Workshop: Writing Reusable Modules

Goal of the Workshop

- Consolidate your understanding of writing modular d3 visualizations
- Practice building your own d3 module
- Putting modules together

Why Build Visualization Modules

Basic principles

- Repeatable
- Modifiable
- Configurable
- Extensible

Outline of Thought Process (1)

In order to re-use a block of code, we need to encapsulate it within a function.

This allows us to call the function later on a d3 selection, like so:

```
//customChart is a function
selection
    .datum(...)
    .call(customChart);
```

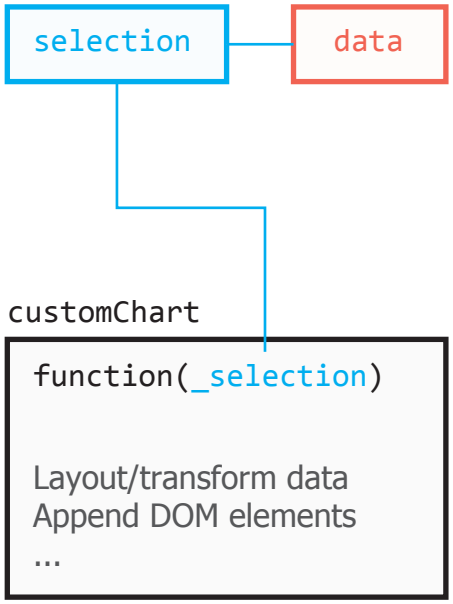
Be sure the review the documentation for `selection.call(...)`

Outline of Thought Process (2)

```
# selection.call(function[, arguments…])
```

Invokes the specified function once, passing in the current selection along with any optional arguments. The call operator always returns the current selection, regardless of the return value of the specified function. The call operator is identical to invoking a function by hand; but it makes it easier to use method chaining. For example, say we want to set a number of attributes the same way in a number of different places.

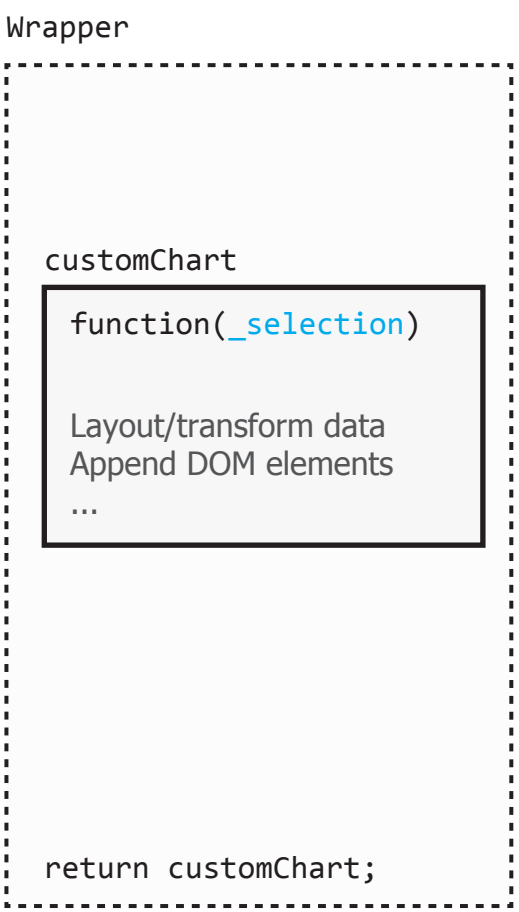
Outline of Thought Process (3)




```
selection
    .datum(...)
    .call(customChart);
```

But this implementation of `customChart` isn't configurable! i.e. we can't change its attributes!

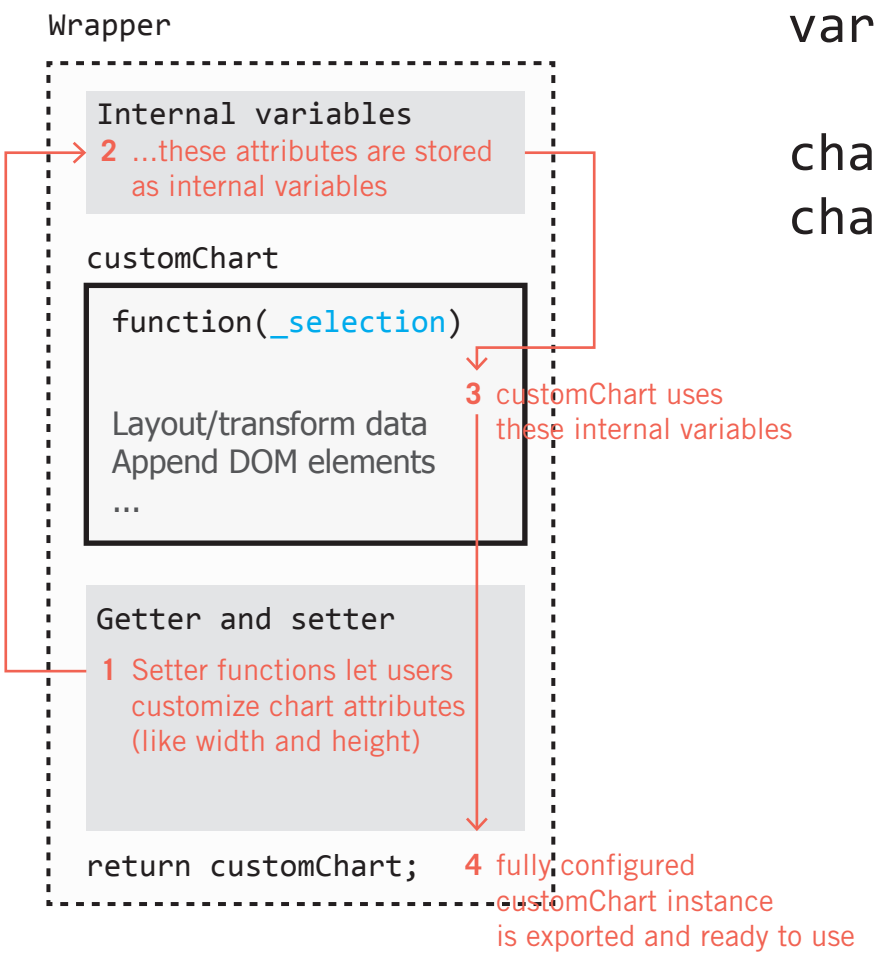
Outline of Thought Process (4)



The solution: wrap `customChart` in a `wrapper` function.

Use the `wrapper` function to configure, and export an instance of, `customChart`

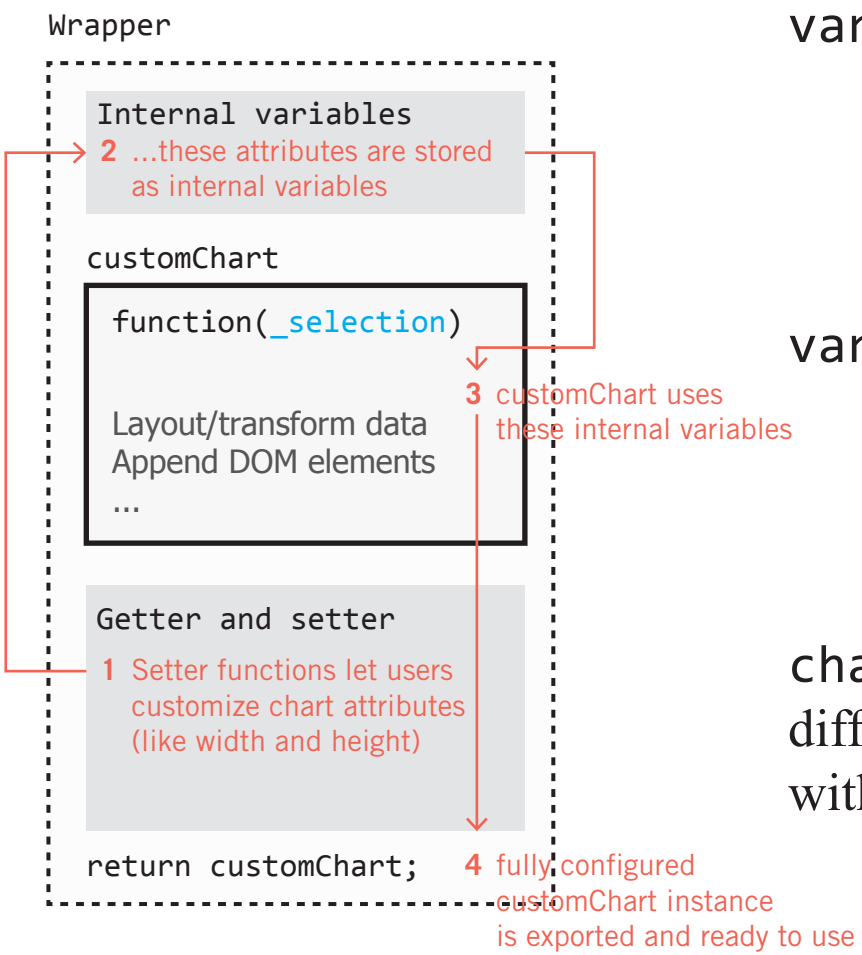
Outline of Thought Process (5)




```
var chart1 = wrapper();

chart1.width(400);
chart1.height(600);
```

Outline of Thought Process (6)

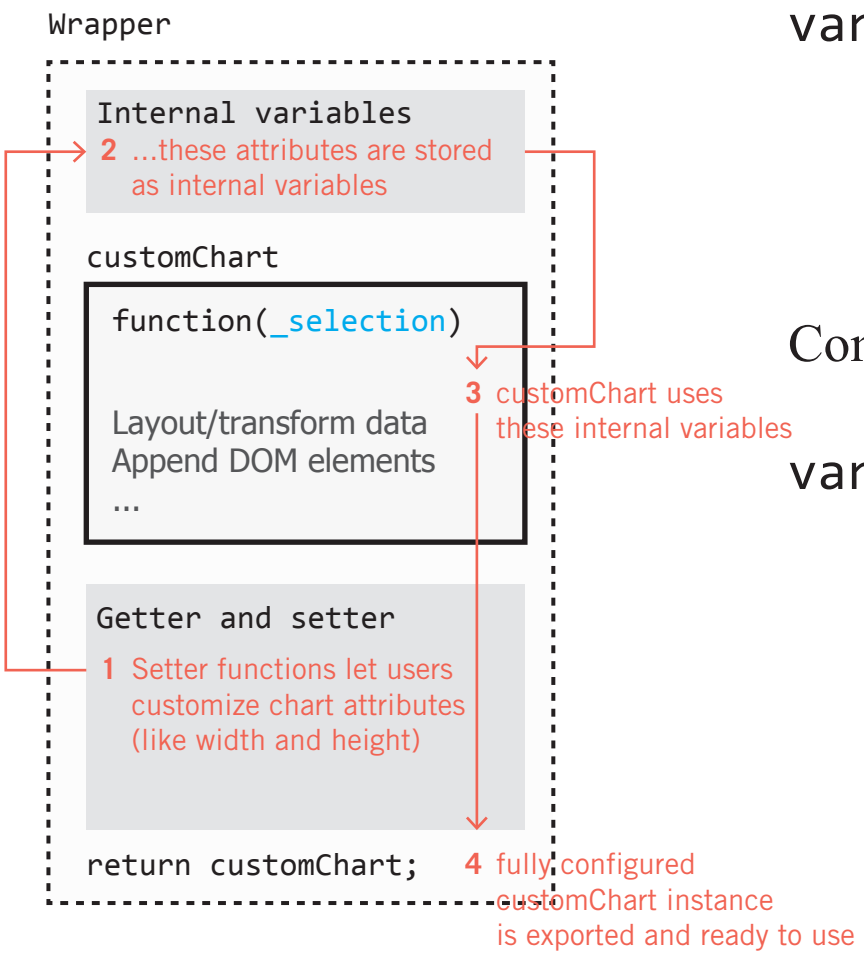



```
var chart1 = wrapper()
    .width(400)
    .height(600)
    ...

var chart2 = wrapper()
    .width(300)
    .height(400)
    ...
```

`chart1` and `chart2` are two different instances of `customChart`, with different configurations.

Additional Considerations: Chaining




```
var chart1 = wrapper()
    .width(400)
    .height(600)
    ...
```

Compare this to something familiar:

```
var axisX = d3.svg.axis()
    .scale(...)
    .ticks(...)
```

Additional Considerations: Updates to the Chart

```
var chart1 = wrapper()
    .width(400)
    .height(600);
d3.select('div.chart')
    .datum(someData)
    .call(chart1);

chart1.width(500);
d3.select('div.chart')
    .datum(newData)
    .call(chart1);
```

If chart configuration changes, or new data is bound, we should be able to make updates by simply calling the function again.

Our implementation of the module should support this behavior.

Defining Our Own API

Crucially, our design of the module makes important assumptions:

- The form of the data
- The nature of the d3 selection (? <svg>? <g>?)
- The allowed arguments for the getter/setter functions

Our API must make these assumptions explicit!

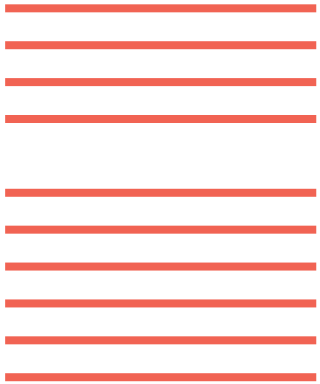
Application Programming Interface (API)

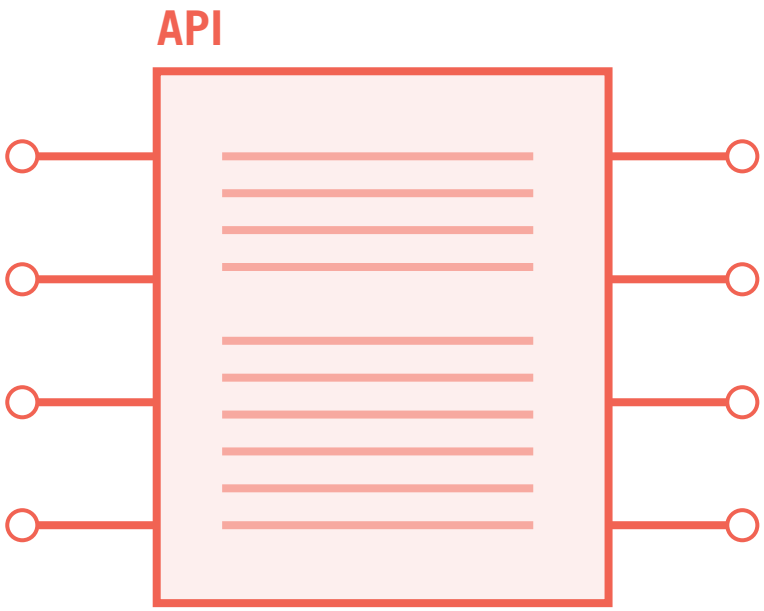
Defines a software component in terms of its

- Operations i.e. what does it do
- Inputs
- Outputs
- Underlying type

These are indepent of the component's underlying implementation (you don't need to know it!)

Implementation




API

Time Series Module