**Creational**:

You have three instances: Person, Employee and Shopper.

class Person {  
  
 constructor(name='unnamed person') {  
 this.name = name;  
 }  
}

class Shopper extends Person {  
  
 constructor(name, money=0) {  
 super(name);  
 this.money = money;  
 this.employed = false;  
 }  
}

class Employee extends Shopper {  
  
 constructor(name, money=0, employer='') {  
 super(name, money);  
 this.employerName = employer;  
 this.employed = true;  
 }  
}

You need to create Shopper **Alex Banks** and Employee **Eve Porcello**, each has 100 moneys (🤷‍♂️),

But you can’t just create them with simple initialization.

const eve = new Employee('Eve Porcello', ...);

You need to delegate initialization logic to another instance, which you need to design and implement, based on your Creational patterns’ knowledge. Also, Employee should be initialized only once – if you try to initialize new instance of it, previously created object should be returned (also, make sense to warn about it in the console).

**Structural**

1. Given code below

class OldCalculator {  
 constructor() {  
 this.operations = function(term1, term2, operation) {  
 switch (operation) {  
 case 'add':  
 return { res: term1 + term2 };  
 case 'sub':  
 return { res: term1 - term2 };  
 default:  
 return ***NaN***;  
 }  
 };  
 }  
}  
  
class NewCalculator {  
 constructor() {  
 this.multiply = function(term1, term2) {  
 return term1 \* term2;  
 };  
 this.divide = function(term1, term2) {  
 return term1 / term2;  
 };  
 }  
}

you need to create a new class, which would let interact those two calculators:

X.multiply(oldCalculator.operations(3,2, "sub"), 6);

1. Create an **UltimateCalculator**, that would let you use all the operations at once with one interface, which should reuse **OldCalculator** and **NewCalculator**.
2. Create a **CleverCalculator**, that would let you cache results of the **UltimateCalculator** calculation depending on arguments and type of operation.
3. Wrap **CleverCalculator** so on any function call it would log its arguments and type of operation.

**Behavior**

1. Create a **CumulativeSum** class that would let you get the sum of the elements as follow:

const sum1 = new CumulativeSum();  
***console***.log(sum1.add(10).add(2).add(50).sum);

1. Given class **SpecialMath** create a new instance **Command**, which would store all the commands given to the **SpecialMath**

class SpecialMath {  
 constructor(num) {  
 this.\_num = num;  
 }  
  
 square() {  
 return this.\_num \*\* 2;  
 }  
  
 cube() {  
 return this.\_num \*\* 3;  
 }  
  
 squareRoot() {  
 return ***Math***.sqrt(this.\_num);  
 }  
}  
  
...  
  
const x = new Command(new SpecialMath(5));  
x.execute('square');  
x.execute('cube');  
  
***console***.log(x.commandsExecuted); // ['square', 'cube']

1. You have a list of users. Add functionality to subscribe to the changes on it (e.g. after calling **push**, **pop**).

const users = ["Alex Banks", "Eve Porcello"];