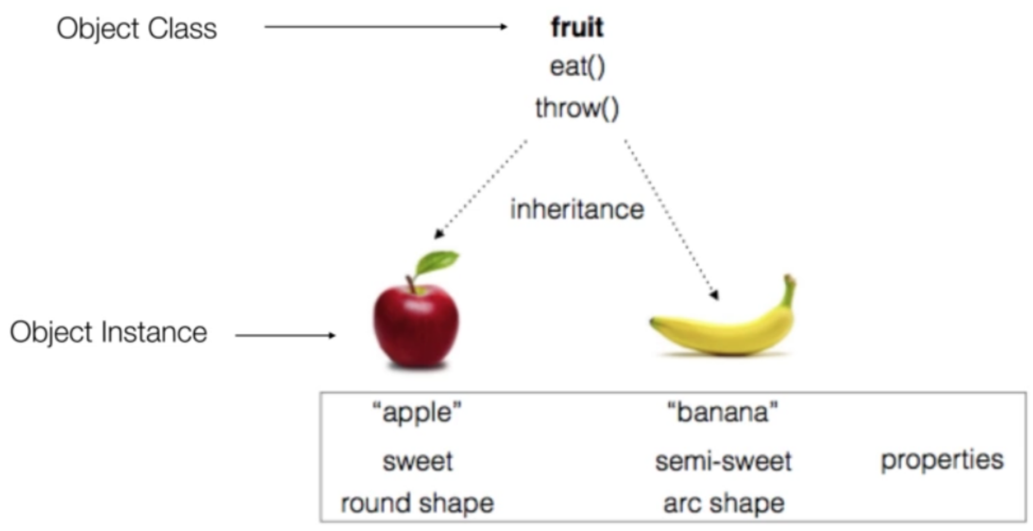
File Structure and Linked Views

After adding a lot of different event listeners, the JavaScript file can become messy. This section focuses on writing readable code in an 'Object Oriented' way for larger projects (but OOP will not be covered in depth here). Once a class is set up for a visualization, it can be easily reused.



The idea to have a main JS file and then a separate file for each visualization as a class. Keep in mind file order matters when loading scripts in HTML, so the main file should be loaded last.

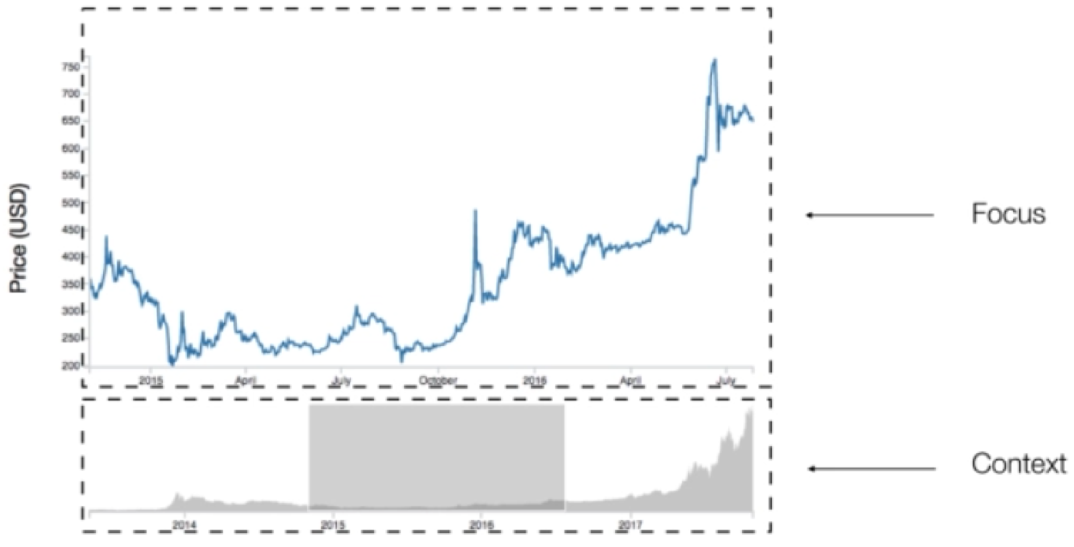
I'm going to write the following examples in JavaScript ES6+, but TypeScript will also work. The big difference is JavaScript only supports local and global objects, while TypeScript's classes are more akin to a language like Java.

```
// from barChart.js
class BarChart{
    constructor(_parentElement, data) {
        this.parentElement = _parentElement
        this.data = _data
    }

    // class method
    initVis() {
        const vis = this
        vis.WIDTH = 250
        vis.HEIGHT = 100
            ...
    }
}


// from main.js
const barChart = new BarChart("#bar-chart-area", data) // new class instance
barChart.initVis()
```

By using an object oriented manner, it it easier to create multiple of the same graphic without duplicating much of the same code, and allows us to create multiple objects that react to a single event.

Brushes

D3 Brushes are used for selecting an area of a visualization; For example, adding a context graph beneath our line chart that allows the user to zoom in or out.



There's three steps to adding brush behaviour to an HTML or SVG element:

- call `d3.brush()` to create a **brush behavior** function
- add an event handler that's called when a brush event occurs. The event handler receives the brush extent which can then be used to select elements, define a zoom area etc.
- attach the brush behavior to an element(s)

```
let data = [], width = 600, height = 400, numPoints = 100;


let brush = d3.brush()
.on('start brush', handleBrush);
```

```
let brushExtent;

function handleBrush(e) {
  brushExtent = e.selection;
  update();
}

function initBrush() {
  d3.select('svg g')
    .call(brush);
}

function updateData() {
  data = [];
  for(let i=0; i<numPoints; i++) {
    data.push({
      id: i,
      x: Math.random() * width,
      y: Math.random() * height
    });
  }
}

function isInBrushExtent(d) {
  return brushExtent &&
    d.x >= brushExtent[0][0] &&
    d.x <= brushExtent[1][0] &&
    d.y >= brushExtent[0][1] &&
    d.y <= brushExtent[1][1];
}

function update() {
  d3.select('svg')
    .selectAll('circle')
    .data(data)
    .join('circle')
    .attr('cx', function(d) { return d.x; })
    .attr('cy', function(d) { return d.y; })
    .attr('r', 4)
    .style('fill', function(d) {
      return isInBrushExtent(d) ? 'red' : null;
    });
}


initBrush();
updateData();
update();
```

For more info check out this d3indepth article on interaction.

Revision #2
Created 7 September 2023 19:45:32 by Elkip
Updated 3 October 2023 19:59:24 by Elkip