# Solutions-Lab: Object Composition

## 1.Order Rectangles

function orderRectangles(rectanglesData) {

let rectangles = [];

for (let [width, height] of rectanglesData) {

let rectangle = makeRectangle(width, height);

rectangles.push(rectangle);

}

rectangles.sort((a, b) => a.compareTo(b));

return rectangles;

function makeRectangle(width, height){

let rectangle = {

width: width,

height,

area: () => rectangle.width \* rectangle.height,

compareTo: function(otherRectangle){

let areaComparation = otherRectangle.area() - rectangle.area();

let widthComparation = otherRectangle.width - rectangle.width;

return areaComparation || widthComparation;

}

};

return rectangle;

}

}

console.log(orderRectangles([[10,5],[5,12]]))

console.log(orderRectangles([[10,5], [3,20], [5,12]]))

## 2.Fibonacci

function getFibonator(){

let sum = 0;

let first = 0;

let second = 1;

return makeFibonator;

function makeFibonator(){

sum = first + second;

first = second;

second = sum;

return first;

}

}

let fibonacciNumber = getFibonator();

console.log(fibonacciNumber()); // 1

console.log(fibonacciNumber()); // 1

console.log(fibonacciNumber()); // 2

console.log(fibonacciNumber()); // 3

console.log(fibonacciNumber()); // 5

console.log(fibonacciNumber()); // 8

console.log(fibonacciNumber()); // 13

## 3.List Processor

let processedList = //in Judge must be paste only the IIFE

(function processLict(){

let processedArray = [];

let command = {

add: function(processedArray, stringForAdding){

processedArray.push(stringForAdding);

return processedArray;

},

remove: function(processedArray, stringForRemoving){

processedArray = processedArray.filter(s => s !== stringForRemoving);

return processedArray;

},

print: function(){

console.log(processedArray.join(','));

return processedArray;

}

};

return function readInputArrayAndApplyCommand(inputArray){

for(let stringCommand of inputArray){

let [commandName, commandSubject] = stringCommand.split(' ');

processedArray = command[commandName](processedArray, commandSubject);

}

}

})()

processedList(['add hello', 'add again', 'remove hello', 'add again', 'print'])

//processedList(['add pesho', 'add gosho', 'add pesho', 'remove pesho','print'])

//processedList(['add JSFundamentals', 'print', 'add JSAdvanced', 'print','add JSApplications','print'])

## 4.Cars

function createOrModifyObject(inputCommands) {

let objectProcessor = (function processObject() {

let processedObjects = {};

function create(name, inherit, parentName) {

if (inherit === 'inherit') {

processedObjects[name] = Object.create(processedObjects[parentName]);

} else {

processedObjects[name] = {};

}

}

function set(name, key, value) {

processedObjects[name][key] = value;

}

function print(name) {

let printedObject = processedObjects[name];

let printedObjectKeys = Object.keys(printedObject);

let printedObjectProperties = [];

for (let printedObjectKey of printedObjectKeys) {

printedObjectProperties.push(printedObjectKey + ':' + printedObject[printedObjectKey]);

}

let printedObjectPrototype = Object.getPrototypeOf(printedObject);

while(printedObjectPrototype !== null && printedObjectPrototype !== undefined){

//while (printedObjectPrototype !== null) {

for (let prototypeKey of Object.keys(printedObjectPrototype)) {

printedObjectProperties.push(prototypeKey + ':' + printedObject[prototypeKey]);

}

printedObjectPrototype = Object.getPrototypeOf(printedObjectPrototype);

}

console.log(printedObjectProperties.join(', '));

}

return { create, set, print };

})();

for (let inputCommand of inputCommands) {

let [command, ...parameters] = inputCommand.split(' ');

objectProcessor[command](...parameters);

}

}

createOrModifyObject(['create c1', 'create c2 inherit c1', 'set c1 color red', 'set c2 model new', 'print c1', 'print c2'])

function createOrModifyObject(inputCommands) {

let objectProcessor = (function processObject() {

let processedObjects = {};

function create(name, inherit, parentName) {

if (inherit === 'inherit') {

processedObjects[name] = Object.create(processedObjects[parentName]);

} else {

processedObjects[name] = {};

}

}

function set(name, key, value) {

processedObjects[name][key] = value;

}

function print(name) {

let printedObject = processedObjects[name];

//let printedObjectKeys = Object.keys(printedObject);

let printedObjectProperties = [];

for (let printedObjectKey in printedObject) {

printedObjectProperties.push(printedObjectKey + ':' + printedObject[printedObjectKey]);

}

// for (let printedObjectKey of printedObjectKeys) {

// printedObjectProperties.push(printedObjectKey + ':' + printedObject[printedObjectKey]);

// }

// let printedObjectPrototype = Object.getPrototypeOf(printedObject);

// while(printedObjectPrototype !== null && printedObjectPrototype !== undefined){

// //while (printedObjectPrototype !== null) {

// for (let prototypeKey of Object.keys(printedObjectPrototype)) {

// printedObjectProperties.push(prototypeKey + ':' + printedObject[prototypeKey]);

// }

// printedObjectPrototype = Object.getPrototypeOf(printedObjectPrototype);

// }

console.log(printedObjectProperties.join(', '));

}

return { create, set, print };

})();

for (let inputCommand of inputCommands) {

let [command, ...parameters] = inputCommand.split(' ');

objectProcessor[command](...parameters);

}

}

createOrModifyObject(['create c1', 'create c2 inherit c1', 'set c1 color red', 'set c2 model new', 'print c1', 'print c2'])

|  |
| --- |
| function cars(commands) { |
|  | let map = new Map(); |
|  | let carManager = { |
|  | create: function ([name, , parent]){ |
|  | parent = parent ? map.get(parent) : null; |
|  | let newObj = Object.create(parent); |
|  | map.set(name, newObj); |
|  | return newObj; |
|  | }, |
|  | set: function ([name, key, value]) { |
|  | let obj = map.get(name); |
|  | obj[key] = value; |
|  | }, |
|  | print: function ([name]) { |
|  | let obj = map.get(name); |
|  | let allProperties = [] |
|  | Object.keys(obj).forEach(key => allProperties.push(`${key}:${obj[key]}`)); |
|  | while(Object.getPrototypeOf(obj)) { |
|  | Object.keys(Object.getPrototypeOf(obj)).forEach(key => allProperties.push(`${key}:${Object.getPrototypeOf(obj)[key]}`)); |
|  | obj = Object.getPrototypeOf(obj); |
|  | } |
|  | console.log(allProperties.join(', ')) |
|  | } |
|  | }; |
|  |  |
|  | for(let cmd of commands){ |
|  | let tokens = cmd.split(' '); |
|  | let action = tokens.shift(); |
|  | carManager[action](tokens); |
|  | } |
|  | } |

|  |
| --- |
| function solve(params) { |
|  | let processor = (function () { |
|  | let objects = {}; |
|  |  |
|  | function create(name, parent) { |
|  | if (parent === undefined) { |
|  | objects[name] = {}; |
|  | } else { |
|  | objects[name] = Object.create(objects[parent]); |
|  | } |
|  | } |
|  |  |
|  | function set(name, key, value) { |
|  | objects[name][key] = value; |
|  | } |
|  |  |
|  | function print(name) { |
|  | let obj = objects[name]; |
|  | let keys = Object.keys(obj); |
|  | let result = []; |
|  | for (const key of keys) { |
|  | result.push(`${key}:${obj[key]}`); |
|  | } |
|  | let proto = Object.getPrototypeOf(obj); |
|  | while (proto !== null && proto !== undefined) { |
|  | for (const k of Object.keys(proto)) { |
|  | result.push(`${k}:${obj[k]}`); |
|  | } |
|  | proto = Object.getPrototypeOf(proto); |
|  | } |
|  |  |
|  | console.log(result.join(", ")); |
|  | } |
|  |  |
|  | return { |
|  | create, |
|  | set, |
|  | print |
|  | }; |
|  | })(); |
|  |  |
|  | params.forEach(command => { |
|  | let tokens = command.split(/\s+/g).filter(i => i !== "" && i !== "inherit"); |
|  | processor[tokens.shift()](...tokens); |
|  | }); |
|  | } |
|  |  |
|  | solve(['create c1', |
|  | 'create c2 inherit c1', |
|  | 'set c1 color red', |
|  | 'set c2 model new', |
|  | 'print c1', |
|  | 'print c2' |
|  | ]) |

## 5.Sum

function addOrSubtract(){

let elementNum1, elementNum2, elementResult;

function init(selectorNum1, selectorNum2, selectorResult){

elementNum1 = document.querySelector(selectorNum1);

elementNum2 = document.querySelector(selectorNum2);

elementResult = document.querySelector(selectorResult);

}

function add(){

elementResult.value = Number(elementNum1.value) + Number(elementNum2.value);

}

function subtract(){

elementResult.value = Number(elementNum1.value) - Number(elementNum2.value);

}

let returnedFunctions = {init, add, subtract};

return returnedFunctions;

}

|  |
| --- |
| function result() { |
|  | let num1, num2, result; |
|  | function init(num1Sel, num2Sel, resultSel) { |
|  | num1 = $(num1Sel); |
|  | num2 = $(num2Sel); |
|  | result = $(resultSel); |
|  | } |
|  |  |
|  | function add() { |
|  | action((a, b) => a + b); |
|  | } |
|  |  |
|  | function subtract() { |
|  | action((a, b) => a - b); |
|  | } |
|  |  |
|  | function action(operation) { |
|  | let val1 = Number(num1.val()); |
|  | let val2 = Number(num2.val()); |
|  | result.val(operation(val1, val2)); |
|  | } |
|  |  |
|  | let model = {init, add, subtract}; |
|  | return model; |
|  | }; |

|  |
| --- |
| function solve(params) { |
|  | return (function () { |
|  | let sel1; |
|  | let sel2; |
|  | let resSel; |
|  | return { |
|  | init: function (selector1, selector2, resultSelector) { |
|  | sel1 = $(`${selector1}`); |
|  | sel2 = $(`${selector2}`); |
|  | resSel = $(`${resultSelector}`); |
|  | }, |
|  | add: function () { |
|  | resSel.val(Number(sel1.val()) + Number(sel2.val())); |
|  | }, |
|  | subtract: function () { |
|  | resSel.val(Number(sel1.val()) - Number(sel2.val())); |
|  | } |
|  | } |
|  | })(); |
|  | } |