Java Script Object Oriented Programming

Table of Contents

[OOP 2](#_Toc25871484)

[Objects 2](#_Toc25871485)

[Factory function 2](#_Toc25871486)

[Constructor function 2](#_Toc25871487)

[dynamic objects 2](#_Toc25871488)

[Clone object 3](#_Toc25871489)

[Getter and Setter 3](#_Toc25871490)

[Prototypes 3](#_Toc25871491)

[Base object 3](#_Toc25871492)

[Property descriptor 3](#_Toc25871493)

[Prototype inheritance 3](#_Toc25871494)

[ES6 Classes 3](#_Toc25871495)

[Class 3](#_Toc25871496)

[Private with Symbol() 4](#_Toc25871497)

[Private with WeakMaps() 4](#_Toc25871498)

[Get and Set 4](#_Toc25871499)

[Inheritance 5](#_Toc25871500)

[Stack with JavaScript 5](#_Toc25871501)

# OOP

Encapsulation

Abstraction

Inheritance

Polymorphism

# Objects

const circle = {

radius: 1,

location: {

x: 1,

y: 1

},

isVisible: true,

draw: function() {

console.log(“draw”);

}

};

circle.radius = 2;

circle[“radius”] = 2;

circle.draw();

## Factory function

function createCircle(radius) {

return {

radius,

draw() {

console.log(“draw”);

}

};

}

const circle = createCircle(5);

## Constructor function

function Circle(radius) {

*let color = “red”; // private member*

this.radius = radius; *// public member*

this.draw = function() { *// public method*

console.log(“draw”);

}

*let paint = function() { // private method*

*}*

}

const circle = new Circle(5);

## dynamic objects

const circle = {

radius: 1

};

circle.color = “yellow”;

circle.draw = function() { }

delete circle.color;

delete circle.draw;

## Clone object

const circle2 = Object.assign( {}, circle); // add object to existing object

const circle2 = { …circle }; // clone object

## Getter and Setter

const person = {

name: “Ronnnie”,

age: 30,

get Age() {

return age;

}

set Age(value) {

age = value;

}

};

person.Age = 30;

console.log(person.Age);

# Prototypes

## Base object

// Every object (except the root object) has a prototype (parent).

// To get the prototype of an object:

Object.getPrototypeOf(obj);

## Property descriptor

// To get the attributes of a property:

Object.getOwnPropertyDescriptor(obj, 'propertyName');

// To set the attributes for a property:

Object.defineProperty(obj, 'propertyName', {

configurable: false, // cannot be deleted

writable: false,

enumerable: false

});

## Prototype inheritance

Circle.prototype.draw = function() {}

# ES6 Classes

## Class

class Circle {

constructor(radius) {

this.radius = radius;

}

// These methods will be added to the prototype.

draw() {

}

// This will be available on the Circle class (Circle.parse())

static parse(str) {

}

}

## Private with Symbol()

const \_size = Symbol();

const \_draw = Symbol();

class Square {

constructor(size) {

// "Kind of" private property

this[\_size] = size;

}

// "Kind of" private method

[\_draw]() {

}

}

## Private with WeakMaps()

const \_width = new WeakMap();

class Rectangle {

constructor(width) {

\_width.set(this, width);

}

draw() {

console.log('Rectangle with width' + \_width.get(this));

}

}

## Get and Set

class Circle {

constructor(radius) {

this.radius = radius;

}

*get radius() {*

*return this.radius;*

*}*

*set radius(value) {*

*this.radius = value;*

*}*

}

*const \_radius = new WeakMap();*

class Circle {

constructor(radius) {

*\_radius.set(this, radius);*

}

*get radius() {*

*return \_radius.get(this);*

*}*

*set radius(value) {*

*\_radius.set(this, value);*

*}*

}

## Inheritance

*class Shape {*

*constructor(color) {*

*this.color = color;*

*}*

*move() {*

*console.log(“move”);*

*}*

*}*

class Triangle extends Shape {

constructor(color) {

super(color);

}

draw() {

super.draw();

}

}

# ES6 Modules

## Module

const \_radius = new WeakMap();

export class Circle {

constructor(radius) {

\_radius.set(this, radius);

}

draw() {

console.log('Circle with radius ' + \_radius.get(this));

}

}

## Use

import {Circle} from './circle.js';

const c = new Circle(10);

c.draw();

# Stack with JavaScript

const \_items = new WeakMap();

class Stack {

constructor() {

\_items.set(this, []);

}

push(obj) {

\_items.get(this).push(obj);

}

pop() {

const items = \_items.get(this);

if (items.length === 0)

throw new Error('Stack is empty.');

return items.pop();

}

peek() {

const items = \_items.get(this);

if (items.length === 0)

throw new Error('Stack is empty.');

return items[items.length - 1];

}

get count() {

return \_items.get(this).length;

}

}