	`true`	One-way: target to host

.bind(this)

```
class Abc extends Polymer.Element {
  aFunction() {
    this.data = 100;
    // Using ordinary function
    __.addEventListener('___', function() {
      this.data--;  // won't update this.data
    });
  }
}
```

```
class Abc extends Polymer.Element {
  aFunction() {
    this.data = 100;
    // Using arrow operator
    __.addEventListener('___', () => {
      this.data--;  // updates this.data
    });
  }
}
```

The (blue) this and (red) this have two different scopes.

```
class Abc extends Polymer.Element {
  aFunction() {
    this.data = 100;
    // Using ordinary function
    __.addEventListener('___', function() {
      this.data--;  // updates this.data
    }.bind(this));
  }
}
```


Simple Property Observers

```
class MyElement extends Polymer.Element {
  static get properties() {
    return {
      userName: {
        type     : String,
        value    : null,
        observer: "_userUpdated"  // coding convention: underscore prefix
                                  // for "private" functions
      }
    }
  }

  _userUpdated(newValue, oldValue) {
    // this function gets invoked when the property
    // userName changes
  }
}
```

Complex Property Observers

```
class MyElement extends Polymer.Element {
  static get observers() {
    return [
      '_submissionChanged(qid,response)',  // qid and response are properties
      '_orderUpdated(orderId,quantity)'
    ]
  }

  _submissionChanged(q, r) {
    // this function gets invoked when either the property qid or response changes
  }

  _orderUpdated(id, qty) {
    // this function gets invoked when either the property orderId or quantity changes
  }
}
```

Use these observers when your custom element must react to "simultaneous" changes on two or more values

Reference Element By ID

```
// plain JavaScript
var elem = ____.getElementById("myInput");
```

```
// jQuery
var elem = $("#myInput");
```

```
// Polymer
var elem = this.$.myInput;
```

Connecting to Firebase DB

```
# download polymerfire dependencies (latest version 2.2.x)
bower install --save firebase/polymerfire
```

```
<link rel="import" href="___/bower_components/polymerfire/firebase-app.html">
<link rel="import" href="___/bower_components/polymerfire/firebase-query.html">
<dom-module id="___">
  <template>
    <firebase-app api-key="___" auth-domain="___" database-url="___"
                  storage-bucket="___" messaging-sender-id="___"></firebase-app>
    <firebase-query path="/path/to/node/of/interest/" data="{{orders}}">
    </firebase-query>

    <ul>
    <template is="dom-repeat" items="[[orders]]" as="ord">
      <li>[[ord.$key]]  [[ord.itemName]]  [[ord.quantity]]</li>
    </template>
    </ul>
  <template>
</dom-module>
```


Manipulating Firebase DB

```
<link rel="import" href="___/bower_components/polymerfire/firebase-app.html">
<link rel="import" href="___/bower_components/polymerfire/firebase-query.html">
<dom-module id="___">
  <template>
    <firebase-app id="frApp"
       api-key="___" auth-domain="___" database-url="___"
       storage-bucket="___" messaging-sender-id="___"></firebase-app>

  <template>
</dom-module>
```

```
// JavaScript
var dbRef = this.$.frApp.database();

dbRef.ref('_____').push().set({ your data here });
dbRef.ref('_____').push().update({ your data here });
```

Demo #4: Firebase

`<firebase-app> & <firebase-query>`

Custom CSS Properties

(Custom CSS Variables)