Это набор соглашений по коду позволяющее упростить, сократить и структурировать код. Есть несколько принципов которым нужно следовать:

1. Наследование – способность объекта унаследовать все указанные свойства и методы у родительского объекта.
2. Инкапсуляция – требование, в рамках которого обособленный функционал полностью собран и замкнут в конкретном отдельном объекте.
3. Полиморфизм – необходимое следствие наследования заключающийся в возможности потомка переопределить методы родители.

Все эти принципы реализуются в специализированной структуре данных под названием классы. В js классов нет (Они есть (Классы Шредингера)). Наследование конструкторов. Конструкторы (то есть функции) могут реализовать принцип наследования друг от друга путём переопределения контекста потомка контекстам родителя.

function F( ) {

this.value = 123;

}

function G( ){

// наследование осуществляется с помощью специальных методов конструктора (у функций есть функции)

F.call(this, p)

}

const g = new G;

g.value -> 123

Наследоваться можно от одного родителя, но у родителя может быть много потомков. В случае наследования от нескольких родителей, родители наследуют от других родителей.

Правила наследования:

1. Наследования в проектах должно быть как можно меньше.

**Прототипы в js**

Любой объект в js имеет свойство прототип. Свойство прототип это специализированный объект, который можно доопределять, переопределять или наследовать. Когда пользователь обращается к какому либо свойству или методу объекта, то сначала поиск осуществляется сначала в свойствах и методах объекта, потом в прототипе этого объекта, если ничего не нашлось, то поиск может быть совершён в прототипе родителя (если он есть).

const Data = {

key1:1,

key2:true,

};

const Obj = {}

Obj.prototype = Data;

В новом стандарте добавлена конструкция класса, которая является синтаксическим сахаром над ппрототипами.

Объявление класса:

class Man {

eat( ){

return true;

}

live(){

return this.eat();

}

}

const man = new Man;

man.eat( );

Свойства класса:

class Man {

name = “Вася”;

money = 100;

hp = 100;

eat(){

this.hp += 10;

this.money -=1;

}

Поскольку классы это обёртка над прототипом, а прототип это объект, то все свойства и методы класса по умолчанию публичные, то есть доступны к вызову в экземпляре класса. В новом стандарте сделали приватные методы. У классов есть несколько стандартных методов, а именно это конструктор и деструктор. Метод конструктор автоматически вызывается при создании экземпляра класса. Деструкторов в js нет, но он автоматически вызывается при уничтожении экземпляра класса.

class Man{

construcot(name, money = 0){

this.name = name;

this.money = money;

this.hp = 100;

}

eat(){…}

}

const man1 = new Man(“Вася”, 100);

const man2 = new Man(“Петя”, 1000);

Наследование происходит с помощью ключевого слова extend.

class Woman extends Man{…}

Если класс потомок не имеет своего конструктора, то это наследование можно таким и оставить. Если класс потомок имеет свой конструктор, то внутри этого конструктора в обязательном порядке нужно вызвать конструктор родителя, иначе скрипт выкинет ошибку и не будет.

class Woman extends Man {

constructor(name, money){

super(name, money);

this.beauty = 100000;

}

При переопределении метода родителя в потомке, в потомке можно вызывать как и неизменный метод родителя, так и изменяемый метод в потомке.

Объявление класса осуществляется в новом файле в котором присутствует только класс. Название класса совпадает с названием файла, пишется в большой буквы.

**Написание универсального калькулятора**

class RealCalculator{

add(a,b){return a+b;}

sub(a,b){return a-b;}

mult(a,b){return a\*b;}

div(a,b){return a/b;}

pow(a, n){return Math.pow(a,n);}

prod(a, p){return a \* p;}

one(){

return 1;

}

zero(){

return 0;

}

}

class Complex{

constructor(re=0, im = 0;){

this.re = re;

this.im = im;

}

}

class ComplexCalculator extends RealCalculator{

add(a,b){

return new Complex(a.re + b.re, a.im + b.im);

}

sub(a,b){

return new Complex(a.re - b.re, a.im - b.im);

}

mult(){…}

div(){…}

prod(a,p){

return new Complex(a.re \* p, a.im \* p);

}

one(){

return new Complex(super.one());

}

zero(){

return Complex();

}

В папке js сделать 2 папки калькуляторов и типов.

Д/з: реализовать обычный и комплексный калькулятор

\*комплексный и обычный объеденить в один