ПМ3 Разработка модулей ПО.

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

Тема 1. Введение в ООП.

Лекция 6. Методы объектов: упаковка, распаковка, toString()

Цель занятия:

Познакомиться с процессами упаковки (boxing) и распаковки (unboxing) примитивов в JavaScript, а также с методами toString() и valueOf(), применяемыми для преобразования объектов к примитивным типам.

Учебные вопросы:

1. Примитивы и объекты в JavaScript.
2. Концепция "упаковки" (Boxing).
3. Концепция "распаковки" (Unboxing).
4. Методы toString(), valueOf()

1. Примитивы и объекты в JavaScript.

В JavaScript все данные делятся на две основные категории: примитивы и объекты.

Ключевое различие между ними заключается в том, как они хранятся и как с ними взаимодействует движок JavaScript.

Примитивные типы данных.

Примитивы — это простейшие типы данных, которые хранят одно значение и являются неизменяемыми. Это означает, что после создания их значение нельзя изменить.

Вместо этого, любая операция, которая, казалось бы, изменяет примитив, на самом деле создаёт новое значение.

Значение по ссылке: Когда вы присваиваете примитив переменной, она хранит само значение.

Когда вы копируете переменную с примитивом, вы создаёте независимую копию этого значения.

Примеры примитивов:

- string (строка)
- number (число)
- boolean (логическое значение)
- null (нулевое значение)
- undefined (неопределённое значение)
- symbol (уникальный идентификатор)
- bigint (большое целое число)

Пример:

```
let a = 10;
let b = a; // b получает копию значения a
b = 20;
console.log(a); // 10, значение 'a' не изменилось
```

Объекты.

Объекты — это сложные, изменяемые структуры данных.

Они хранятся в памяти по ссылке, а не по значению.

Когда вы присваиваете объект переменной, она хранит не сам объект, а только адрес (ссылку) на него в памяти.

Значение по ссылке:

- Если вы копируете переменную с объектом, вы копируете только ссылку.
- Обе переменные будут указывать на один и тот же объект в памяти.
- Изменение объекта через одну переменную отразится на другой.

Примеры объектов:

- Object (общий объект)
- Array (массив)
- Function (функция)
- Date (дата)
- Map, Set и другие

Пример:

```
let obj1 = { value: 10 };
let obj2 = obj1; // obj2 получает ссылку на тот же объект
obj2.value = 20;
console.log(obj1.value); // 20, объект изменился через вторую переменную
```

Краткое сравнение		
Характеристика	Примитивы	Объекты
Изменяемость	Неизменяемые	Изменяемые
Хранение	По значению	По ссылке
Копирование	Создаётся независимая копия	Копируется только ссылка на объект

2. Концепция "упаковки" (Boxing).

В JavaScript "упаковка" (boxing) — это автоматический процесс, при котором примитивный тип данных временно преобразуется в объект-обёртку.

Этот механизм позволяет нам использовать методы и свойства, которые доступны только для объектов, на примитивных значениях.

Как это работает?

Представьте, что у вас есть строка-примитив, например, 'hello’.

У примитивов, в отличие от объектов, нет методов.

Но что произойдет, если вы вызовете метод toUpperCase()?

```
const str = 'hello';
const upperStr = str.toUpperCase();
console.log(upperStr); // 'HELLO'
```

Этот код работает, потому что JavaScript выполняет следующие шаги:

- Создание временного объекта-обёртки: Движок JavaScript видит, что вы пытаетесь вызвать метод на примитиве. Он временно создает объект String с этим примитивным значением.
- Вызов метода: Метод toUpperCase() вызывается на этом временном объекте.
- Уничтожение объекта: После выполнения операции временный объект-обёртка уничтожается, а результат (новое примитивное значение) возвращается.

Этот процесс называется автобоксингом (auto-boxing) и происходит неявно "за кулисами".

Он применим ко всем примитивным типам, у которых есть соответствующие встроенные объекты-обёртки:

- String для строк
- Number для чисел
- Boolean для логических значений

Зачем нужна упаковка?

Упаковка позволяет использовать богатый функционал объектов для работы с простыми данными, не жертвуя при этом эффективностью примитивов.

Вам не нужно вручную создавать объект new String('hello'), чтобы использовать его методы. Это делает код более чистым и интуитивно понятным.

Пример с Number:

```
const num = 123.456;
const fixedNum = num.toFixed(2); // Вызывается метод на примитиве
console.log(fixedNum); // '123.46'
```

Здесь JavaScript временно превращает 123.456 в объект Number и вызывает его метод toFixed(). После этого, временный объект Number исчезает.

Суть "упаковки":

- Примитивы в JavaScript не имеют методов, но когда вы пытаетесь вызвать метод на примитивном значении (например, .toUpperCase() на строке), движок JavaScript неявно выполняет следующее:
- Создает временный объект-обёртку, который содержит это примитивное значение.
- Вызывает метод на этом временном объекте.
- Уничтожает временный объект после выполнения операции, возвращая результат.

3. Концепция "распаковки" (Unboxing

"Распаковка" (unboxing) — это обратный процесс "упаковки".

Она происходит, когда JavaScript преобразует объект-обёртку обратно в его примитивное значение.

Этот процесс также выполняется автоматически "за кулисами", когда это необходимо.

Как это работает?

Распаковка происходит, когда объект используется в контексте, где ожидается примитивное значение, например, в математических операциях или при сравнении. JavaScript делает это, вызывая специальные методы объекта: valueOf() и, если необходимо, toString().

- valueOf(): Этот метод возвращает примитивное значение, связанное с объектом.
- toString(): Этот метод возвращает строковое представление объекта.

Движок JavaScript пытается сначала вызвать valueOf(); если он возвращает примитив, этот результат используется. Если valueOf() не возвращает примитив или не существует, движок пытается вызвать toString().

Пример с valueOf()

Рассмотрим объект, созданный вручную с помощью конструктора Number.

```
const objNum = new Number(10); // Создан объект-обёртка
const result = objNum + 5; // Происходит распаковка
console.log(result); // 15
```

Здесь JavaScript видит, что вы пытаетесь сложить объект (objNum) с числом (5). Для выполнения операции он вызывает valueOf() на objNum, который возвращает примитивное значение 10. Затем происходит сложение.

Пример с toString()

В некоторых случаях, когда ожидается строка, используется toString().

```
const objNum = new Number(10);
const str = "Value is " + objNum; // Происходит распаковка в строку

console.log(str); // "Value is 10"
```

В этом примере JavaScript преобразует objNum в строку, вызывая toString() (так как сложение со строкой требует строкового значения).

Суть "распаковки" (Unboxing)

"Распаковка" — это автоматический процесс, при котором JavaScript преобразует объект-обёртку обратно в его примитивное значение.

Это происходит неявно, когда объект используется в контексте, где ожидается примитив (например, в математических операциях).

Движок JavaScript для этого использует методы valueOf() и toString(), чтобы получить примитивное значение объекта.

4. Методы toString(), valueOf()

Метод toString() возвращает строковое представление объекта.

По умолчанию, если вы не переопределите этот метод в своём классе, он вернёт строку, указывающую на тип объекта, например [object Object].

Однако для встроенных типов данных, таких как массивы, даты или числа, toString() уже имеет полезную реализацию.

Как работает: Этот метод вызывается автоматически, когда JavaScript ожидает получить строку, например, при конкатенации строк (+) или при использовании объекта в строковом шаблоне.

Использование: Вы можете переопределить toString() в своём классе, чтобы он возвращал более осмысленную и информативную строку.

```
// Поведение по умолчанию
const obj = {};
console.log(obj.toString()); // "[object Object]"

// Переопределённый toString() в собственном классе
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  toString() {
    return `Person: ${this.name}, Age: ${this.age}`;
  }
}

const person = new Person('Иван', 30);
console.log(String(person)); // "Person: Иван, Age: 30"
console.log(`The person is ${person}.`); // "The person is Person: Иван, Age: 30."
```

Метод valueOf() возвращает примитивное значение объекта.

Он используется, когда JavaScript ожидает получить примитив, например, в математических операциях или при сравнении.

Как работает: Этот метод вызывается автоматически перед toString(), когда нужно преобразовать объект в примитив.

Если valueOf() возвращает примитивное значение (число, строку, булево), то toString() не вызывается.

Если valueOf() возвращает объект, то движок попытается вызвать toString().

Использование: Вы можете переопределить valueOf() для создания объектов, которые ведут себя как примитивы в математических контекстах.

```
class Temperature {
  constructor(celsius) {
    this.celsius = celsius;
  }
  // Возвращает примитивное значение (число)
  valueOf() {
    return this.celsius;
  }
  toString() {
    return `${this.celsius}°C`;
  }
}

const temp = new Temperature(25);
// Использование valueOf() в математической операции
const result = temp + 10;
console.log(result); // 35
// Использование toString() в строковом контексте
console.log(`Температура: ${temp}`); // "Температура: 25°C"
```

В этом примере, при сложении temp + 10, вызывается valueOf(), который возвращает 25.

А при интерполяции строки toString() возвращает форматированную строку.

Итоги лекции:

- Примитивы не имеют методов. Это простые, неизменяемые значения.
- Упаковка (Boxing) — автоматическое преобразование примитива ('abc') в объект-обёртку (new String('abc')) для вызова методов.
- Распаковка (Unboxing) — обратный процесс: преобразование объекта-обёртки обратно в примитив.
- toString() — метод, который возвращает строковое представление объекта.
- valueOf() — метод, который возвращает примитивное значение объекта. Вызывается перед toString() в контексте, где ожидается примитив.

Контрольные вопросы:

- Что такое упаковка (boxing) и зачем она нужна в JavaScript?
- Приведите пример, когда примитив number временно превращается в объект-обёртку.
- В чём разница между методами toString() и valueOf()?
- Как можно изменить поведение метода toString() для вашего собственного класса, чтобы он возвращал кастомную строку?

Домашнее задание:

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

Материалы лекций:

https://github.com/ShViktor72/Education2025

Обратная связь:

colledge20education23@gmail.com