Цель урока:

Вы узнаете все 4 принципа ООП и научитесь применять их на практике. Также в уроке будут рассмотрены способы работы с классами в JavaScript.

Содержание урока:

1. Что такое ООП?
2. Как использовать классы в JavaScript (class и function)?
3. 4 основных принципа ООП: инкапсуляция, наследование, полиморфизм, абстракция
4. Статические методы и свойства

Дополнительные материалы:

1. Классы в JavaScript: https://learn.javascript.ru/classes
2. Про ООП простыми словами: https://habr.com/ru/post/345658/
3. Конструкторы, создание объектов через new: https://learn.javascript.ru/constructor-new
4. Урок 7. JavaScript. Все о ES6 Классах (+ Практическое Применение): https://youtu.be/uLY9GXGMXaA

Задание #1

Вам дана функция-конструктор, с помощью которой можно создавать экземпляры объектов.

Ваша задача состоит в том, чтобы переписать данную функцию на класс (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);
```

Задание #2

Вам необходимо создать класс **Person**, от него мы сможем создавать экземпляры людей. Конструктор класса будет принимать 2 параметра:

1. **name** - имя человека
2. **age** - количество полных лет

Также вам необходимо реализовать метод compareAge в классе Person. Он принимает в себя экземпляр класса Person и сравнивает значения полных лет.

Данный метод должен возвращать результат в следующем формате: **Если** у одного **Person** количество лет больше либо равно, чем у другого, то выводите сообщение через alert “name1 старше, чем name2”. **Иначе** же “name1 младше, чем name2”.

Запускайте данный код для теста работы вашего класса **Person**

```
const person1 = new Person('Максим', 24);
const person2 = new Person('Светлана', 36);
const person3 = new Person('Ирина', 23);

person1.compareAge(person2); // Максим младше, чем Светлана
person2.compareAge(person3); // Светлана старше, чем Ирина
person3.compareAge(person1); // Ирина младше, чем Максим
```

Задание #3

Вам поручили задачу на проекте. Необходимо разработать словарь, в который можно добавлять слова с их описанием, удалять слова и получать их.

Для начала создайте класс Dictionary. Его конструктор принимает 1 параметр name. Инициализируйте данное значение при помощи this. Также в конструкторе должно объявляться поле words, которое по умолчанию должно быть пустым объектом.

Вам необходимо в класс Dictionary добавить 4 метода:

1. **add**. Данный метод добавляет новое слово в словарь. Он принимает в себя 2 параметра: **word** - слово (тип данных строка) **description** - описание слова (тип данных строка)

   Присвойте объекту **words** по ключу **word** (этот параметр передали в метод add) значение объекта, в котором будут храниться данные о слове и его описание.

Пример объекта **words** после добавления слова "js":

```
// words
{
    js: {
        word: 'js',
        description: 'язык программирования',
    }
}
```

2. **Если слово** уже существует в объекте **words**, то **не нужно** перезаписывать слово, которое уже было сохранено ранее. **remove**. Данный метод удаляет слово из словаря. Он принимает в себя ключ по которому нужно удалить объект слова из объекта **words**
3. **get.** Данный метод получает слово из словаря. Он принимает в себя ключ из объекта **words.** Из метода необходимо возвратить найденный объект слова по переданному параметру ключа
4. **showAllWords**. Данный метод не принимает в себя никаких параметров. Его задача состоит в том, чтобы вывести все слова, которые есть в объекте **words**, в консоль в формате “word - description” (**word** - само слово, **description** - описание слова)

Для тестирования написанного класса используйте данный код:

```
const dictionary = new Dictionary('Толковый словарь');
dictionary.add('JavaScript', 'популярный язык программирования');
dictionary.add('Веб-разработчик', 'Человек, который создает новые
    сервисы и сайты или поддерживает и дополняет существующие');

dictionary.remove('JavaScript');
dictionary.showAllWords();
// Веб-разработчик - Человек, который создает новые
// сервисы и сайты или поддерживает и дополняет существующие
```

Задание #4

В прошлом задании вы создали словарь. Заказчик доволен проделанной работой и команда разработки тоже. Так как вы уже хорошо разобрались с технологией словаря, то вас попросили реализовать еще один словарь. Заказчик хочет создать новую категорию слов “Сложные слова”.

Используйте код, написанный в прошлом задании. Вам необходимо создать дочерний класс **HardWordsDictionary** от класса **Dictionary**. Конструктор у **HardWordsDictionary** принимает в себя также параметр name и вызывает конструктор родительского класса.

Все методы, которые есть в **Dictionary**, должны остаться без изменений в **HardWordsDictionary**, кроме одного. Вам необходимо переопределить метод **add**, который должен работать по той же логике, что и **add** у **Dictionary**, только объект слова должен выглядеть следующим образом:

```
{
   word: 'word',
   description: 'description',
   isDifficult: true,
}
```

Для теста класса HardWordsDictionary используйте данный код:

```
const hardWordsDictionary = new HardWordsDictionary('Сложные слова');

hardWordsDictionary.add('дилетант', 'Тот, кто занимается наукой или
искусством без
специальной подготовки, обладая только поверхностными знаниями.');

hardWordsDictionary.add('неологизм', 'Новое слово или выражение, а
также новое
значение старого слова.');

hardWordsDictionary.add('квант', 'Неделимая часть какой-либо величины
в физике.');

hardWordsDictionary.remove('неологизм');

hardWordsDictionary.showAllWords();

// дилетант - Тот, кто занимается наукой или искусством
// без специальной подготовки, обладая только поверхностными
знаниями.
// квант - Неделимая часть какой-либо величины в физике.
```

Задание #5

Ваша компания создает внутренний продукт для облегчения контроля за ростом своих разработчиков. Вам поручили задачу реализовать такой продукт.

Для начала вам необходимо создать класс **Developer**, конструктор которого будет принимать 3 параметра:

1. **fullName** - имя разработчика
2. **age** - возраст разработчика
3. **position** - текущая позиция разработчика в компании (например, Junior, Middle, Senior)

Инициализируйте все параметры при помощи **this**. Также создайте внутри конструктора поле **technologies**, которое по умолчанию будет равно пустому массиву.

Кроме этого в классе **Developer** вам необходимо создать 2 метода:

1. **code** - метод, у которого тело изначально пустое (в фигурных скобках ничего нет)
2. **learnNewTechnology** - данный метод принимает в себя 1 параметр technology, который должен добавляться в конец массива technologies

После проделанных действий у вас должен получиться шаблон **Developer**, благодаря которому вы будете создавать дочерние классы.

Вам сейчас необходимо создать 3 дочерних класса от класса **Developer**:

1. **JuniorDeveloper**. Конструктор данного класса принимает 2 параметра: **fullName** и **age**. Вызовите конструктор родительского класса и передайте туда эти 2 параметра. В качестве 3-го у нас выступает **position**. Вам необходимо его указать по умолчанию. Напишите значение “Junior” в качестве 3-го параметра вызова родительского конструктора. Кроме этого переопределить метод code, чтобы он выводил в консоль строку “Junior разработчик пишет код...”. Массив technologies должен содержать следующие технологии: 'HTML', 'CSS', 'JavaScript'
2. **MiddleDeveloper**. Проделайте ту же самую работу в конструкторе, что и JuniorDeveloper. Только на место position передавайте значение “Middle”. Метод code у класса MiddleDeveloper должен выводить в консоль строку “Middle разработчик пишет код...”. Массив technologies должен содержать следующие технологии: 'HTML', 'CSS', 'JavaScript', ‘React’
3. **SeniorDeveloper**. Проделайте ту же самую работу в конструкторе, что и JuniorDeveloper. Только на место position передавайте значение “Senior”. Метод code у класса SeniorDeveloper должен выводить в консоль строку “Senior разработчик пишет код...”. Массив technologies должен содержать следующие технологии: 'HTML', 'CSS', 'JavaScript', ‘React’, ‘NodeJS’

Примечание: когда вы переопределите метод **code** в дочерних классах, то вы используете принцип Полиморфизм.

Тестируйте написанные классы с помощью этого кода:

```
const juniorDeveloper = new JuniorDeveloper('Анастасия', 20)
const middleDeveloper = new MiddleDeveloper('Игорь', 25)
const seniorDeveloper = new SeniorDeveloper('Максим', 30)

juniorDeveloper.code(); // Junior разработчик пишет код...
middleDeveloper.code(); // Middle разработчик пишет код…
seniorDeveloper.code(); // Senior разработчик пишет код...

console.log(juniorDeveloper.fullName, juniorDeveloper.age,
juniorDeveloper.position); // 'Анастасия', 20, 'Junior'
console.log(middleDeveloper.fullName, middleDeveloper.age,
middleDeveloper.position); // 'Игорь', 25, 'Middle'
console.log(seniorDeveloper.fullName, seniorDeveloper.age,
seniorDeveloper.position); // 'Максим', 30, 'Senior'
```

Задание #6 (*дополнительное*)

В заданиях ранее вы реализовали логику для двух словарей и они работают прекрасно. Но тут к вам приходит старший разработчик и говорит, что вам необходимо использовать инкапсуляцию в вашем коде, так как это важный принцип ООП.

Вы конечно же соглашаетесь со старшим разработчиком. Он вам посоветовал сделать поля **name** и **words** приватными. Реализуйте это с помощью знака решетки “**#**”.

Усовершенствуйте свое решение из задания 4

Теперь вам необходимо добавить геттеры и сеттер в класс **Dictionary**, чтобы иметь доступ до приватных переменных.

Для **#name** создайте геттер **mainName** (через ключевое слово get) и сеттер **setMainName** (через ключевое слово set).

Для **#words** создайте геттер **allWords** (через ключевое слово get).

Также создайте **addNewWord** - обычный метод класса, который будет добавлять новое слово в приватный объект **#words** (вместо сеттера). Он должен принимать:

word - слово, тип данных строка

description - описание слова, тип данных строка

Метод **addNewWord** должен только создавать новое слово в объекте **#words** без каких-либо проверок. Он реализуется без **set**, так как:

1. **set** не может принимать в себя больше 1-го параметра.
2. **set** в данном случае логичнее использовать для установки **полностью нового значения**, а не дополнения предыдущего.

Метод **addNewWord** будет использоваться в методе **add**. Мы их разделили так как у них разная зона ответственности:

1. **addNewWord** - отвечает за просто добавление слова в объект. Он используется только внутри классов в методе add.
2. **add** - проверяет, есть ли уже данное слово в словаре, и если слова нет, то вызывает метод **addNewWord**, чтобы добавить новое слово. Метод **add** будет использоваться (вызываться) у экземпляра класса для безопасного добавления новых слов (пример ниже)

Вам необходимо исправить логику для классов **Dictionary** и **HardWordsDictionary**

Итоговый код тестируйте на данном примере:

```
const hardWordsDictionary = new HardWordsDictionary('Сложные слова');

hardWordsDictionary.add('дилетант', 'Тот, кто занимается наукой или
искусством
без специальной подготовки, обладая только поверхностными
знаниями.');

hardWordsDictionary.add('неологизм', 'Новое слово или выражение, а
также новое
значение старого слова.');

hardWordsDictionary.add('квант', 'Неделимая часть какой-либо величины
в физике.');
```

```
hardWordsDictionary.remove('неологизм');

hardWordsDictionary.showAllWords();

console.log(hardWordsDictionary.mainName); // Сложные слова
hardWordsDictionary.setMainName = 'Новый Словарь';
console.log(hardWordsDictionary.mainName); // Новый Словарь
console.log(hardWordsDictionary.allWords); // выводит объект в
котором есть слова
// дилетант и квант
```

Задание #7 (*дополнительное*)

Ваш друг Артем является владельцем автосервиса. Сейчас все заявки на ремонт машин обрабатывается вручную, что конечно же не очень удобно. Поэтому Артем попросил вас как веб-разработчика помочь в автоматизации данного процесса.

Вам необходимо создать класс **CarService**, в конструктор которого будут переданы 2 параметра:

1. **name** - название автосервиса
2. **workingHours** - время работы автосервиса. Объект, с ключами **from и till. from** - время **начала** рабочего дня **till** - время **конца** рабочего дня. Значения времени записываются в формате “h:mm” (например, 9:00)

Инициализируйте данные параметры в конструкторе с помощью **this**. Также, параметр **workingHours** является необязательным. Поэтому задайте значение (тип данных **object**) по умолчанию. Данный объект должен называться **DefaultWorkingHours** и быть статическим (static). Если параметр **workingHours** не был передан, то подставьте в **this.workingHours** значение из **DefaultWorkingHours**, которое равно:

```
static DefaultWorkingHours = {
  from: '9:00',
  till: '20:00',
}
```

Добавьте в класс метод под названием repairCar, который будет чинить машину. Он должен принимать в себя 1 параметр carName - имя машины для ремонта (тип данных string)

Если **carName** не был передан, то отобразите ошибку “Вам необходимо указать название машины, чтобы ее отремонтировать” в консоли через console.error. На этом выполнение функции должно заканчиваться.

Если же параметр **carName** был передан в **repairCar**, то вам необходимо сравнить текущее часы с временем работы автосервиса. Если текущее время (часы) не входит в диапазон работы автосервиса, то отобразите через alert сообщение “К сожалению, мы сейчас закрыты. Приходите завтра”. Иначе же выводите сообщение “Сейчас

отремонтируем вашу машину carName! Ожидайте, пожалуйста” (carName - параметр метода repairCar).

Примечание: при сравнении времени берите в учет только часы, на минуты не обращайте внимание. Пусть минуты всегда будут строкой “00”. **Для теста работоспособности** класса **CarService** используйте данный код:

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

Подсказка 1: для того, чтобы извлечь и строки “8:00” значение часов, вы можете использовать .split(‘:’), получить первый элемент из полученного массива и преобразовать его к числу.