13

Reutilização de código com Composição e Mixin

Esse exercício é de apenas reflexão. Você pode executar seu código se assim desejar.

Temos as seguintes classes:

```
<!-- troca-troca.html -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <p class="info">Era uma vez...</p>
    <script>

        class Aviao {

            constructor(nome) {
                this._nome = nome;
            }

            voa() {
                alert(`${this._nome} está voando`);
            }

            ligaMotor() {
                console.log('liga o motor');
            }

            fechaPortas() {
                console.log('Portas sendo fechadas');
            }
        }

        class Passarinho {

            constructor(nome) {
                this._nome = nome;
            }

            voa() {
                // hum..precisamos implementar esse método também!
            }
        }

    </script>
</body>
</html>
```

Veja que o método `voa` de `Passarinho` não esta completo. Podemos até usar herança e herdar de `Aviao`, mas com certeza um passarinho não `ligaMotor` nem `fechaPortas`. Não podemos usar herança porque `Passarinho` não é um `Aviao`.

Reutilização de código através de composição

Uma maneira de solucionar esse problema é usar composição no lugar de herança. Na composição, a classe que deseja usar o método de outra possui uma instância dessa classe. Por mais que a instância tenha vários métodos, só chamamos aqueles que nos interessa:

Alterando nosso código para usar composição:

```
<!-- troca-troca.html -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <p class="info">Era uma vez...</p>
    <script>

        class Aviao {

            constructor(nome) {
                this._nome = nome;
            }

            voa() {
                alert(`${this._nome} está voando`);
            }

            ligaMotor() {
                console.log('liga o motor');
            }

            fechaPortas() {
                console.log('Portas sendo fechadas');
            }
        }

        class Passarinho {

            constructor(nome) {
                this._nome = nome;
                // guarda uma instância de avião
                this._aviao = new Aviao(nome);
            }

            voa() {
                // usa o método voa de Aviao
                this._aviao.voa();
            }
        }
```

```
    </script>
</body>
</html>
```

Nessa solução, quem usa a instância da classe `Passarinho` nem sabe que o método `voa` usa por debaixo dos panos uma instância de `Aviao` para funcionar. Veja que a composição tem a vantagem de podermos escolher quais métodos queremos reaproveitar, diferente da herança que é tudo ou nada. Contudo, veja que com composição precisamos escrever um pouco mais, pois temos que delegar as chamadas dos métodos `voa` de `Passarinho` para o `voa` de avião.

Ainda há outra forma de resolver este problema sem usar herança nem composição, mas usando **mixin**!

Reutilização de código através de mixin!

Com mixin podemos "pegar emprestado" o método de outra classe sem termos que ter uma instância dessa classe como é o caso de composição.

Vamos alterar a classe `Passarinho` removendo a instância de `Aviao` :

```
<!-- troca-troca.html -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <p class="info">Era uma vez...</p>
    <script>

        class Aviao {

            constructor(nome) {
                this._nome = nome;
            }

            voa() {
                alert(`${this._nome} está voando`);
            }

            ligaMotor() {
                console.log('liga o motor');
            }

            fechaPortas() {
                console.log('Portas sendo fechadas');
            }
        }

        class Passarinho {

            constructor(nome) {
                this._nome = nome;
            }
```

```
            voa() {
                // executa o método `voa` de `Avião` usando como contexto a instância de `Passa
                Reflect.apply(Aviao.prototype.voa, this, []);
            }
        }

    </script>
</body>
</html>
```

Olha ai o `Reflect.apply` novamente! Nesta linha de código estamos querendo executar o método `voa` da classe `Aviao`, mas usando como contexto o `this` da instância de `Passarinho`. O último parâmetro é um array que contém os parâmetros do método. Como `voa` não recebe parâmetro algum, passamos um array vazio.

Um detalhe: foi necessário fazer Aviao.**prototype**.voa porque métodos criados usando ES6 são adicionados no `prototype`. Qualquer método adicionado em `prototype` estará disponível para todas as instâncias.