JS Patterns

DEADLINE: 27/06/2022

FOLDER STRUCTURE

```
FL19_HW8/ *                                          * - required
    ├─ task
    ├─ homework/ *
        ├─ js/ *
            ├─ index.js *
        ├─ index.html *
        ├─ .eslintrc.js *
```

TASK 1

1.1) Imagine there is toy factory which can produce teddy toys, wooden toys and plastic toys. This factory should be implemented by **Factory Method** js pattern. Factory has "produce" method which can create teddy, wooden or plastic toy according to entry name *(f.e. "teddy").*

Toy:

- Create abstract class Toy which has **"name"** and **"price"** properties and method **"getToyInfo()"** which returns string *"The toy name is [toy name]. It costs [toy price] dollars."*.
- Create tree another classes for teddy, wooden and plastic toys. Those classes should be extended from abstract class Toy. Each class should has string property "**material**" with value of material (f.e. material = "cotton"/"wood"/"plastic"). Also each class has **"getMaterialInfo()"** method which returns string *"The toy [toy name] was made of [toy material]."*

Factory:

- Should has **"produce"** method which creates new instance of toy;
- Receives type and according to this produces teddy, wooden or plastic toy.
- Uses classes of toys inside
- Make **"type"** parameter optional and when there is no type factory creates plastic toy (see example).

Examples:

```
const teddyBear = factory.produce('Bear', 200, 'teddy')
console.log(teddyBear.getToyInfo());  // The toy name is "Bear". It costs 200 dollars.
console.log(teddyBear.getMaterialInfo());  // The toy "Bear" was made of cotton.

const plasticCar = factory.produce('Car', 100);
console.log(plasticCar.getToyInfo());  // The toy name is "Car". It costs 100 dollars.
console.log(plasticCar.getMaterialInfo());  // The toy "Car" was made of plastic.
```

1.2) In this task it is necessary to extend our factory and allow it to create only unique toys (by name). For it extend factory using **Flyweight** js pattern
Example:

```
const teddyBear = factory.produce('Bear', 200, 'teddy')
console.log(teddyBear.getToyInfo());  // The toy name is "Bear". It costs 200 dollars.
console.log(teddyBear.getMaterialInfo());  // The toy "Bear" was made of cotton.

const plasticBear = factory.produce('Bear', 150, 'plastic')
console.log(plasticBear.getToyInfo())  // The toy name is "Bear". It costs 200 dollars.
console.log(plasticBear.getMaterialInfo());  // The toy "Bear" was made of cotton.
```

1.3) But customer of our factory needs only one wooden toy, so we need to create only one instance of wooden toy. Use **Singleton** js pattern to achieve this.
Example:

```
const woodenHorse = factory.produce('Horse', 400, 'wooden');
console.log(woodenHorse.getToyInfo());  // The toy name is "Horse". It costs 400 dollars.

const woodenBear = factory.produce('Bear', 200, 'wooden');
console.log(woodenHorse.getToyInfo());  // The toy name is "Horse". It costs 400 dollars.
```

TASK 2

Imagine a doctor who has “Mercedes” car and who drives to hospital every day. But he lives in city and has to drive through traffic jams. In that way he is late on hospital and patients suffer. So we need to help him and extend his “Mercedes” by ambulance siren. To help doctor we need:

- Create Car class which has properties of car name and car host. Also add “carSound()” method which returns string “Usual car sound.”
- Create **Decorator** pattern which will extend our car class and add it “ambulanceSound” method which return string “Siren sound.”.
- Decorator should be appliable to another cars

Example:

```
const mercedes = new Car('Mercedes', 'Doctor');
const ambulanceMercedes = AmbulanceCar(mercedes);
console.log(ambulanceMercedes.ambulanceSound()); // // Siren sound.

const toyota = new Car('Toyota', 'Doctor2');
const ambulanceToyota = AmbulanceCar(toyota);
console.log(ambulanceToyota.ambulanceSound()); // Siren sound.
```

RESTRICTIONS

- Adding **task/** folder is forbidden. Do not push it to repository. (Only **homework/** folder should be pushed)
- Do not use any external libraries
- Write both tasks in index.js file
- Do not change index.html and eslintrc.js

BEFORE SUBMIT

- Code should be clean, without comments, readable, and tested
- Make sure your GitLab folder structure meets folder structure from this document (without **task** folder)
- Use linter :
    - In order to use npm package manager you should install nodejs (https://nodejs.org/)
    - Install eslint to check your code (npm install -g eslint)
    - open a terminal (or cmd)
    - run eslint (i.e. eslint ./js/index.js)

Code should be without 'errors'

USEFUL LINKS

https://refactoring.guru/uk/design-patterns/catalog/
https://www.patterns.dev/posts/classic-design-patterns/

SUBMIT

- The **FL19_HW8** folder without **task** folder should be uploaded to GitLab repository "**FL-19**" into **main** branch.