Report on

Mixin Design Pattern in JS

Submitted in partial fulfillment of the requirements for ***Sem VI***

UE18CS341 Design Patterns

Bachelor of Technology
in
Computer Science & Engineering

Submitted by Team 28:

Neel Kamath	**PES2201800467**
Naveen K Murthy	**PES2201800051**

Under the guidance of

Prof. N S Kumar
Visiting Professor
PES University, Bengaluru

January – May 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Abstract

Though there are various definitions of mixins, all differing slightly in practical application, a mixin is simply a class that contains methods to be used by other classes.
The Mixin Design Pattern is thus a way of appending various functionality to an existing object.

In more concrete OOP terms, a mixin is an abstract subclass that can be used to create concrete subclasses after modifying/specialising the behavior of existing baseclasses.

It often does this by defining new methods that perform some actions and then calls the corresponding parent methods. In this way, a mixin acts as a subclass factory.

Since JavaScript does not support multiple inheritance natively, mixins can be used to achieve this behaviour.

1. Intent

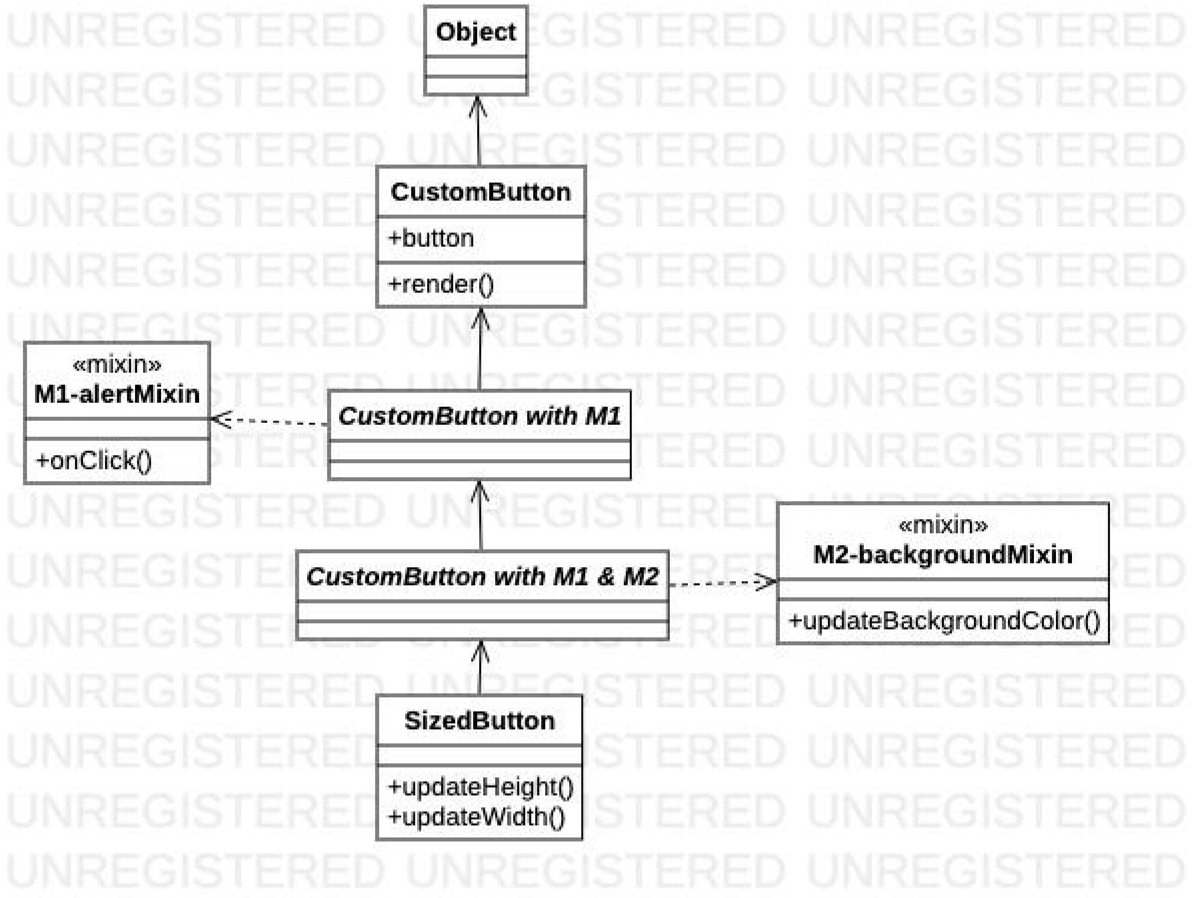
Attach functionality to a subclass from a superclass without specialisation and without the need for a particular relationship between them. Mixins provide an alternate way to compose behaviour in a class, as opposed to inheritance.

2. Applicability

Use Mixins

- to inject functionality and specialise the behaviour of a number of parent classes.
- to provide a mechanism for multiple inheritance in languages that don't support it.

3. Structure



4. Participants

- `Object`:
  - Base data type in JS from which all other types are derived.
- `CustomButton` (Concrete base class)
  - A superclass depicting a button type to be used for creating specific buttons.
- `M1,M2` (`alertMixin`, `backgroundMixin`)
  - Mixins that contain methods for use by other classes without being inherited from.
- `SizedButton` (Concrete subclass)
  - A derived class with properties composed from both mixins.

5. Collaborations

- Classes such as `CustomButton` and `SizedButton` can be instantiated directly. In the case of a class with mixins, the superclass won't receive the additional functionality but the child classes will.
- For example, if a class `C` inherits from a class `B` which inherits from a class `A` where `B` uses a mixin, then only `B` and `A` will have access to the mixin's functionality.
- Mixins should not be instantiated on their own because they may require access to an object's context (i.e., the `this` keyword).

6. Sample Code

```javascript
class CustomButton {
    constructor(value) {
        this.button = document.createElement('button');
        this.button.textContent = value;
        this.button.addEventListener('click', () => this.onClick());
    }

    render() {
        alert('Rendering...');
    }
}

class SizedButton extends CustomButton {
    constructor(value) {
        super(value);
    }

   updateHeight() {
        const height = Math.floor(Math.random() * 10);
        this.button.style.paddingTop = `${height}em`;
    }

    updateWidth() {
        const width = Math.floor(Math.random() * 10);
        this.button.style.paddingLeft = `${width}em`;
    }

    render() {
        document.querySelector('#sized-button').append(this.button);
    }
}

const alertMixin = {
    onClick() {
        alert(`${this.button.textContent} got clicked on.`);
    }
};

const backgroundMixin = {
    updateBackgroundColor() {
        this.button.style.backgroundColor = this.getRandomColor();
    },
    updateFont() {
        const fonts = [
            'sans-serif',
            'cursive',
            'Georgia',
            'Gill Sans',
            'serif'
        ];
        this.button.style.fontFamily =
            fonts[Math.floor(Math.random() * fonts.length)];
        },
    };

    Object.assign(SizedButton.prototype, alertMixin);
    Object.assign(SizedButton.prototype, backgroundMixin);
```

References

1. http://www.bracha.org/oopsla90.pdf
2. https://blog.bitsrc.io/understanding-mixins-in-javascript-de5d3e02b466
3. https://www.digitalocean.com/community/tutorials/js-using-js-mixins
4. https://justinfagnani.com/2015/12/21/real-mixins-with-javascript-classes/#bettermixinsthroughclassexpressions
5. https://javascript.info/mixins
6. https://addyosmani.com/resources/essentialjsdesignpatterns/book/#mixinpatternjavascript