Classes in JavaScript

Amateur JavaScript

Your code may looks like this:

- Mostly all in one file
- All global functions
- Global variables to save state between events

It would be nice to write code in a **modular** way...

```
//
// Album view functions
//
let currentIndex = null;
function onThumbnailClick(event) {
  currentIndex = event.currentTarget.dataset.index;
  const image = createImage(event.currentTarget.src);
  showFullsizeImage(image);
  document.body.classList.add('no-scroll');
  modalView.style.top = window.pageYOffset + 'px';
  modalView.classList.remove('hidden');
}

//
// Photo view functions
//
function createImage(src) {
  const image = document.createElement('img');
  image.src = src;
  return image;
}

function showFullsizeImage(image) {
  modalView.innerHTML = '';

  image.addEventListener('pointerdown', startDrag);
  image.addEventListener('pointermove', duringDrag);
  image.addEventListener('pointerup', endDrag);
  image.addEventListener('pointercancel', endDrag);
  modalView.appendChild(image);
}

let originX = null;
function startDrag(event) {
  event.preventDefault();
  // Needed so clicking on picture doesn't cause modal dialog to close.
  event.stopPropagation();

  originX = event.clientX;
  event.target.setPointerCapture(event.pointerId);
}

function duringDrag(event) {
  if (originX) {
    const currentX = event.clientX;
    const delta = currentX - originX;
    const element = event.currentTarget;
    element.style.transform = 'translateX(' + delta + 'px)';
  }
}

function endDrag(event) {
  if (!originX) {
    return;
  }

  const currentX = event.clientX;
  const delta = currentX - originX;
  originX = null;

  let nextIndex = currentIndex;
  if (delta < 0) {
    nextIndex++;
  } else {
    nextIndex--;
  }

  if (nextIndex < 0 || nextIndex == PHOTO_LIST.length) {
    event.currentTarget.style.transform = '';
    return;
  }
```

ES6 classes

We can define **classes** in JavaScript using a syntax that is similar to Java or C++:

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

These are often called "**ES6 classes**" or "**ES2015 classes**" because they were introduced in the EcmaScript 6 standard, the 2015 release

- Recall that EcmaScript is the standard; JavaScript is an implementation of the EcmaScript standard

Wait a minute…

Wasn't JavaScript created in 1995?

And classes were introduced… 20 years later in 2015?

Q: Was it seriously not possible to create classes in JavaScript before 2015?!

Objects in JavaScript

In JavaScript, there are several ways to create blueprints for objects. Two broad approaches:

1. Functional
    a. This approach has existed since the creation of the JavaScript
    b. Weird syntax for people used to languages like Java, C++, Python
    c. Doesn't quite behave the same way as objects in Java, C++, Python
2. Classical
    a. This is the approach that just got added to the language in 2015
    b. Actually just "syntactic sugar" over the functional objects in JavaScript, so still a little weird
    c. But syntax is much more approachable

Objects in JavaScript

In JavaScript, there are several ways to create blueprints for objects. Two broad approaches:

1. **Functional**
    a. This approach has existed since the creation of the JavaScript
    b. Weird syntax for people used to languages like Java, C++, Python
    c. Doesn't quite behave the same way as objects in Java, C++, Python
2. **Classical**
    a. This is the approach that just got added to the language in 2015
    b. Actually just "syntactic sugar" over the functional objects in JavaScript, so still a little weird
    c. But syntax is much more approachable

This approach is quite controversial.

Class controversy

"There is one thing I am certain is a bad part, a very terribly bad part, and that is the new class syntax [in JavaScript]... [T]he people who are using `class` will go to their graves never knowing how miserable they were." (source)

-- Douglas Crockford, author of *JavaScript: The Good Parts*; prominent speaker on JavaScript; member of TC39 (comm
that makes ES decisions)

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

`constructor` is optional.

Parameters for the constructor and methods are defined in the same they are for global functions.

You do not use the `function` keyword to define methods.

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodOne() {
    this.methodTwo();
  }
  methodTwo() {
    ...
  }
}
```

Within the class, you must always refer to other methods in the class with the **`this.`** prefix.

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

All methods are **public**, and you **cannot** specify private methods… yet.

Public fields

```
class ClassName {

  constructor(params) {
    this.fieldName = fieldValue;
    this.fieldName = fieldValue;
  }

  methodName() {
    this.fieldName = fieldValue;
  }
}
```

Define public fields by setting **`this.`*fieldName*** in the constructor… or in any other function.

Class fields : ES2019

```
class MyClass {
    name = "toto";

    constructor(age) {
        this.age = age;
    }
}
```

Class fields are public

Only since ES2019 - check compatibility!

Public fields

```
class ClassName {
  constructor(params) {
    this.someField = someParam;
  }
  methodName() {
    const someValue = this.someField;
  }
}
```

Within the class, you must always refer to fields with the `this.` prefix.

Private fields : ES2019

```
class MyClass {
    #name = "toto";
    constructor(age) {
        this.age = age;
    }
    display() {
        console.log(this.#name + " : " + this.age);
    }
}
let m = new MyClass(22);
m.display();
// console.log(m.#name + " : " + m.age);  // ERROR
```

Only since ES2019 - check compatibility!

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/Private_class_fields

Getter and setter : since 2015

```
class Lang {
    log = [];
    set current(name) {
        this.log.push(name);
    }
    get latest() {
        if (this.log.length === 0) {
          return undefined;
        }
        return this.log[this.log.length - 1];
    }
}
```

```
let lang = new Lang();
console.log(lang.latest);
lang.current = 'FR';
lang.current = 'EN';
console.log(lang.latest);
console.log(lang.log);
```

Ouvrons nos cadeaux ....

index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Simple class: present</title>
    <script src="scripts/main.js" defer></script>
    <style>img { width: 200px; }</style>
  </head>
  <body>
    <div id="presents"></div>
  </body>
</html>
```

Ouvrons nos cadeaux ....

scripts > JS main.js > ...

```
1   class Present {
2       constructor(containerElement) {
3           this.containerElement = containerElement;
4
5           // Create image and append to container.
6           const image = document.createElement('img');
7           image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
8           image.addEventListener('click', this.openPresent);
9           this.containerElement.append(image);
10      }
11
12      openPresent(event) {
13          const image = event.currentTarget;
14          image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
15          image.removeEventListener('click', this.openPresent);
16      }
17  }
18
19  const presents = document.querySelector('#presents');
20  const present = new Present(presents);
```

Don't forget `this`

scripts > JS main.js > ...

```
1   class Present {
2       constructor(containerElement) {
3           this.containerElement = containerElement;
4
5           // Create image and append to container.
6           const image = document.createElement('img');
7           image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
8           image.addEventListener('click', this.openPresent);
9           this.containerElement.append(image);
10      }
11
12      ope
13
14                                                              ;
15
16      }
17  }
18
19  const presents = document.querySelector('#presents');
20  const present = new Present(presents);
```

If the event handler function you are passing to `addEventListener` is a method in a class, you must pass "`this.`***functionName***"

`this` in event handler

```
12      openPresent(event) {
13          const image = event.currentTarget;
14          image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
15          image.removeEventListener('click', this.openPresent);
16      }
```

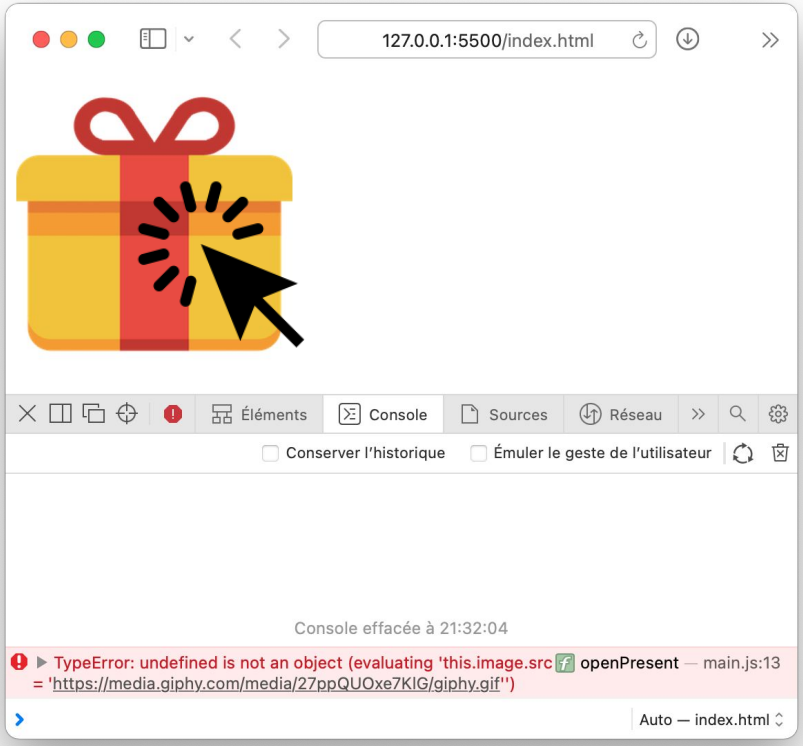
Right now we access the image we create in the constructor in **`openPresent`** via **`event.currentTarget`**

`this` in event handler

```
12      openPresent(event) {
13          this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
14          this.image.removeEventListener('click', this.openPresent);
15      }
```

Q: What if we make the `image` a field and access it in `openPresent` via `this.image` instead of `event.currentTarget`?

`this` in event handler



Error message!

What's going on?

JavaScript `this`

The `this` keyword in JavaScript is **dynamically assigned**, or in other words: `this` means different things in different contexts (mdn list)

- In our constructor, `this` refers to the instance
- When called in an event handler, `this` refers to… the element that the event handler was attached to (mdn).

`this` in event handler

```
12      openPresent(event) {
13          this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
14          this.image.removeEventListener('click', this.openPresent);
15      }
```

That means `this` refers to the `<img>` element, not the instance variable of the class...

...which is why we get this error message.

Solution: `bind`

To make `this` always refer to the instance object for a method in the class (i.e. to get `this` to behave as you'd expect), you can add the following line of code in the `constructor`:

```
this.methodName = this.methodName.bind(this);
```

scripts > JS main.js > Present > constructor

```
1   class Present {
2       constructor(containerElement) {
3           this.containerElement = containerElement;
4
5           // Bind event listeners
6           this.openPresent = this.openPresent.bind(this);
```

Solution: `bind`

Now **`this`** in the **`openPresent`** method refers to the instance object

```
12      openPresent(event) {
13          this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
14          this.image.removeEventListener('click', this.openPresent);
15      }
```

Moral of the story:

Don't forget to `bind()` event listeners in your constructor!!

scripts > JS main.js > Present > constructor

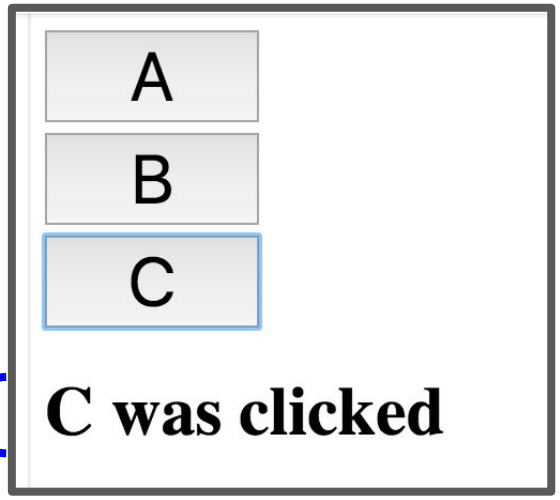
```
1   class Present {
2       constructor(containerElement) {
3           this.containerElement = containerElement;
4
5           // Bind event listeners
6           this.openPresent = this.openPresent.bind(this);
```

One more time:

Don't forget to `bind()` event listeners in your constructor!!

Example: Buttons

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Menu and buttons examples</title>
  </head>
  <body>
    <div id="menu"></div>
    <h1 id="status-bar"></h1>
  </body>
</html>
```



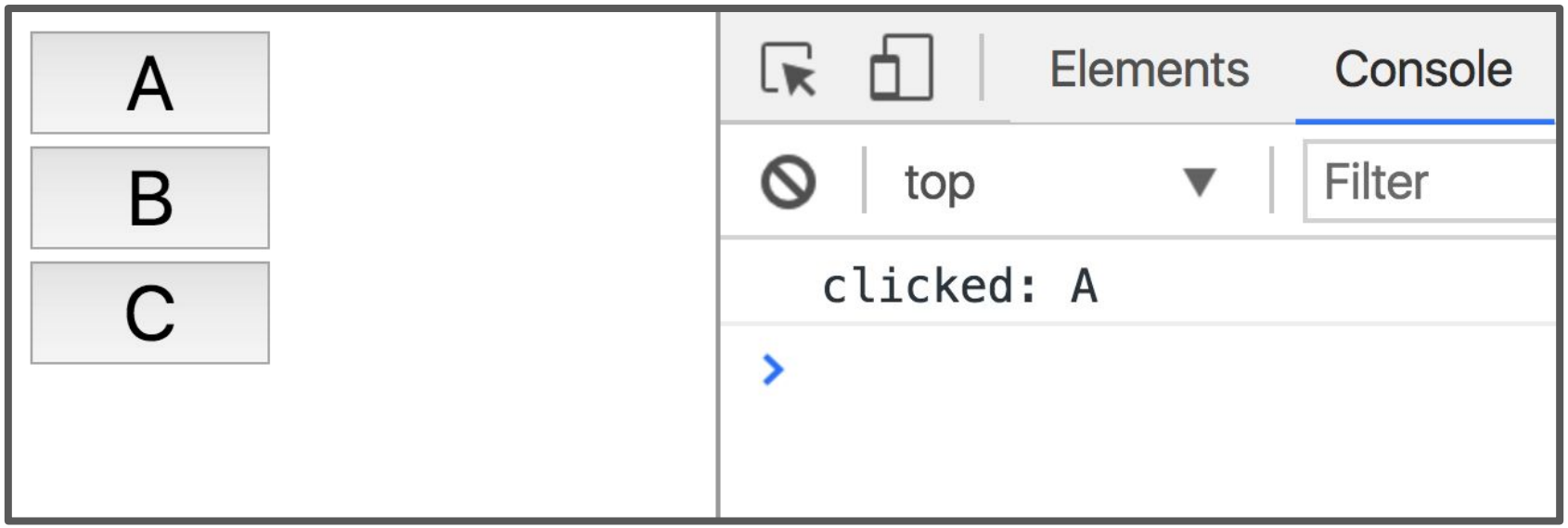
We want to:

- Fill the `<div id="menu"></div>` with buttons A, B, and C
- Update the `<h1>` with the button that was clicked
- Live example

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;

    const button = document.createElement('button');
    button.textContent = text;
    this.containerElement.append(button);
  }
}

const buttonContainer = document.querySelector('#menu');
const button1 = new Button(buttonContainer, 'A');
const button2 = new Button(buttonContainer, 'B');
const button3 = new Button(buttonContainer, 'C');
```

First step: Create a `Button` class and create three `Button`s. (CodePen)

Click handler for Button



Let's make it so that every time we click a button, we print out which button was clicked in the console. (Live)

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;

    const button = document.createElement('button');
    button.textContent = text;
    this.containerElement.append(button);
  }
}
```

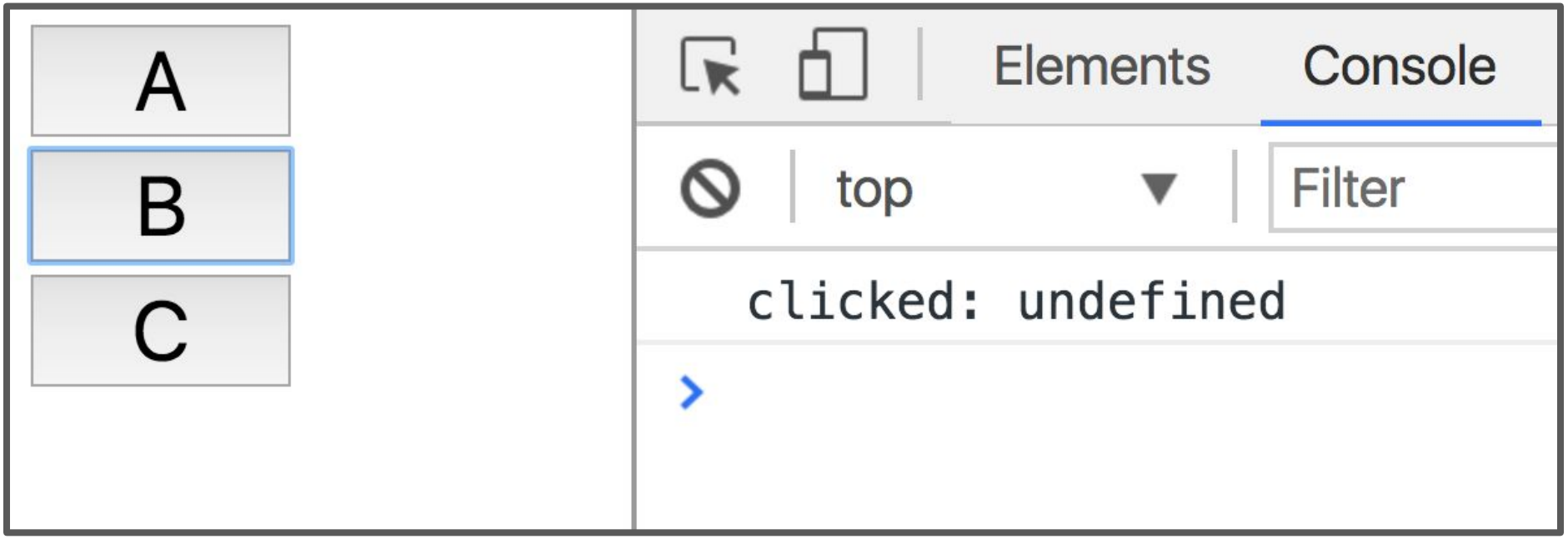
Starting with this definition of Button...

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

An initial attempt might look like this. (CodePen)

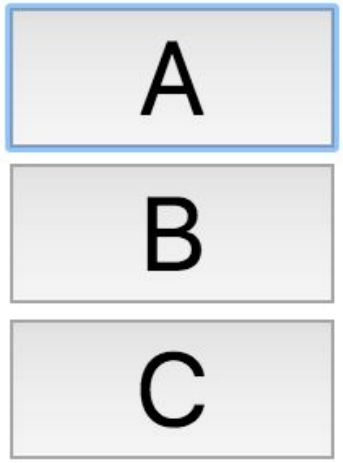
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

An initial attempt might look like this. (CodePen)



But when we run it, that gives us "clicked: undefined" (CodePen) **Why?**

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    console.log(this);
  }
}
```




```
clicked: undefined
    <button>A</button>
>
```

That's because the value of `this` in `onClick` is not the `Button` object; it is the `<button>` element to which we've attached the `onClick` event handler.

What?!?

`this` in JavaScript

`this` in the constructor

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
}
```

In the constructor of a class, `this` refers to the new object that is being created.

That's the same meaning as `this` in Java or C++.

`this` in the constructor

```
// Java
public class Point {
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int x;
  public int y;
}
```

Here's roughly the equivalent code in Java. `this` refers to the new object that is being created.

`this` in Java

```
// Java
public class Point {
   ...

   String toString() {
     return this.x + ", " + this.y;
   }
}
```

In Java, `this` **always** refers to the new instance being created, no matter what method you're calling it from, or how that method is invoked.

`this` in JavaScript

```
class Point {
  ...

  toString() {
    return this.x + ", " + this.y;
  }
}
```

But in JavaScript, `this` **can have a different meaning** if used outside of the constructor, depending on the **context** in which the function is called.

`this` in JavaScript

```
toString() {
  return this.x + ", " + this.y;
}
```

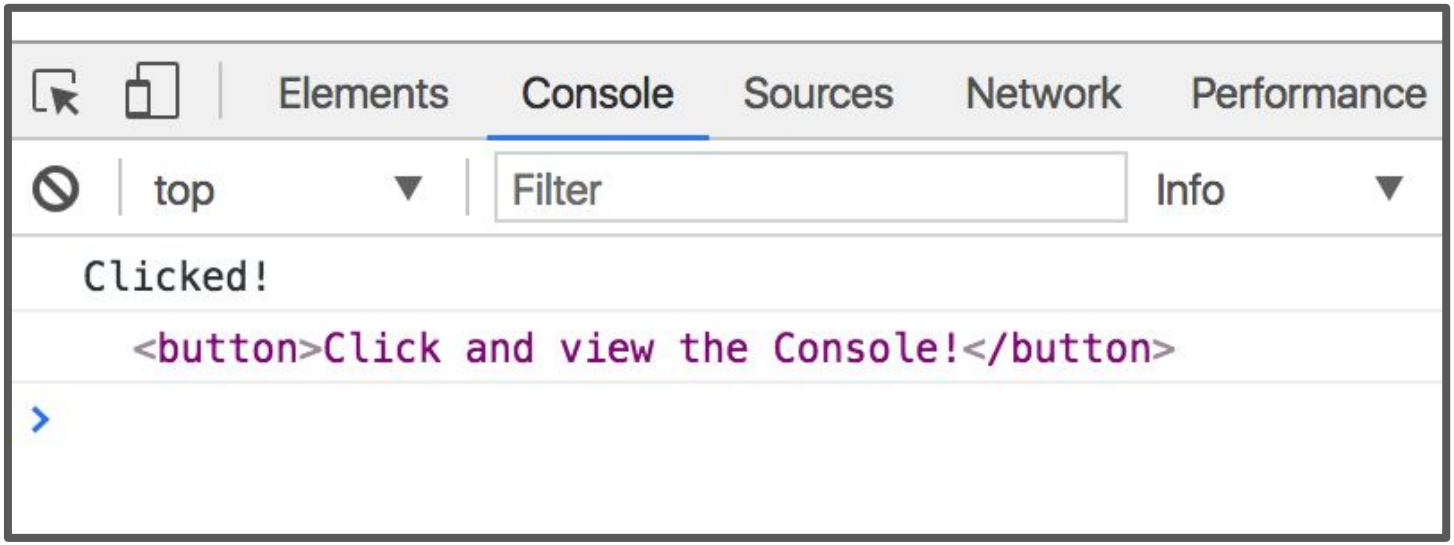
In JavaScript, `this` is:

- A implicit **parameter** that is passed to **every JavaScript function**, including functions not defined in a class!
- The value of the `this` parameter changes depending on how it is called.

`this` in `addEventListener`

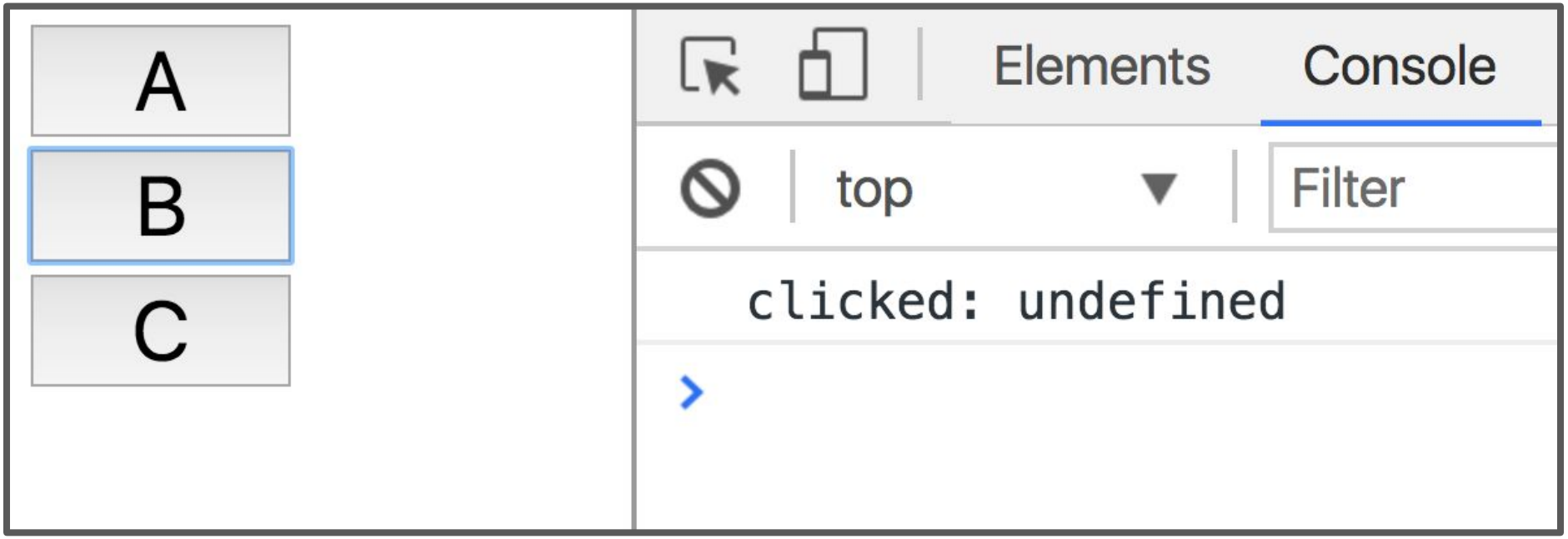
```
function onClick() {
  console.log('Clicked!');
  console.log(this);
}

const button = document.querySelector('button');
button.addEventListener('click', onClick);
```

When used in an event handler, `this` is set to the **element to which that the event was added.** (mdn / CodePen / live)




```
function onClick() {
  console.log('Clicked!');
  console.log(this);
}
const button = document.querySelector('button');
button.addEventListener('click', onClick);
```

In `onClick`, this refers to `<button>` because it `onClick` was invoked by `addEventListener`.



Let's revisit our undefined text… (CodePen)

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }
```

In the constructor, `this` refers to the new object we're creating. No problems here.

```
  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

But in `onClick`, `this` will mean something different depending on how the function is called.

That is because we are using `this` in a function that is **not** a constructor.

```
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

Specifically, because `onClick` is attached to the `<button>` via `addEventListener`...

```
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

...we know the value of `this` will be the `<button>` element when the click event is fired and invokes `onClick`.

Since `HTMLButtonElement` doesn't have a `text` property, `this.text` is `undefined`.

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;
```

...

```
  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

It'd be nice if we could set the value of "`this`" in `onClick` to be the `Button` object, like it is in the constructor.

"Bind" the value of `this`

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);
```

That is what this line of code does:

"Hey, use the current value of `this` in `onClick`"

(And the current value of this is the new object, since we're in the constructor)

CodePen / Live

`bind` in classes

```
constructor() {
  const someValue = this;
  this.methodName = this.methodName.bind(someValue);
}
```

This is saying:

- Make a copy of ***methodName***, which will be the exact same as ***methodName*** except `this` in ***methodName*** is always set to the `someValue`
- The value of `someValue` is `this` to `bind()`, which is the value of the new object since we are in the constructor

`bind` in classes

```
constructor() {
  this.methodName = this.methodName.bind(this);
}
```

And of course, you don't need the intermediate `someValue` variable.

CodePen / Live

One more time...

`this` in the constructor

`this` in the constructor refers to the new object you are creating.

```
constructor(x, y) {
  this.x = x;
  this.y = y;
}
```

`this` in a function

`this` in a function that is **not** a constructor has a different value, depending on **how the function is called**.

```
onClick() {
  console.log(this.x);
  console.log(this.u);
}
```

- When invoked as a response to an event, the `this` in `onClick` will be `Event.targetElement`, or the element onto which the `onClick` event handler was attached.

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

Right now, `this` in the constructor always refers to the new object we're creating...

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

But `this` in `onClick` function refers to a different value, depending on how onClick is called.

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

It'd be nice if we could make the "`this`" value in `onClick`:

- Refer to the new object we're constructing, instead of things like the dom element, etc
- And make it always refer to the new object we're constructing

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(someParam);
  }
```

That's what **`bind`** does:

- It is saying, "Hey, in the `onClick` function, I want the `this` value to always be ***someParam***," i.e. the value that we are passing as a parameter to `bind`.

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(someParam);
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

We want the value of `this` in `onClick` to be the value of the new object being created.

In other words, we want ***someParam*** to be the value of the new object being created.

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(someParam);
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

In the constructor, how do we access the new object being created?

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(this);
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

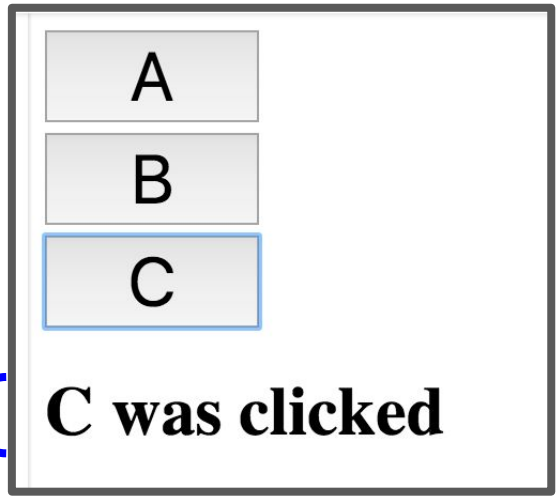
In the constructor, the new object is referenced by `this`.

Now the `this` in `onClick` always referring to the new object.

What were we
trying to do again?

Example: Buttons

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Menu and buttons examples</title>
  </head>
  <body>
    <div id="menu"></div>
    <h1 id="status-bar"></h1>
  </body>
</html>
```

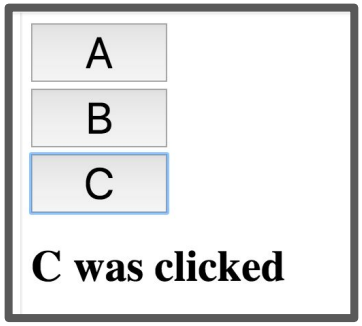


We want to:

- Fill the `<div id="menu"></div>` with buttons A, B, and C
- Update the `<h1>` with the button that was clicked
- Live example

(Contrived) OO example

For practice, we'll write this using 2 classes:

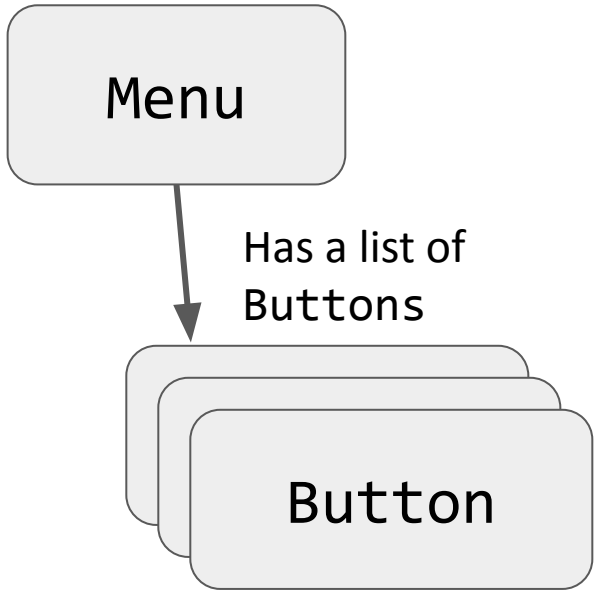


`Menu`:

- Has an array of `Buttons`
- Also updates the <h1> with what was clicked

`Button`:

- Notifies `Menu` when clicked, so that `Menu` can update the <h1>



```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }
}
```

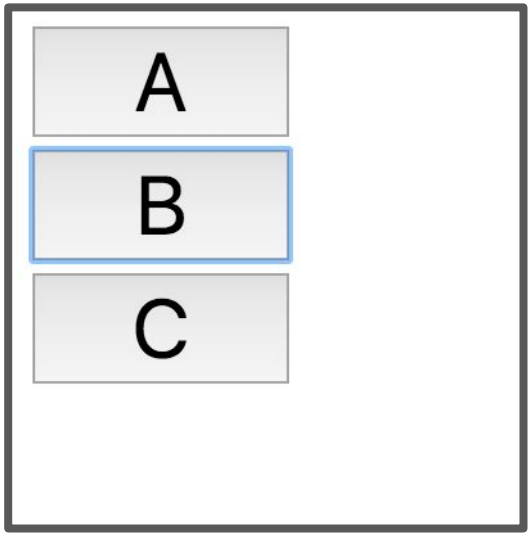
Partial solution: We create a Menu class, which creates the Buttons (CodePen)

```
const menu = new Menu();
```

Then we create the Menu (and the menu creates the Buttons) when the page loads. (CodePen)

Update Menu when Button clicked

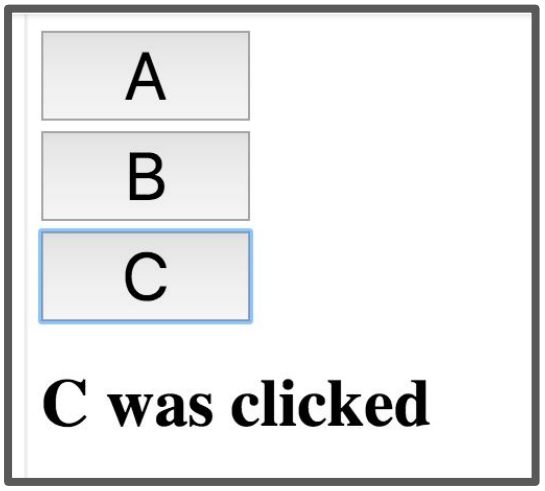
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }
}
```



Our current Menu doesn't do much.

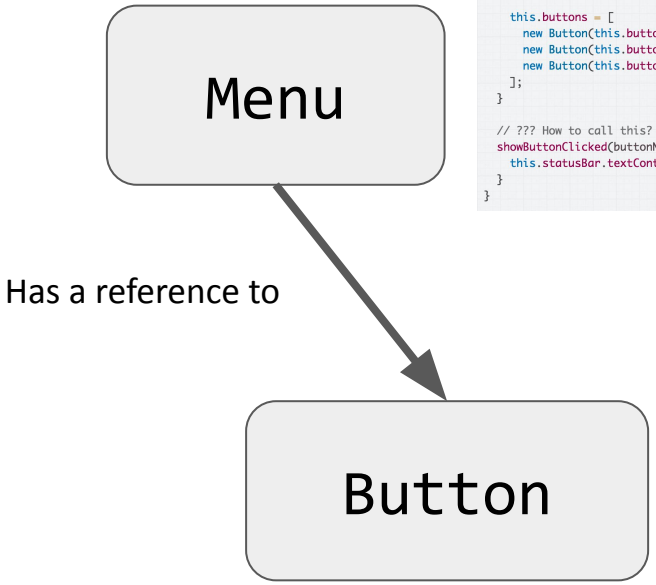
Update Menu when Button clicked

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```



We want the Menu to update the <h1> when one of the Buttons are clicked. **How do we do this?**

Communicating upstream

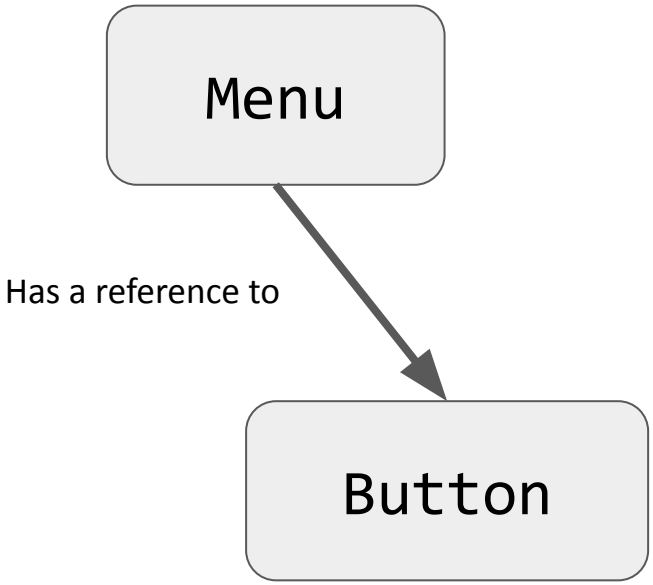



```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

Button is the thing that knows it was clicked...

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

Communicating upstream

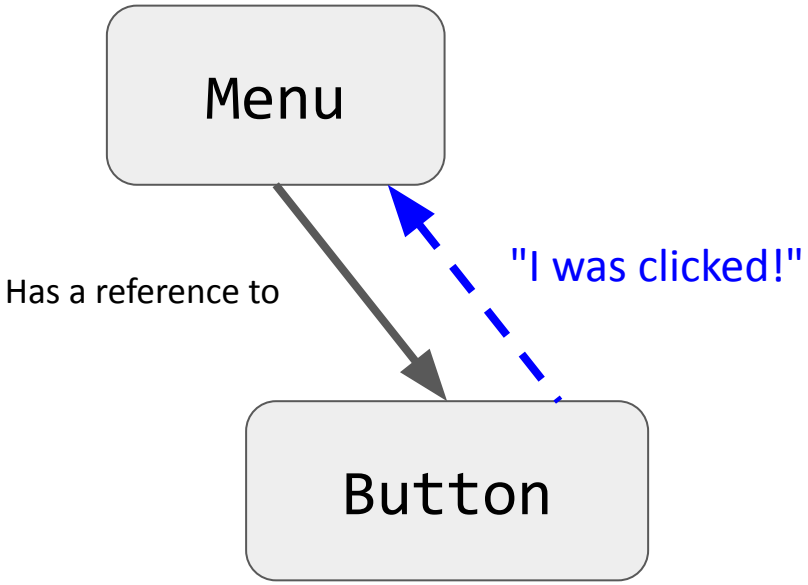



```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

But Menu is the thing that can update the header.

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

Communicating upstream



It needs to be possible for a `Button` to tell the `Menu` that it has been clicked.

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

One strategy for doing this: Custom events

Custom Events

You can listen to and dispatch Custom Events to communicate between classes (mdn):

```
const event = new CustomEvent(
    eventNameString, optionalParameterObject);
element.addEventListener(eventNameString,
functionName);
element.dispatchEvent(eventNameString);
```

Custom Events on `document`

CustomEvent **can only be listened to / dispatched on HTML elements,** and not on arbitrary class instances.

Therefore we are going to be adding/dispatching events on the `document` object, so that events can be globally listened to/dispatched.

```
document.addEventListener(eventNameString, functionName);
document.dispatchEvent(eventNameString);
```

Define a custom event

We'll define a custom event called `'button-click'`:

Menu will listen for the event:

```
document.addEventListener(
    'button-click', this.showButtonClicked);
```

Button will dispatch the event:

```
document.dispatchEvent(
    new CustomEvent('button-click'));
```

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }
}
```

A first attempt: We should listen for the custom `'button-click'` event in `Menu`.

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

A first attempt: Listen for the custom `'button-click'` event in `Menu`. **Note the call to `bind`!** (CodePen)

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

A first attempt: Listen for the custom `'button-click'` event in `Menu`. **Note the call to `bind`!** (CodePen)

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

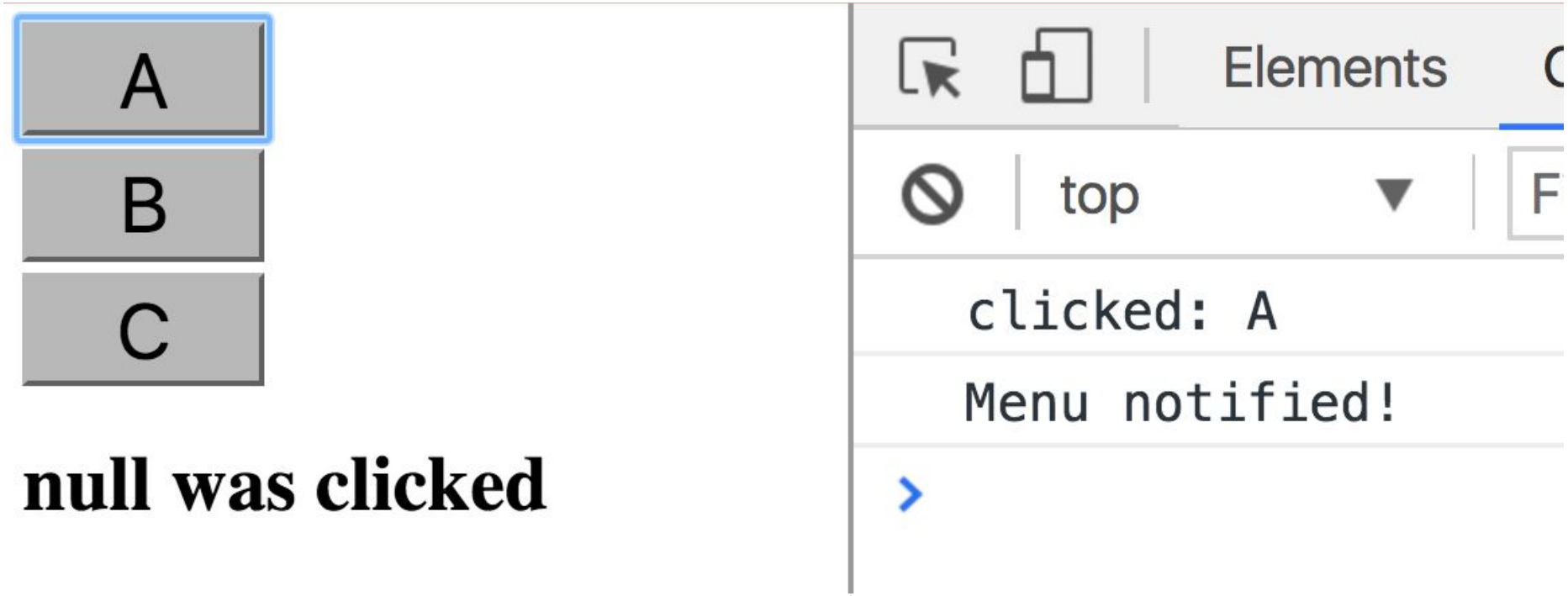
Then we want to dispatch the `'button-click'` event in the `onClick` event handler in `Button`.

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    document.dispatchEvent(new CustomEvent('button-click'));
  }
}
```

Dispatch the `'button-click'` event in the `onClick` event handler in `Button` (CodePen).

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    document.dispatchEvent(new CustomEvent('button-click'));
  }
}
```

Dispatch the `'button-click'` event in the `onClick` event handler in `Button` (CodePen).



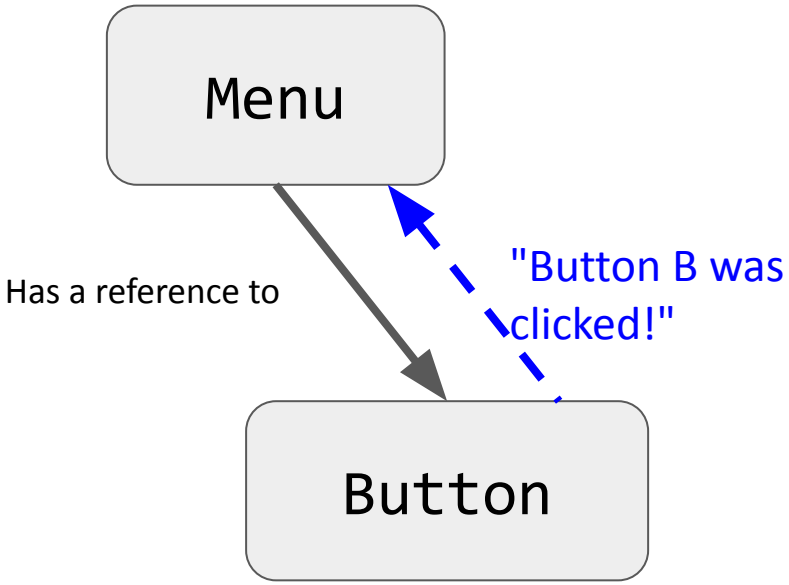
When we try it out, the event dispatching seems to work… but our output is "null was clicked"

(CodePen / Live)

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

The problem is we are adding custom event listeners to `document`, meaning `event.currentTarget` is going to be `document`, and not `<button>`

Communicating upstream



Menu knows some button was clicked… How do we tell the Menu which button was clicked?

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    document.dispatchEvent(new CustomEvent('button-click'));
  }
}
```

`CustomEvent` parameters

You can add a parameter to your `CustomEvent`:

- Create an object with a `detail` property
- The value of this `detail` property can be whatever you'd like.

```
onClick() {
  const eventInfo = {
    buttonName: this.text
  };
  document.dispatchEvent(
      new CustomEvent('button-clicked', { detail: eventInfo }));
}
```

`CustomEvent` parameters

You can add a parameter to your `CustomEvent`:

- The event handler for your `CustomEvent` will be able to access this `detail` property via `Event.detail`

```
    document.addEventListener('button-clicked', this.showButtonClicked);
  }

  showButtonClicked(event) {
    this.statusBar.textContent = event.detail.buttonName + ' was clicked';
  }
}
```

Finished CodePen