Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 13**

Инструкционно-технологическая карта

Тема: Способы применения ключевого слова this.

Цель: Научиться использовать ключевое слово this различными способами.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Теоретические сведения

## Методы объекта, «this»

Объекты обычно создаются, чтобы представлять сущности реального мира, будь то пользователи, заказы и так далее:

// Объект пользователя

let user = {

name: "Джон",

age: 30

};

И так же, как и в реальном мире, пользователь может *совершать действия*: выбирать что-то из корзины покупок, авторизовываться, выходить из системы, оплачивать и т.п.

Такие действия в JavaScript представлены свойствами-функциями объекта.

## Примеры методов

Для начала нужно научить нашего пользователя (user) здороваться:

let user = {

name: "Джон",

age: 30

};

user.sayHi = function() {

alert("Привет!");

};

user.sayHi(); // Привет!

Здесь была использована Function Expression (функциональное выражение), чтобы создать функцию для приветствия, и присвоена её свойству user.sayHi нашего объекта.

Затем был произведен ее вызов. Теперь пользователь может говорить!

Функцию, которая является свойством объекта, называют *методом* этого объекта.

Итак, получен метод sayHi объекта user.

Конечно, можно было заранее объявить функцию и использовать её в качестве метода, примерно так:

let user = {

// ...

};

// сначала объявление

function sayHi() {

alert("Привет!");

};

// затем добавление в качестве метода

user.sayHi = sayHi;

user.sayHi(); // Привет!

### Объектно-ориентированное программирование

Когда пишется код и используются объекты для представления сущностей реального мира, – это называется объектно-ориентированное программирование или сокращённо: «ООП».

ООП является большой предметной областью и интересной наукой само по себе. Как выбрать правильные сущности? Как организовать взаимодействие между ними? Это – создание архитектуры, и есть хорошие книги по этой теме, такие как «Приёмы объектно-ориентированного проектирования. Паттерны проектирования» авторов Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес или «Объектно-ориентированный анализ и проектирование с примерами приложений» Гради Буча, а также ещё множество других книг.

## Сокращённая запись метода

Существует более короткий синтаксис для методов в литерале объекта:

// эти объекты делают одно и то же (одинаковые методы)

user = {

sayHi: function() {

alert("Привет");

}

};

// сокращённая запись выглядит лучше

user = {

sayHi() { // то же самое, что и "sayHi: function()"

alert("Привет");

}

};

Как было показано, допускается пропустить ключевое слово «function» и просто написать sayHi().

Нужно отметить, что эти две записи не полностью эквивалентны. Есть тонкие различия, связанные с наследованием объектов (что будет рассмотрено позже), но на данном этапе изучения – это неважно. В большинстве случаев сокращённый синтаксис предпочтителен.

## Ключевое слово «this» в методах

Как правило, методу объекта необходим доступ к информации, которая хранится в объекте, чтобы выполнить с ней какие-либо действия (в соответствии с назначением метода).

Например, коду внутри user.sayHi() может понадобиться имя пользователя, которое хранится в объекте user.

Для доступа к информации внутри объекта метод может использовать ключевое слово this.

Значение this – это объект «перед точкой», который использовался для вызова метода.

Например:

let user = {

name: "Джон",

age: 30,

sayHi() {

// this - это "текущий объект"

alert(this.name);

}

};

user.sayHi(); // Джон

Здесь во время выполнения кода user.sayHi() значением this будет являться user (ссылка на объект user).

Технически также возможно получить доступ к объекту без ключевого слова this, ссылаясь на него через внешнюю переменную (в которой хранится ссылка на этот объект):

let user = {

name: "Джон",

age: 30,

sayHi() {

alert(user.name); // использование переменной "user" вместо ключевого слова "this"

}

};

Но такой код будет ненадёжным. Если будет принято решение скопировать ссылку на объект user в другую переменную, например, admin = user, и перезапишем переменную user чем-то другим, тогда будет осуществлён доступ к неправильному объекту при вызове метода из admin.

Это показано ниже:

let user = {

name: "Джон",

age: 30,

sayHi() {

alert( user.name ); // приведёт к ошибке

}

};

let admin = user;

user = null; // обнуление переменной для наглядности, теперь она не хранит ссылку на объект.

admin.sayHi(); // Ошибка! Внутри sayHi() используется user, которая больше не ссылается на объект!

Если используется this.name вместо user.name внутри alert, тогда этот код будет работать.

## «this» не является фиксированным

В JavaScript ключевое слово «this» ведёт себя иначе, чем в большинстве других языков программирования. Оно может использоваться в любой функции.

В этом коде нет синтаксической ошибки:

function sayHi() {

alert( this.name );

}

Значение this вычисляется во время выполнения кода и зависит от контекста.

Например, здесь одна и та же функция назначена двум разным объектам и имеет различное значение «this» при вызовах:

let user = { name: "Джон" };

let admin = { name: "Админ" };

function sayHi() {

alert( this.name );

}

// использование одной и той же функцию в двух объектах

user.f = sayHi;

admin.f = sayHi;

// вызовы функции, приведённые ниже, имеют разное значение this

// "this" внутри функции является ссылкой на объект, который указан "перед точкой"

user.f(); // Джон (this == user)

admin.f(); // Админ (this == admin)

admin['f'](); // Админ (неважен способ доступа к методу - через точку или квадратные скобки)

Правило простое: при вызове obj.f() значение this внутри f равно obj. Так что, в приведённом примере это user или admin.

## Вызов без объекта: this == undefined

Есть возможность вызова функции вовсе без использования объекта:

function sayHi() {

alert(this);

}

sayHi(); // undefined

В строгом режиме ("use strict") в таком коде значением this будет являться undefined. Если будет произведена попытка получить доступ к name, используя this.name – это вызовет ошибку.

В нестрогом режиме значением this в таком случае будет глобальный объект (. Это – исторически сложившееся поведение this, которое исправляется использованием строгого режима («use strict»).

Обычно подобный вызов является ошибкой программирования. Если внутри функции используется this, тогда ожидается, что она будет вызываться в контексте какого-либо объекта.

## Последствия свободного this

В других языках программирования применяется принцип «фиксированного this» – когда методы, определённые внутри объекта, всегда сохраняют в качестве значения this ссылку на свой объект (в котором был определён метод).

В JavaScript this является «свободным», его значение вычисляется в момент вызова метода и не зависит от того, где этот метод был объявлен, а зависит от того, какой объект вызывает метод (какой объект стоит «перед точкой»).

Эта идея вычисления this в момент исполнения имеет как свои плюсы, так и минусы. С одной стороны, функция может быть повторно использована в качестве метода у различных объектов (что повышает гибкость). С другой стороны, большая гибкость увеличивает вероятность ошибок.

## Внутренняя реализация: Ссылочный тип

Некоторые хитрые способы вызова метода приводят к потере значения this, например:

let user = {

name: "Джон",

hi() { alert(this.name); },

bye() { alert("Пока"); }

};

user.hi(); // Джон (простой вызов метода работает хорошо)

// теперь попытка вызывать user.hi или user.bye

// в зависимости от имени пользователя user.name

(user.name == "Джон" ? user.hi : user.bye)(); // Ошибка!

В последней строчке кода используется условный оператор ?, который определяет, какой будет вызван метод (user.hi или user.bye) в зависимости от выполнения условия. В данном случае будет выбран user.hi.

Затем метод тут же вызывается с помощью скобок (). Но вызов не работает как положено!

Можно увидеть, что при вызове будет ошибка, потому что значением "this" внутри функции становится undefined (строгий режим).

Так работает (доступ к методу объекта через точку):

user.hi();

Так уже не работает (вызываемый метод вычисляется):

(user.name == "Джон" ? user.hi : user.bye)(); // Ошибка!

Почему? Если есть желание понять, почему так происходит, можно разобраться, как работает вызов методов (obj.method()).

Присмотревшись поближе, в выражении obj.method() можно заметить две операции:

1. Сначала оператор точка '.' возвращает свойство объекта – его метод (obj.method).
2. Затем скобки () вызывают этот метод (исполняется код метода).

Итак, каким же образом информация о this передаётся из первой части во вторую?

Если поместить эти операции в отдельные строки, то значение this, естественно, будет потеряно:

let user = {

name: "Джон",

hi() { alert(this.name); }

};

// получение метода объекта и его вызов в разных строках

let hi = user.hi;

hi(); // Ошибка, потому что значением this является undefined

Здесь hi = user.hi сохраняет функцию в переменной, и далее в последней строке она вызывается полностью сама по себе, без объекта, так что нет this.

Для работы вызовов типа user.hi(), JavaScript использует трюк – точка '.' возвращает не саму функцию, а специальное значение «ссылочного типа», называемого Reference Type.

Этот ссылочный тип (Reference Type) является внутренним типом. Он используется внутри языка.

Значение ссылочного типа – это «триплет»: комбинация из трёх значений (base, name, strict), где:

1. base – это объект.
2. name – это имя свойства объекта.

strict – это режим исполнения. Является true, если действует строгий режим (use strict).

Результатом доступа к свойству user.hi является не функция, а значение ссылочного типа. Для user.hi в строгом режиме оно будет таким:

// значение ссылочного типа (Reference Type)

(user, "hi", true)

Когда скобки () применяются к значению ссылочного типа (происходит вызов), то они получают полную информацию об объекте и его методе, и могут поставить правильный this (=user в данном случае, по base).

Ссылочный тип – исключительно внутренний, промежуточный, используемый, чтобы передать информацию от точки . до вызывающих скобок ().

При любой другой операции, например, присваивании hi = user.hi, ссылочный тип заменяется на собственно значение user.hi (функцию), и дальше работа уже идёт только с ней. Поэтому дальнейший вызов происходит уже без this.

Таким образом, значение this передаётся правильно, только если функция вызывается напрямую с использованием синтаксиса точки obj.method() или квадратных скобок obj['method']() (они делают то же самое). Позднее в этом учебнике мы изучим различные варианты решения проблемы потери значения this. Например, такие как func.bind().

## У стрелочных функций нет «this»

Стрелочные функции особенные: у них нет своего «собственного» this. Если использовать this внутри стрелочной функции, то его значение берётся из внешней «нормальной» функции.

Например, здесь arrow() использует значение this из внешнего метода user.sayHi():

let user = {

firstName: "Илья",

sayHi() {

let arrow = () => alert(this.firstName);

arrow();

}

};

user.sayHi(); // Илья

Это является особенностью стрелочных функций. Они полезны, когда на самом деле не хочется иметь отдельное значение this, а брать его из внешнего контекста.

## Итого

1. Функции, которые находятся в объекте в качестве его свойств, называются «методами».
2. Методы позволяют объектам «действовать»: object.doSomething().
3. Методы могут ссылаться на объект через this.
4. Значение this определяется во время исполнения кода.
5. При объявлении любой функции в ней можно использовать this, но этот this не имеет значения до тех пор, пока функция не будет вызвана.
6. Эта функция может быть скопирована между объектами (из одного объекта в другой).
7. Когда функция вызывается синтаксисом «метода» – object.method(), значением this во время вызова является объект перед точкой.

Также ещё раз заметим, что стрелочные функции являются особенными – у них нет this. Когда внутри стрелочной функции обращаются к this, то его значение берётся снаружи.

### Задания:

**Задание 1.**

Создайте объект calculator (калькулятор) с тремя методами:

read() (читать) запрашивает два значения и сохраняет их как свойства объекта.

sum() (суммировать) возвращает сумму сохранённых значений.

mul() (умножить) перемножает сохранённые значения и возвращает результат.

**Задание 2.**

Существует объект ladder (лестница) – объект, который позволяет подниматься вверх и спускаться:

let ladder = {

step: 0,

up() {

this.step++;

},

down() {

this.step--;

},

showStep: function() { // показывает текущую ступеньку

alert( this.step );

}

};

Теперь, если нужно сделать несколько последовательных вызовов, можно выполнить это так:

ladder.up();

ladder.up();

ladder.down();

ladder.showStep(); // 1

Измените код методов up, down и showStep таким образом, чтобы их вызов можно было сделать по цепочке, например так:

ladder.up().up().down().showStep(); // 1

# Контрольные вопросы

1. Как называется функцкия внутри объекта?
2. Для чего используется this в методах объекта?

**Литература**

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А. В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич