**Тема:** Структурные паттерны: «Мост», «Фасад», «Информационный эксперт», «Приспособленец».

**«Мост»**

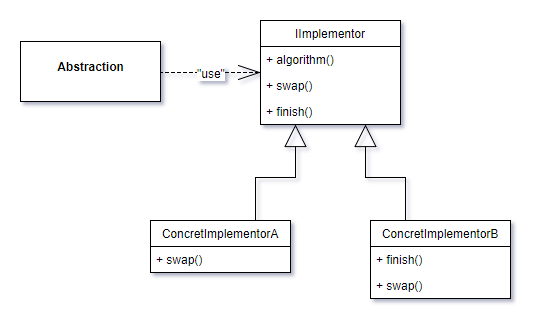
Описание:

Разделяет абстракцию и реализацию так, чтобы они могли изменяться независимо.

Пример реализации:

Создадим абстрактный класс IImplementor, в который объединим общую логику пузырьковой сортировки для конкретных реализаций ConcretImplementorA и ConcretImplementorB. В ConcretImplementorA реализуем визуализацию сортировки, а в ConcretImplementorB реализуем вывод сортировки в консоль.

Диаграмма классов:



Исходный код:

Abstraction.js

**class** Abstraction {  
 constructor() {  
 **this**.implementor = **null**;  
 }  
 getImplementor() { **return this**.implementor; }  
 setImplementor(val) {  
 **this**.implementor = val;  
 }  
 doSort() {  
 **this**.implementor.algorithm();  
 }  
 doShuffle() {  
 **this**.implementor.algorithm('shuffle');  
 }  
 doReverse() {  
 **this**.implementor.algorithm('reverse');  
 }  
}

IImplementor.js

**class** IImplementor {  
 constructor(scene) {  
 **this**.scene = scene;  
 **this**.eventSorted = **new** Event('sorted', { bubbles: **true**, cancelable: **true** });  
 **this**.algorithm = **this**.algorithm.bind(**this**);  
 }  
  
 algorithm(direction = '') {  
 **const** timestamp = **new** Date();  
 **let** colorBlocks = **this**.scene.childNodes,  
 i = 0, j = 0;  
 **const** iteration = () => {  
 **let** color1 = parseInt(colorBlocks[j].dataset.hsv, 10),  
 color2 = parseInt(colorBlocks[j + 1].dataset.hsv, 10),  
 condition;  
 **switch** (direction) {  
 **case** 'reverse':  
 condition = color1 < color2;  
 **break**;  
 **case** 'shuffle':  
 **case** 'random':  
 **case** 'rand':  
 condition = (Math.random() - 0.5) > 0;  
 **break**;  
 **default**:  
 condition = color1 > color2;  
 }  
 **if** (condition) {  
 **this**.swap(colorBlocks, j);  
 }  
 **if** (j < colorBlocks.length - 2 - i) {  
 j++;  
 timerID = setTimeout(iteration, 1, direction);  
 } **else** {  
 **if** (i < colorBlocks.length - 2) {  
 i++;  
 j = 0;  
 timerID = setTimeout(iteration, 1, direction);  
 } **else** {  
 clearTimeout(timerID);  
 **this**.finish(colorBlocks);  
 console.log('set event "sorted" with ' + (((**new** Date() - timestamp) / 1000) / 60 \* 60) + ' seconds');  
 **this**.scene.dispatchEvent(**this**.eventSorted);  
 }  
 }  
 };  
 **let** timerID = setTimeout(iteration, 1, direction);  
 }  
  
 swap(array, firstIndex) {  
 }  
  
 finish(array) {  
 }  
}

ConcretImplementorA.js

**class** ConcreteImplementorA **extends** IImplementor {  
 swap(array, firstIndex) {  
 **this**.scene.insertBefore(array[firstIndex + 1], array[firstIndex]);  
 }  
}

ConcretImplementorB.js

**class** ConcreteImplementorB **extends** IImplementor {  
 ColorBlocksToNumericArray(ColorBlocks) {  
 **let** numericArray = [];  
 ColorBlocks.forEach((curr) => {  
 numericArray.push(parseInt(curr.dataset.hsv, 10))  
 });  
 **return** numericArray;  
 }  
 swap(array, firstIndex) {  
 **let** numericArray = **this**.ColorBlocksToNumericArray(array);  
  
 **let** tmp = numericArray[firstIndex];  
 numericArray[firstIndex] = numericArray[firstIndex + 1];  
 numericArray[firstIndex + 1] = tmp;  
 console.log(numericArray);  
 }  
 finish(array) {  
 console.clear();  
 console.log(**this**.ColorBlocksToNumericArray(array));  
 }  
}

BridgeClient.js

**class** BridgeClient **extends** React.Component {  
 constructor(props) {  
 **super**(props);  
 **this**.abstr = **new** Abstraction();  
 **this**.withoutConsole = **this**.withoutConsole.bind(**this**);  
 **this**.withConsole = **this**.withConsole.bind(**this**);  
 }  
  
 withoutConsole() {  
 **this**.abstr.setImplementor(**new** ConcreteImplementorA(**this**.scene));  
 **this**.abstr.doSort(); // "ConcreteImplementorA.operation"  
 }  
 withConsole() {  
 **this**.abstr.setImplementor(**new** ConcreteImplementorB(**this**.scene));  
 **this**.abstr.doReverse(); // "ConcreteImplementorB.operation"  
 }  
 render() {...}  
}

Снимки экрана:

