Exercise 1

You are given a constructor function that can be used to instantiate objects.

Your task is to rewrite this function to a class.

function Student(name, age) {

this.name = name;

this.age = age;

this.technologies = [];

this.status = 'Junior';

this.setTechnologies = function(technologies) {

this.technologies = [

...this.technologies,

...technologies,

];

}

this.setNewStatus = function(newStatus) {

this.status = newStatus;

}

}

const student = new Student('Maxim', 20);

student.setTechnologies(['HTML', 'CSS', 'JavaScript']);

student.setNewStatus('Middle');

console log(student);

Task 2

You need to create a Person class whose constructor will take 2 parameters:

name. The name of the person.

age. Number of completed years.

You also need to implement the compareAge method in the Person class. compareAge takes an instance of the Person class and compares full years.

This code should return the result in the following format. If one Person has more or equal number of years than another, then display a message via alert “name1 is older than name2”. Otherwise, "name1 is younger than name2".

Run this code to test how your Person class works

const person1 = new Person('Maxim', 24);

const person2 = new Person('Svetlana', 36);

const person3 = new Person('Irina', 23);

person1.compareAge(person2); // Maxim is younger than Svetlana

person2.compareAge(person3); // Svetlana is older than Irina

person3.compareAge(person1); // Irina is younger than Maxim

Task 3

Your friend Artem is the owner of a car service. Now all requests for car repairs are processed manually, which of course is not very convenient. Therefore, Artem asked you, as a web developer, to help automate this process.

You need to create a CarService class with 2 parameters passed to its constructor:

name. The name of the car service.

working hours. An object that holds the from and till keys. from means the time of the beginning of the working day for a car service, till - the time of the end of the working day. Time values are written in the format “h:mm” (for example, 9:00).

Initialize these parameters with this. Also, the workingHours parameter is optional. So create a value (object data type) as default. This object must be named DefaultWorkingHours and must be static. If the workingHours parameter was not passed, then substitute in this.workingHours the value from DefaultWorkingHours, which is equal to:

static DefaultWorkingHours = {

from: '9:00',

till: '20:00',

}

Add a method to the class called repairCar, which should take 1 parameter carName (the name of the car to be repaired, string data type).

If carName was not passed, then display the error “You need to specify the name of the car in order to repair it” in the console via console.error. This is where the function execution should end.

If the carName parameter was passed to repairCar, then you need to compare the current hours with the car service hours. If the current time (hours) is not within the range of the car service, then display the message “Unfortunately, we are closed now. Come back tomorrow". Otherwise, display the message “Now we will repair your car carName! Please wait” (carName is the parameter of the repairCar method).

Note: when comparing times, take into account only hours, do not pay attention to minutes. Let the minutes always be the string "00".

To test the health of the CarService class, use this code:

const carService = new CarService('RepairCarNow', { from: '8:00', till: '20:00' });

carService.repairCar('BMW');

Hint 1: in order to extract the string "8:00" the hour value, you can use .split(':'), get the first element from the resulting array, and convert it to a number.

Task 4

You have been assigned a task on a project. It is necessary to develop a dictionary in which you can add words with their description, delete words and get them.

First, create a Dictionary class whose constructor takes 1 name parameter. Initialize this value with this. Also, the words field must be declared in the constructor, which must be an empty object by default.

You need to add 3 methods to the Dictionary class:

1. add. This method accepts 2 parameters: word - word, string data type; description - word description, string data type. Assign to the words object by the word key (parameter of the add method) the value of the object that will store data about the word and its description. An example of such an object:

{

word: 'word',

description: 'description',

}

If the word already exists in the words object, then there is no need to overwrite a word that has already been stored previously.

2.remove. This method accepts a key from the words object, by which the word should be deleted and removes this word from the object.

3. get. This method accepts a key from the

who words. It is necessary to return the found word from the method according to the passed key parameter.

4. showAllWords. This method does not accept any parameters. Its task is to output all the words that are in the words object to the console in the format “word - description” (word is the word itself, description is the description of the word).

To test the written class, use this code:

const dictionary = new Dictionary('Explanatory Dictionary');

dictionary. add('JavaScript', 'popular programming language');

dictionary.add('Web developer', 'A person who creates new services and sites or maintains and supplements existing ones');

dictionary.remove('JavaScript');

dictionary.showAllWords();

// Web developer - A person who creates new services and sites or maintains and complements existing ones

Task 5

In the last task, you created a dictionary. The customer is satisfied with the work done and the development team too. Since you already have a good understanding of dictionary technology, you have been asked to implement another dictionary. The customer wants to create a new category of words “Compound words”.

Use the code written in the last task. You need to create a child class HardWordsDictionary from the Dictionary class. The constructor of the HardWordsDictionary also accepts the name parameter and calls the constructor of the parent class.

All methods that are in the Dictionary should remain unchanged in the HardWordsDictionary, except for one. You need to override the add method, which should work in the same logic as Dictionary's add method, only the word object should look like this:

{

word: 'word',

description: 'description',

isDifficult: true

}

To test the HardWordsDictionary class, use this code:

const hardWordsDictionary = new HardWordsDictionary('Hard words');

hardWordsDictionary.add('amateur', 'One who practices science or art without special training, having only a superficial knowledge.');

hardWordsDictionary.add('neologism', 'A new word or expression, as well as the new meaning of the old word.');

hardWordsDictionary.add('quantum', 'The indivisible part of some quantity in physics.');

hardWordsDictionary.remove('neologism');

hardWordsDictionary.showAllWords();

// amateur - Someone who is engaged in science or art without special training, possessing only superficial knowledge.

// quantum - An indivisible part of any quantity in physics.

Task 6

You are well done! You have implemented the logic for two dictionaries and they work just fine. But then a senior developer comes to you and says that you need to use encapsulation in your code, as this is an important principle of OOP.

Of course, you agree with the senior developer. He advised you to make the name and words fields private. Implement this with the pound sign “#”.

Now you need to add getters and setters for the words and name fields to the Dictionary class. For name, name the getter mainName and the setter setMainName. For words, name the getter allWords and the setter addNewWord. mainName, setMainName and allWords must be defined using the set or get keywords.

But addNewWord must be created as a regular method that takes 2 parameters: word - word, data type string; description - word description, string data type. The addNewWord method must create a new word in the words object without any checks. addNewWord is implemented without set, because set cannot take more than 1 parameter.

After you have made the fields private and created the getters and setters, you need to fix the logic for the Dictionary and HardWordsDictionary classes.

Test the final code on this example:

const hardWordsDictionary = new HardWordsDictionary('Hard words');

hardWordsDictionary.add('amateur', 'One who practices science or art without special training, having only a superficial knowledge.');

hardWordsDictionary.add('neologism', 'A new word or expression, as well as the new meaning of the old word.');

hardWordsDictionary.add('quantum', 'The indivisible part of some quantity in physics.');

hardWordsDictionary.remove('neologism');

hardWordsDictionary.showAllWords();

console.log(hardWordsDictionary.mainName); // Difficult words

hardWordsDictionary.setMainName = 'New Dictionary';

console.log(hardWordsDictionary.mainName); // New Dictionary

console.log(hardWordsDictionary.allWords); // displays an object that contains the words dilettante and quantum

Task 7

Your company is building an internal product to help you manage the growth of your developers. You have been given the task of implementing such a product.

First you need to create a Developer class, the constructor of which will take 3 parameters:

fullName. Developer name.

age. Developer age.

position. The current position of the developer in the company (for example, Junior, Middle, Senior).

Initialize all parameters with this. Also create a technologies field inside the constructor, which is by default

iu will be equal to an empty array.

In addition, in the Developer class, you need to create 2 methods:

code. A method whose body is initially empty (nothing in curly braces).

learn New Technology. This method accepts 1 technology parameter, which must be added to the end of the technologies array.

After the done steps, you should get a Developer template, thanks to which you will create child classes.

You now need to create 3 child classes from the Developer class:

junior developer. The constructor of this class takes 2 parameters: fullName and age. Call the constructor of the parent class and pass these 2 parameters there. As the 3rd we have position. You need to set it as default. Write the value "Junior" as the 3rd parameter of the call to the parent constructor. In addition, override the code method so that it prints the string “Junior developer is writing code...” to the console. The technologies array must contain the following technologies: 'HTML', 'CSS', 'JavaScript'.

Middle Developer. Do the same work in the constructor as JuniorDeveloper. Only in place of position pass the value "Middle". The code method of the MiddleDeveloper class should output the line “Middle developer is writing code...” to the console. The technologies array must contain the following technologies: 'HTML', 'CSS', 'JavaScript', 'React'.

senior developer. Do the same work in the constructor as JuniorDeveloper. Only pass the value “Senior” to the position position. The code method of the SeniorDeveloper class should output the string “Senior developer is writing code...” to the console. The technologies array must contain the following technologies: 'HTML', 'CSS', 'JavaScript', 'React', 'NodeJS'.

Note: When you override the code method in child classes, you are using the Polymorphism principle.

Test your written classes with this code:

juniorDeveloper.code(); // Junior developer writes code...

middleDeveloper.code(); // Middle developer writes code...

seniorDeveloper.code(); // Senior developer writes code...

console.log(juniorDeveloper.fullName, juniorDeveloper.age,

juniorDeveloper.position); // 'Anastasia', 20, 'Junior'

console.log(middleDeveloper.fullName, middleDeveloper.age,

middleDeveloper.position); // 'Igor', 25, 'Middle'

console.log(seniorDeveloper.fullName, seniorDeveloper.age,

seniorDeveloper.position); // 'Maxim', 30, 'Senior'