ПМ3 Разработка модулей ПО.

РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.

Тема 1. Введение в ООП.

Лекция 3. 5. Классы в JavaScript.

Цель занятия:

Познакомиться с современным синтаксисом классов, понять их связь с прототипами и преимущества использования.

Учебные вопросы:

1. Понятие класса.
2. Геттеры и сеттеры.
3. Статические свойства и методы.
4. Отличие классов от функций-конструкторов и прототипов.
5. Примеры использования классов.

1. Понятие класса.

Класс в JavaScript — это шаблон для создания объектов. Он позволяет организовать код более структурировано и удобно, особенно при работе с объектно-ориентированным программированием.

Классы в JavaScript можно рассматривать как «синтаксический сахар» над функциями-конструкторами и прототипами.

Это означает, что они предоставляют более удобный и понятный способ создания объектов и работы с их свойствами и методами.

Преимущества:

- Более читаемый и понятный синтаксис.
- Автоматическое создание прототипов для методо

Синтаксис классов.

Класс объявляется с помощью ключевого слова **class**, за которым следует его имя (по соглашению, с заглавной буквы).

```
class Company {
    // Тело класса
}
```

constructor — это специальный метод внутри класса, который вызывается автоматически при создании нового экземпляра. Он используется для инициализации свойств объекта.

```
class Company {
  constructor(name, website) {
    this.name = name;         // свойство
    this.website = website; // свойство
  }
}
```

this внутри конструктора ссылается на создаваемый экземпляр.

Здесь мы присваиваем значения, переданные в качестве аргументов, свойствам name и website нашего объекта.

Свойства можно объявить и вне конструктора (с использованием синтаксиса полей класса):

```
class Company {
  // Поля класса (свойства) объявлены здесь
  name = 'Hexlet';
  website;

  constructor(name, website) {
    // В конструкторе мы можем переопределить значения по умолчанию
    this.name = name;
    this.website = website;
  }

  getName() {
    return this.name;
  }
}
```

Современный синтаксис свойства напрямую в теле класса, без необходимости указывать их в конструкторе.

```
class Person {
    gender;           // свойство
    role = "Guest";   // значение по умолчанию
    salary = 100000; // значение по умолчанию

    constructor(name, age, role) {
        this.name = name; // свойство
        this.age = age;   // свойство
        // свойство можно переопределить
        this.role = role ?? this.role;
    }
}
```

Методы объявляются прямо в теле класса, без использования ключевых слов function или const.

```
class Company {
  constructor(name, website) {
    this.name = name;
    this.website = website;
  }

  // Метод класса
  getName() {
    return this.name;
  }

  // Ещё один метод
  getWebsite() {
    return this.website;
  }
}
```

Все методы, объявленные таким образом, автоматически помещаются в **прототип** класса, что экономит память, так как они не дублируются в каждом экземпляре.

Для создания экземпляра класса используется оператор **new**, который вызывает конструктор класса.

```
class Person {
    gender;           // свойство
    role = "Guest";   // значение по умолчанию
    salary = 100000; // значение по умолчанию

    constructor(name, age, role) {
        this.name = name; // свойство
        this.age = age;   // свойство
        // свойство можно переопределить
        this.role = role ?? this.role;
    }

    getInfo(){
        console.log(this.name, this.age)
    }
}
```

Создание экземпляров класса:

```
const person1 = new Person("Ivan", 22, "Admin");
const person2 = new Person("Anna", 25);
const person3 = new Person();
console.log(person1);
console.log(person2);
console.log(person3);
```

```
                                                    test.js:21
▸ Person {gender: undefined, role: 'Admin', salary: 100000, name:
  'Ivan', age: 22}
                                                    test.js:22
▸ Person {gender: undefined, role: 'Guest', salary: 100000, name:
  'Anna', age: 25}
                                                    test.js:23
▸ Person {gender: undefined, role: 'Guest', salary: 100000, name:
  undefined, age: undefined}
```

2. Геттеры и сеттеры.

Публичные и приватные свойства

По умолчанию все свойства в классах JavaScript являются **публичными**. Это означает, что к ним можно получить доступ, прочитать или изменить их значение из любого места кода.

Публичные свойства объявляются напрямую в классе или в конструкторе с помощью **this**. Они доступны для прямого доступа.

```
class User {
  constructor(name) {
    this.name = name; // Публичное свойство
  }
}

const user = new User('Иван');
console.log(user.name); // 'Иван'
user.name = 'Пётр'; // можно изменить
console.log(user.name); // 'Пётр'
```

С 2020 года в JavaScript появилась возможность объявлять **приватные** свойства с помощью символа **решетки #**.

Доступ к таким свойствам ограничен и возможен **только внутри класса**, в котором они были объявлены.

Попытка обратиться к ним извне класса вызовет ошибку. Это обеспечивает инкапсуляцию, скрывая внутреннюю реализацию объекта.

Пример:

```
class Product {
  #price; // Приватное свойство

  constructor(price) {
    this.#price = price;
  }

  getPrice() {
    // внутри класса есть доступ
    return this.#price;
  }
}

const item = new Product(100);
console.log(item.getPrice()); // 100 - Доступ через метод
// вне класса доступа нет
console.log(item.#price);
// SyntaxError: Private field '#price' must be declared in an enclosing class
```

Геттеры и сеттеры.

Геттеры (get) и сеттеры (set) — это специальные методы, которые позволяют контролировать доступ к свойствам, даже к публичным.

Они создают дополнительный уровень абстракции, позволяя добавлять логику при чтении или записи данных, не нарушая инкапсуляцию.

Геттеры (get).

Геттер — это метод, который вызывается, когда вы пытаетесь прочитать значение свойства.

Он объявляется с ключевым словом **get** и используется как обычное свойство, но может выполнять вычисления или форматирование данных.

Назначение: Возвращать вычисляемые значения, которые зависят от других свойств.

Пример:

```
class Person {
  #firstName;
  #lastName;

  constructor(firstName, lastName) {
    this.#firstName = firstName;
    this.#lastName = lastName;
  }

  get fullName() {
    return `${this.#firstName} ${this.#lastName}`;
  }
}

const person = new Person('Мария', 'Иванова');
// К свойствам нельзя обратиться напрямую:
// console.log(person.#firstName); // SyntaxError
// Доступ возможен только через геттер:
console.log(person.fullName); // Мария Иванова
```

Сеттеры (set).

Сеттер — это метод, который вызывается, когда вы пытаетесь изменить значение свойства. Он объявляется с ключевым словом **set** и принимает один аргумент — новое значение.

Назначение: Добавлять логику при записи, например, валидацию данных или их нормализацию.

Пример:

```
class Circle {
  #radius;

  set radius(value) {
    if (value < 0) {
      throw new Error("Радиус не может быть отрицательным.");
    }
    this.#radius = value;
  }
}

const circle = new Circle();
circle.radius = 10; // Вызывается сеттер
// circle.radius = -5; // Бросит ошибку
```

Итог:

Геттеры и сеттеры особенно полезны в сочетании с приватными свойствами, так как они предоставляют контролируемый способ взаимодействия с данными, которые скрыты от внешнего кода.

Таким образом, вы обеспечиваете, что данные, хранящиеся в экземпляре класса, защищены, а внешний код взаимодействует с объектом только через публичный интерфейс, который вы определили.

Это делает ваш код более надёжным и предсказуемым.

3. Статические свойства и методы.

Статические свойства и статические методы в классах JavaScript принадлежат самому классу, а не его отдельным экземплярам.

Это значит, что для доступа к ним не нужно создавать объект (new ClassName()).

Вы обращаетесь к ним напрямую через имя класса. Они полезны, когда у вас есть данные или функции, которые логически связаны с классом, но не зависят от конкретного объекта.

Статические методы.

Статические методы объявляются с использованием ключевого слова **static**.

Они часто используются для вспомогательных функций, которые не требуют доступа к данным экземпляра.

Например, метод для создания объекта из определённого формата данных или для выполнения общей операции, связанной с классом.

Особенность: Статические методы не имеют доступа к this экземпляра. В них можно обращаться только к другим статическим методам или свойствам.

Синтаксис:

```
class Calculator {
  // статический метод
  static sum(a, b) {
    return a + b;
  }
}

// Вызов статического метода
const result = Calculator.sum(5, 3);
console.log(result); // Выведет: 8
```

Статические свойства также объявляются с ключевым словом **static** и являются общими для всего класса.

Они полезны для хранения констант или настроек, которые не меняются от экземпляра к экземпляру.

Использование: Часто используются для хранения конфигурационных данных, например, API-адреса, или для подсчёта количества созданных экземпляров.

Синтаксис:

```
class MathConstants {
  // статическое свойство
  static PI = 3.14159;
}

// Доступ к статическому свойству
console.log(MathConstants.PI); // Выведет: 3.14159
```

Пример класса с статическим счётчиком.

В этом примере, статическое свойство **count** инициализируется нулём.

Каждый раз, когда создаётся новый экземпляр класса **Person** через конструктор, мы увеличиваем значение этого свойства на единицу.

Это позволяет нам легко отслеживать, сколько всего объектов **Person** было создано.

```
class Person {
  // Статическое свойство для подсчета экземпляров
  static count = 0;

  constructor(name, age) {
    this.name = name;
    this.age = age;
    // Увеличиваем статический счётчик при создании каждого нового объекта
    Person.count++;
  }
}

// Создание экземпляров
const person1 = new Person('Иван', 30);
const person2 = new Person('Мария', 25);
const person3 = new Person('Анна', 33)
// Доступ к статическому свойству напрямую через класс
console.log(`Общее количество созданных персон: ${Person.count}`);
// Вывод: Общее количество созданных персон: 3
```

Итог:
Статические свойства и методы (static):

- Принадлежат классу, а не его экземплярам.
- Доступ к ним осуществляется напрямую через имя класса (ClassName.method()).
- Полезны для общих функций, констант и счетчиков.
- Объявляются с ключевым словом static.
- Существуют в единственном экземпляре для всего класса.

4. Отличие классов от функций-конструкторов и прототипов.

Основные отличия:

- Классы — это современный синтаксис, который был добавлен в ECMAScript 2015 (ES6). Они предлагают более читаемый и привычный для многих разработчиков синтаксис, напоминающий классы в других языках, таких как Java или Python.
- Функции-конструкторы — это традиционный, "старый" способ создания объектов и их инициализации. Они представляют собой обычные функции, которые вызываются с оператором new.
- Прототипы — это основной механизм, на котором строится наследование в JavaScript. Каждая функция-конструктор имеет свойство prototype, которое содержит методы и свойства, доступные для всех экземпляров, созданных с помощью этой функции. Классы, по сути, являются "синтаксическим сахаром" для этого механизма.

Ключевые выводы:

- Классы не заменяют прототипы. Они просто предоставляют более удобный и современный способ работы с ними. Код, написанный с использованием классов, по-прежнему работает на основе прототипного наследования.
- Классы решают проблему "дублирования методов". В старых подходах часто приходилось вручную добавлять методы в prototype, чтобы избежать их создания в каждом экземпляре. Классы делают это автоматически.
- Приватные свойства (#) — это особенность классов, которая не имеет прямого аналога в функциях-конструкторах.
- Стрелочные функции не могут быть использованы как функции-конструкторы, но могут быть методами внутри классов.

5. Примеры использования классов.

Представим, что нам нужно создать простую систему для управления данными о студентах.

Класс **Student** поможет нам структурировать информацию о каждом студенте (имя, курс, оценки) и предоставит методы для работы с этими данными.

```
class Student {
  // Приватные свойства для инкапсуляции
  #name;
  #grades = [];

  // Публичное свойство
  course;

  // Статическое свойство, общее для всех экземпляров
  static studentCount = 0;

  constructor(name, course) {
    this.#name = name;
    this.course = course;

    // Увеличиваем общий счетчик студентов при создании нового экземпляра
    Student.studentCount++;
  }
```

```
// Сеттер для добавления оценок
set addGrade(grade) {
  if (grade >= 0 && grade <= 100) {
    this.#grades.push(grade);
  } else {
    console.error("Оценка должна быть в диапазоне от 0 до 100.");
  }
}

// Геттер для вычисления среднего балла
get averageGrade() {
  if (this.#grades.length === 0) {
    return 0;
  }
  const sum = this.#grades.reduce((total, grade) => total + grade, 0);
  return sum / this.#grades.length;
}
```

```
  // Геттер для получения имени студента
  get name() {
    return this.#name;
  }

  // Статический метод для получения общего количества студентов
  static getStudentsCount() {
    return Student.studentCount;
  }
}
```

Использование класса Student:

```
// Создаем двух студентов
const student1 = new Student('Иван Иванов', 2);
const student2 = new Student('Мария Петрова', 3);

// Добавляем оценки для Ивана через сеттер
student1.addGrade = 85;
student1.addGrade = 90;
student1.addGrade = 78;
student1.addGrade = 110; // Это вызовет ошибку в консоли

// Добавляем оценки для Марии через сеттер
student2.addGrade = 95;
student2.addGrade = 88;
student2.addGrade = 55;
student2.addGrade = 70;
```

```
// Получаем и выводим средний балл каждого студента с помощью геттера
console.log(`Средний балл ${student1.name}: ${student1.averageGrade.toFixed(2)}`);
console.log(`Средний балл ${student2.name}: ${student2.averageGrade.toFixed(2)}`);

// Доступ к публичному свойству
console.log(`Курс Ивана: ${student1.course}`);

// Используем статический метод, чтобы узнать общее количество студентов
console.log(`Всего студентов в системе: ${Student.getStudentsCount()}`);
```

```
▶Оценка должна быть в диапазоне от 0 до 100.
Средний балл Иван Иванов: 84.33
Средний балл Мария Петрова: 77.00
Курс Ивана: 2
Всего студентов в системе: 2
```

Итоги лекции:

Класс — это "синтаксический сахар" над функциями-конструкторами и прототипами, современный синтаксис .

Основной метод в классе — это constructor, который инициализирует свойства объекта при его создании.

Публичные свойства доступны для чтения и записи из любого места.

Приватные свойства (обозначаются #) доступны только внутри класса, обеспечивая инкапсуляцию.

Геттеры (get) и сеттеры (set) — это специальные методы, которые контролируют доступ к свойствам объекта.

Статические свойства и статические методы принадлежат самому классу, а не его экземплярам. Чтобы получить к ним доступ, не нужно создавать объект.

Контрольные вопросы:

- Чем класс отличается от функции-конструктора?
- Для чего нужен метод constructor в классе?
- Что такое статический метод и чем он отличается от метода экземпляра?
- Как классы связаны с прототипами?
- Можно ли вызвать класс как обычную функцию без new?

Домашнее задание:

https://ru.hexlet.io/courses/

Материалы лекций:

https://github.com/ShViktor72/Education2025

Обратная связь:

colledge20education23@gmail.com