Customizing 3rd party Custom Elements

- *Technical challenges*
  - **CSS styles** defined under a shadow DOM **are scoped** only for elements under its shadow root
  - CSS styles defined outside a shadow DOM have to effect on elements under its shadow root
- *Solution*
  - a custom element can provide CSS **custom variables** or **mixins** for its user to customize the element visual styles

Example: paper-button CSS custom variables

The following custom properties and mixins are also available for styling:

Custom property	Description	Default
`--paper-button-ink-color`	Background color of the ripple	`Based on the button's color`
`--paper-button`	Mixin applied to the button	`{}`
`--paper-button-disabled`	Mixin applied to the disabled button. Note that you can also use the `paper-button[disabled]` selector	`{}`
`--paper-button-flat-keyboard-focus`	Mixin applied to a flat button after it's been focused using the keyboard	`{}`
`--paper-button-raised-keyboard-focus`	Mixin applied to a raised button after it's been focused using the keyboard	`{}`

Setting Styles “Globally”

```
<!-- in index.html -->
<html>
  <head>
    <link rel=”import” href=”____/polymer/lib/elements/custom-style.html”>
    <custom-style>
      <style>
      body {
        --paper-button-ink-color: darkorange; // custom property

        --paper-button: {                     // mixin
          font-size: 80%;
          border-radius: 8px;
        }
      }
      </style>
    </custom-style>
  </head>
  <body>
    <custom-er></custom-er>
    <cust-ard></cust-ard>
  </body>
</html>
```

- *CSS custom properties define a single value (“darkorange”)*
- *CSS mixins define a set of CSS properties (“font-size” and “border-radius”)*

Flex Box

- Flexbox Layout: a newer CSS standard for better layout management
- Align, distribute space among items in a container
- Two types of entity: (flex) containers and (flex) items
  - Containers are parents that hold items
  - Items are children of a container
  - A container may hold a subordinate container
- Polymer element: `iron-flex-layout`
  - <link rel=”import” href=”____/iron-flex-layout/iron-flex-layout-classes.html”>
  - New classes that you can use for designing your UI
- Online Reference

Importing iron-flex-layout

```
<link rel="import" href="____/iron-flex-layout/iron-flex-layout-classes.html">
<dom-module id="sam-ple">
  <template>
    <!-- include flex modules needed by your element -->
    <style include="iron-flex iron-flex-alignment"></style>
    <style>
    :host {
      /* define your properties here */
    }

    .myclass {
      /* define your properties here */
    }
    </style>
  </template>
</dom/-module>
```


Iron-flex-layout Modules

- Iron-flex
- Iron-flex-reverse
- Iron-flex-alignment (main axis, cross-axis, self alignment)
- Iron-flex-factors (proportional flex)
- Iron-flex-positioning (miscellaneous)

CSS3 Flexbox Properties

Container Property	Description
`display`	To enable flexbox, set display to flex or inline-flex
`flex-direction`	row, row-reverse, column, column-reverse
`flex-wrap`	Nowrap, wrap, wrap-reverse
`justify-content`	Alignment along the main axis
`align-items`	Align individual items along the cross-axis
`align-content`	Align the entire content along the cross-axis

Children Property	Description
`flex-grow`	How much item can grow
`flex-shrink`	How much item can shrink
`flex-basis`	Define defult size of an item before the remaining space is distributed
`align-self`	Override align-items settings

Additional Reference

Flexbox

- Two types of container layout: horizontal & vertical
  - Recall Java Swing BoxLayout?
- Main Axis vs. Cross Axis
  - In a horizontal container: main axis ⇒ X-axis, cross axis ⇒ Y-axis
  - In a vertical container: main axis ⇒ Y-axis, cross axis ⇒ X-axis
- Justification: positioning along the main axis
- Alignment: positioning along the cross axis

Iron Flex Layout Classes

http://www.cis.gvsu.edu/~dulimarh/CS371/LayoutDemo

Demo: Layout Faculty List

Event Handling

- Declaratively using on-*eventname* attribute (preferred)
- Imperatively using JavaScript function calls
  - `____.addEventListener('event_name', handlerFunction);`
  - `____.removeEventListener('event_name', handlerFunction);`

Declarative Event Handling

```
<dom-module id="sam-ple">
  <template>
    <paper-button on-click="updateOrder">Confirm</paper-button>
  </template>
  <script>
    class Sample extends Polymer.Element {
      static get is() { return 'sam-ple'; }
      updateOrder() {
        /* process order here */
      }
    }
    customElements.define(Sample.is, Sample)
  </scipt>
</dom-module>
```

Event Handling in dom-repeat template

```
<dom-module id=”sam-ple”>
  <template is=”dom-repeat” items=[[orders]]” as=”ord”>
    <span>[[ord.prodName]]</span>
    <paper-button on-click=”addMe”>Add To Cart</paper-button>
  </template>
  <script>
    class Sample extends Polymer.Element {

      addMe(ev) { // use ev.model to access the item
        alert (“Adding “ + ev.model.ord.prodName);
      }
    }
  </scipt>
</dom-module>
```


dom-repeat model object

```
<template is=”dom-repeat” items=[[orders]]” as=”ord”>
  <span>[[ord.prodName]]</span>
  <paper-button on-click=”addMe”>Add To Cart</paper-button>
</template>
```

```
/* object associated with each template item */
{
  index: ____
  ord: { /* whatever the structure of your orders array */
    prodName: _______,
    otherField: ____
  }
}
```

Demo: “Dialing” Faculty Phone

Using Polymer Elements

- <paper-button>
- <iron-icon>
- <iron-list>
- <paper-card>
- <paper-dialog>
- <paper-input type=”file”>

Using <iron-list>

- Scrollable list of items
- Each item is generated within <template>*contents of item*</template>
- Parent of `<iron-list>` must be assigned explicit height and flex properties

```
iron_list {
  height: 80vh;     /* 80% of view height */
}
```


<iron-list> template

```
<iron-list items="[[yourArrayOfObjects]]" as="student">
<template>
  <div>  <!-- item contents must be inside a div →

    <span>([[index]]) [[student.name]]</span>

 </div>
</template>
</iron-list>
```

```
// Each template instance is associated with the following model
{
  index   : _____,   /* integer 0 - (N-1) */
  selected: _____,   /* boolean */
  student : {
    /* what
  }
}
```

Using <paper-card>

```
<paper-card heading="Title">
  <div class="card-content">
    <!-- customize your contents here -->
  </div>

  <div class="card-actions">
    <!-- customize your actionss here (usually buttons) -->
  </div>
</paper-card>
```

card-content and card-actions are CSS classes defined by paper-card

Using <paper-dialog>

```
<!-- insert the following inside the custom element that will use the dialog -->
<paper-dialog id="orderDialog" on-iron-overlay-closed="diaClosed">
  <paper-input id="qty" label="Quantity"></paper-input>

  <div class="buttons">
    <paper-button dialog-dismiss>Cancel</paper-button>
    <paper-button dialog-confirm>Order Now</paper-button>
  </div>
</paper-card>
```

dialog-dismiss and dialog-confirm are attributes defined by paper-dialog.

The closingReason property of the dialog includes two boolean fields: cancelled and confirmed

```
// To show the dialog
this.$.orderDialog.show()
```

```
class ____ extends Polymer.Element {
  diaClosed() {  // dialog close handler
    var reason = this.$.orderDialog.closingReason;
    if (reason.confirmed) {
      // Confirm button was pressed
      alert ("Quantity is " + this.$.qty.value);
    }
  }
}
```

Using <paper-input type=”file”>

```
<paper-input id=”myFile” type=”file”>
<paper-button on-click=”doUpload”>UpLoad</paper-input>
```

```
class ____ extends Polymer.Element {
  doUpload() {
    var fileInfo = this.$.myFile.inputElement.inputElement;
    if (“files” in fileInfo) {
      var reader = new FileReader();
      reader.onload = event => {
        // The file content is in event.target.result

      };
      reader.readAsArrayBuffer(fileInfo.files[0]);  // for binary files
      // reader.readAsText(fileInfo.files[0]);      // for text files
    }
  }
}
```


Browsers Support

	Chrome	Firefox	IE/Edge	Opera	Safari
Template	✔	✔	✔	✔	✔
Imports	✔	Polyfill	Polyfill	✔	Polyfill
Custom Elements	✔	Polyfill	Polyfill	✔	✔
Shadow DOM	✔	Polyfill	Polyfill	✔	✔

Using Polyfill

```
<!-- in index.html -->
<script>
  window.customElements.forcePolyfill = true;
</script>
<script src="/bower_components/webcomponentsjs/webcomponents-loader.js"></script>
```


Mixins

Mix In ⇒ Mixins

Ice Cream Flavors

- Strawberry Cheesecake
- Mint Chocalate Chip
- Deer Trax
- Toffee
- Etc....

Vanilla Ice Cream + Additional ingredients

Additional Reference

OOP / JavaScript Mixins

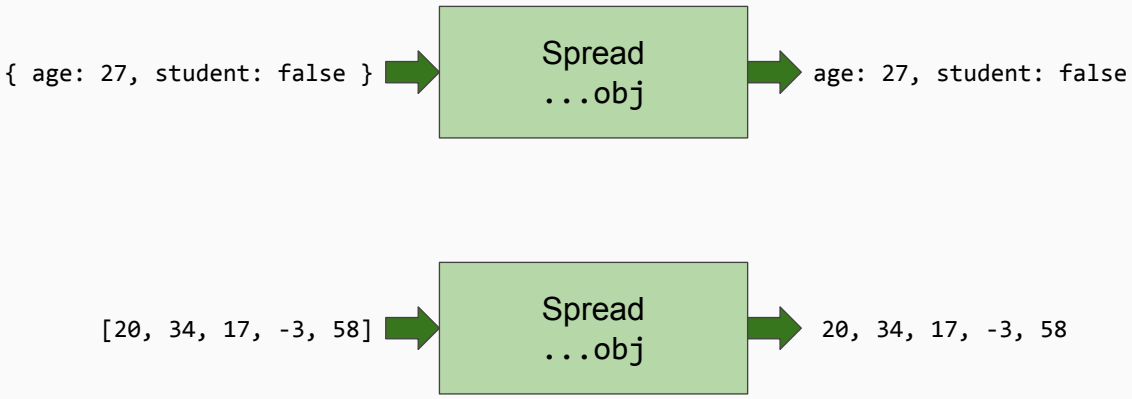
- Traditional class enhancement techniques
  - Inherit a (base) class and add new functionalities in the child class
- Mixins: *functions added to an object* to enhance its capabilities
- Example

```
var obj = { used : false, count : 5 };

obj.inc = function() { // function added AFTER the object is created
  this.count++;
}
```

Mixins Use Cases

- Adding your own function at *runtime* to your own object seems to be ***a bad design*** decisision
  - The function should have been added when the class was designed in the first place
- **A more realistic use case**: add new functions from 3rd party libraries to your objects
  - Objective: enhance your object with capabilities provided by the library
- Two options of Mixin implementation
  - JavaScript spread (`...obj`)
  - `Object.assign()`

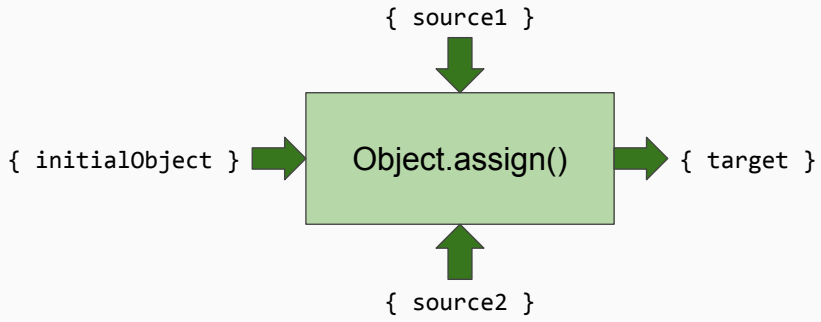
JavaScript Spread (`...obj`)



JavaScript Object.assign()

```
target = Object.assign (initialObject, src(s), ...);
```

- *Properties in initialObject are replaced with properties in the sources (if they have the same key).*
- *Properties of later sources replace earlier ones*



Using JavaScript "spread" and `Object.assign()`

```
var simple   = { used: false, count: 5 };
var enhanced = { ...basic, unit : "px" };

// enhanced is {used: false, count: 5, unit: "px"}
simple.count = 7;

console.log(enhanced.count); // output 5  (NOT 7)
console.log(enhanced.unit);  // output "px"
```

```
var simple = { used: false, count: 5 };
var enhanced = Object.assign(
   {},
   basic,
   { unit : "px"}   /* enhancement */
);
```

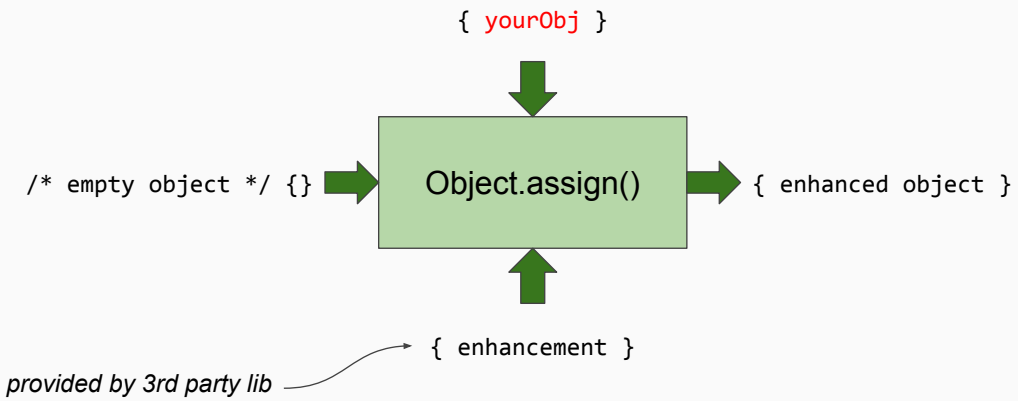
Using JavaScript `Object.assign()`

```
var simple = { used: false, count: 5 };

var enhanced = Object.assign(
   {},
   simple,
   { unit : “px” }   /* enhancement is an attribute */
);
```

```
var simple = { used: false, count: 5 };

var enhanced = Object.assign(
   {},
   simple,
   { inc : function() { this.count += 2; } }  /* enhancement is a function */
);
```

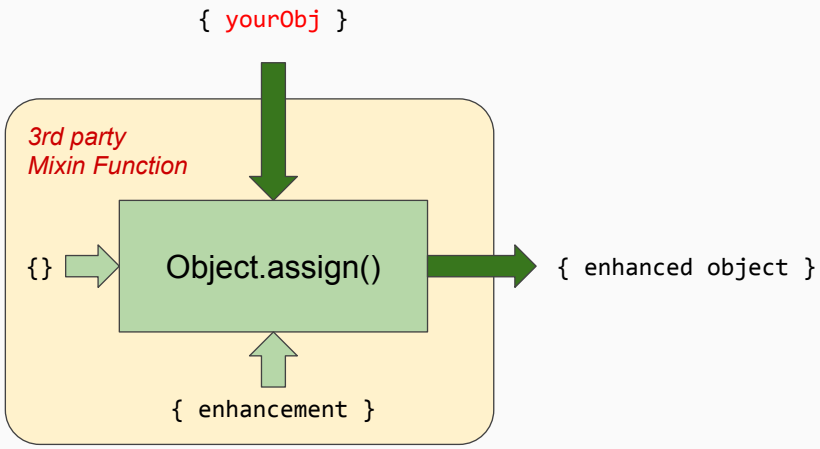

Mixin by 3rd party libraries

```
enhancedObj = Object.assign ( {}, yourObj, external-enhancement )
```



Mixin by 3rd party libraries

```
enhancedObj = MixinByThirdPartyLib (yourObj)
```



<cust-om></cust-om>
vs.
<cust-el>*contents*</cust-el>

<tag-me>*child contents*</tag-me>

```
<ol>
  <li>First Item</li>
  <li>Second Item</li>
  <li>Third Item</li>
</ol>
```

How does <ol> assign the ordinal numbers?

```
<form action="____">
  User: <input type="text" name="uname">
  Password: <input type="password" name="upass">
  _____
</form>
```

How does <form> know that uname and upass must be bundled together?

<slot></slot>

- Use <slot> to allow the child contents of your custom element to render inside the element's shadow DOM
- Practical applications
  - Tabbed pages
  - Coordinate scrolling of contents and page header/page footer
  - **General**: any contents too big to pass as attribute value
- Type of <slot>: Default (unnamed) slots & named slots
- **Flatenning**: the result of *distributing* child contents into shadow DOM

Default <slot>

```
<!-- in cus-tom.html -->
<template>
  <span>Before</span>
  <slot></slot>
  <span>After</span>
</template>
```

```
<!-- in index.html -->
<body>
  <cus-tom>
  <p>Hello</p>
  </cus-tom>
</body>
```

```
<!-- flattenned DOM -->
<body>
  <cus-tom>
    # shadow-root
    <span>Before</span>
    <slot>
      <p>Hello</p>
    </slot>
    <span>After</span>
  </cus-tom>
</body>
```


Named Slots <slot name="_____">

```
<!-- in cus-tom.html -->
<template>
  <slot name="first"></slot>
  <span>Between</span>
  <slot></slot>
</template>
```

```
<!-- in index.html -->
<body>
  <cus-tom>
  <h1 slot="first">Intro</h1>
  <p slot="first">Hello</p>
  <p>Last one<p>
  </cus-tom>
</body>
```

```
<!-- flattenned DOM -->
<body>
  <cus-tom>
    # shadow-root
    <slot name="first">
      <h1 slot="first">Intro</h1>
      <p slot="first">Hello</p>
    </slot>
    <span>Between</span>
    <slot>
      <p>Last one</p>
    </slot>
  </cus-tom>
</body>
```