Урок 4.9. Классы

Подготовка к работе с import

Чтобы иметь возможность импортировать код в `script.js`, необходимо произвести некоторые изменения в привычном шаблоне кода урока:

```
<!--index.html-->
<!doctype html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.
0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Frontend. Lesson 26</title>
</head>
<body>
    <!-- Изменить тип подключаемого скрипта, задав атрибуту type значение module-->
    <script type="module" src="script.js"></script>
</body>
</html>
```

```
// script.js
// Убрать use strict, так как строгий режим по умолчанию включается в скриптах с типом mod
ule
// 'use strict';
```

Создание класса

- Класс можно создать прямо в основном скрипте, для этого нужно просто его объявить:

```
// script.js
class Person {
    _firstName;
    _lastName;

    constructor(firstName = '', lastName = '') {
        this._firstName = firstName;
        this._lastName = lastName;
    }
```

```
//    ...
}

// Создать экземпляр класса:
const person = new Person('Andrew', 'Gulin');
```

- Но гораздо интереснее импортировать класс из другого файла
  - Создайте файл `person.js` (так же, как будет называться класс)
  - Внутри файла объявите и экспортируйте класс:

```
// person.js
export default class Person {
// реализация (так же, как и в реализвции выше)
}
```

  - Импортируйте скрипт в основном скрипте и используйте так же, как использовали бы при объявлении класса в этом же скрипте (без импорта):

```
// script.js
import Person from './person.js';
const person = new Person('Andrew', 'Gulin');
```

Анатомия класса

```
// Объявление класса (class и имя с большой буквы - PascalCase)
class Person {
    // Статическая функция
    // Не требует создания экземпляра класса
    // Не имеет доступа к внутренним полям и методам,
    // поскольку экземпляр класса не создаётся
    //
    // Person.isPerson(andrew);
    // andrew - экземпляр, подающийся на вход для проверки,
     // является ли он экземпляром класса Person
    static isPerson(obj) {
        // instanceof проверяет, является ли левый операнд экземпляром
        // правого операнда
        return obj instanceof Person;
    }
    // Приватные поля (технически будут доступны тому, кто будет использовать класс
    // но по условному соглашению приватные поля именуются с _ в начале
    // и не используются при работе с классом извне)
```

```
    _firstName;
    _lastName;
    // Публичные поля
    // Доступны снаружи для чтения и изменения
    // Не рекомендуется создавать публичные поля
    age;

    // Функция, которая вызывается,
    // когда мы пишем конструкцию с new:
     // const person = new Person()
    // Инициализирует класс
    constructor(firstName = '') {
        // Инициализация внутренней приватной переменной
        // значением, переданном в конструктор как параметр:
        // const person = new Person('Andrew');
        this._firstName = firstName;    }
    // Публичный метод (функция)
    // Может быть вызван после создания экземпляра класса:
    // const person = new Person('Andrew');
    // console.log(person.toString());
    toString() {
        return `${this._firstName}`;
    }
    // сеттеры и геттеры
    // Пара set и get имеет одинаковое название
    // Созданы для работы с приватными полями извне
    // const person = new Person();
    // person.firstName = 'Andrew'; - используется set
    // console.log(person.firstName); - используется get
    set firstName(value) {
        this._firstName = value;
    }
    get firstName() {
        return this._firstName;
    }
}
```

Наследование классов

Один класс можно понаследовать от другого, чтобы переиспользовать какие-то поля и методы и не объявлять их заново.Например, класс `Admin` должен иметь все те же поля и методы, что и класс `User`, но ещё и дополнительные поля `permissions` и `email`:

```
// Наследуем класс Admin от User.
class Admin extends User {
}
```