Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебная дисциплина «Программные средства создания Internet-приложений»

Инструкция
по выполнению лабораторной работы
«Создание объектов и методов. Управление контекстом вызова»

Минск
2020

Лабораторная работа № 20

Тема работы: Создание объектов и методов. Управление контекстом вызова

1. Цель работы

Формирование умений создания классов, пользовательских объектов, их свойств и методов.

2. Задание

Выполнить задания в соответствии с порядком выполнения лабораторной работы.

3. Оснащение работы

ПК, редактор исходного кода, браузер.

4. Основные теоретические сведения

Для создания новых объектов в JavaScript имеются следующие способы:

1) *литеральный способ – с помощью инициализатора объекта.*
2) *другой способ – создать функцию-конструктор и сделать экземпляр объекта с помощью этой функции и оператора new;*
3) *с помощью метода Object.create. Этот метод очень удобен, так как позволяет указывать объект прототип для нового объекта без определения функции конструктора.*

Пустой объект может быть создан одним из двух синтаксисов: с помощью литерала объекта или оператора new с конструктором:

```
1. o = new Object();
2. o = {}; // пустые фигурные скобки
```

Использование инициализатора объекта. При этом сразу задается как объект, так и значения. Для этого используется следующий синтаксис:

```
ИмяОбъекта = { Свойство1: Значение1, Свойство2: Значение2, ..., СвойствоN: ЗначениеN }
```

Следующие два фрагмента кода создают одинаковый объект:

```
let menuSetup = {
  width: 300,
  height: 200,
  title: "Menu"
};

// то же самое, что:

let menuSetup = {};
menuSetup.width = 300;
menuSetup.height = 200;
menuSetup.title = 'Menu';
```

Названия свойств можно перечислять как в кавычках, так и без, если они удовлетворяют ограничениям для имён переменных.

```
let menuSetup = {
  width: 300,
  'height': 200,
  "мама мыла раму": true
```

```
};
```

Объект может иметь свойство, которое будет другим объектом.

```
let myHonda = {
  color: "red",
  wheels: 4,
  engine: {
    cylinders: 4,
    size: 2.2
  }
};
myHonda.engine.cylinders;
```

Использование функции конструктора.

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
let mycar = new Car("Eagle", "Talon TSi", 1993);
```

Объекты также можно создавать с помощью метода **Object.create**.

```
let Animal = {
  type: 'Invertebrates', // Значение type по умолчанию
  displayType: function() {  // Метод отображающий тип объекта Animal
    console.log(this.type);
  }
};

let animal1 = Object.create(Animal);
animal1.displayType(); // Выведет:Invertebrates
```

Для доступа к свойству используется один из операторов доступа: . (точка) или [] (квадратные скобки):

```
let o = {};
o.x = 5;       // Добавили новое свойство
o["y"] = 10;   // Добавили новое свойство
```

Для того, чтобы проверить, есть ли в объекте свойство с определенным ключом используется оператор: "in".

```
"prop" in obj
```

причем имя свойства – в виде строки, например:

```
if ("name" in person) {
  alert( "Свойство name существует!" );
}
```

Чтобы создать метод, следует, прежде всего, написать функцию, которая будет его реализовывать, а затем указать его в конструкторе объекта.

Добавление метода в объект осуществляется простым присвоением функции function(n) { ... }свойству.

Вызов метода осуществляется точно также, как и вызов обычной функции – с помощью оператора () (оператор вызова):

```
<body>
<div id="test">
</div>
<script>
function Person(name, age, sex) {
  this.name = name;
```

```
  this.age = age;
  this.sex = sex;
}

function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
  this.displayCar = displayCar;
}

function displayCar() {
let str = "<table border=\"1\">";
 str += "<tr><td>Производитель:</td><td>"+this.make+"</td></tr>";
 str += "<tr><td>Модель:</td><td>"+this.model+"</td></tr>";
 str += "<tr><td>Год выпуска:</td><td>"+this.year+"</td></tr>";
 str                 +=                  "<tr><td>Владелец:</td><td>"+this.owner.name+",
"+this.owner.age+"л.</td></tr>";
 str += "</table>";
 document.getElementById("test").outerHTML =str;
document.write(str);
}
let MyCar = new Car("Ford", "Escort", 1997, new Person("Petrov",44,"M"));
MyCar.displayCar();
</script>
</body>
```

Свойства объекта, кроме пар «ключ-значение», также имеют дополнительные флаги конфигурации.

Помимо значения value, свойства объекта имеют три специальных атрибута (так называемые «флаги»):

writable – если true, свойство можно изменить, иначе оно только для чтения;

enumerable – если true, свойство перечисляется в циклах, в противном случае циклы его игнорируют;

configurable – если true, свойство можно удалить, а эти атрибуты можно изменять, иначе этого делать нельзя.

Когда свойство создано «обычным способом», все они имеют значение true. Но можно изменить их в любое время.

Метод **Object.getOwnPropertyDescriptor** позволяет получить полную информацию о свойстве:

Возвращаемое значение – это объект, так называемый «дескриптор свойства»: он содержит значение свойства и все его флаги.

```
let user = {
  name: "John"
};

let descriptor = Object.getOwnPropertyDescriptor(user, 'name');

console.log (descriptor);      //      {value: "John", writable: true, enumerable: true,
                                       configurable: true}
```

Чтобы изменить флаги, можно воспользоваться методом **Object.defineProperty**.

Если свойство существует, defineProperty обновит его флаги. В противном случае метод создаёт новое свойство с указанным значением и флагами; если какой-либо флаг не указан явно, ему присваивается значение false.

```
let user = {};

Object.defineProperty(user, "name", {
  value: "John"
```

```
});

let descriptor = Object.getOwnPropertyDescriptor(user, 'name');
console.log (descriptor);//    {value: "John", writable: false, enumerable: false,
                               configurable: false}
```

Чтобы сделать свойство user.name доступным только для чтения, необходимо изменить флаг writable: присвоить им значения false в параметре descriptor.

```
let user = {
  name: "John"
};

Object.defineProperty(user, "name", {
  writable: false
});

user.name = "Pete"; // Ошибка: Невозможно изменить доступное только для чтения свойство
'name'
```

Действия, нарушающие ограничения флагов, в нестрогом режиме просто молча игнорируются.

Основные операции, производимые с объектами, – это добавление новых свойств, изменение уже существующих свойств, удаление свойств и обращение к свойствам.

Метод **Object.create()** создаёт новый объект с указанными объектом прототипа и свойствами.

```
Object.create(proto[, propertiesObject])
```

Выбрасывает исключение TypeError, если параметр **proto** не является **null** или **объектом**.

```
var d = Object.create(null);
// d ---> null
console.log(d.hasOwnProperty);
// undefined, т.к. 'd' не наследуется от Object.prototype
```

В современном JavaScript есть и более продвинутая конструкция «class», которая предоставляет новые возможности, полезные для объектно-ориентированного программирования.

Базовый синтаксис «class» выглядит так:

```
class MyClass {
  prop = value; // свойство
  constructor(...) { // конструктор
    // ...
  }
  method(...) {} // метод
  get something(...) {} // геттер
  set something(...) {} // сеттер
  [Symbol.iterator]() {} // метод с вычисляемым именем (здесь - символом)
  // ...
}
```

Для создания нового объекта со всеми перечисленными методами необходимо выполнить вызов **new MyClass()**.

При этом автоматически вызывается метод **constructor()**, в нём можно инициализировать объект.

```
class User {
constructor(name) {
  this.name = name;
}
sayHi() {
```

```
  console.log(this.name);
}
}

// Использование:
let user = new User("Иван");
user.sayHi();           //Иван
```

Функция constructor запускается при создании new User, остальные методы записываются в User.prototype.

В отличие от Функции-конструктора объявление класса с точки зрения области видимости ведёт себя как **let**. В частности, оно видно только в текущем блоке и только в коде, который находится ниже объявления (Function Declaration видно и до объявления).

Методы, объявленные внутри class, имеют ряд особенностей:

- метод sayHi является именно методом, то есть имеет доступ к **super**;
- все методы класса работают в строгом режиме **use strict**, даже если он не указан;
- все методы класса не перечислимы. То есть в цикле **for..in** по объекту их не будет.

Синтаксис классов отличается от литералов объектов. Внутри классов запятые не требуются: методы в классе не разделяются запятой.

В классах можно объявлять вычисляемые свойства, геттеры/сеттеры и т.д., а также использовать […] для свойств с вычисляемыми именами

```
class MyClass {
  chislo= prompt('Введите число', 1); // свойство
  constructor(power) { // конструктор
    this.power=power;
  }
  vpow() {
    console.log(Math.pow(this.chislo,this.power))

  }
  //get something(...) {} // геттер
  //set something(...) {} // сеттер
 // [Symbol.iterator]() {} // метод с вычисляемым именем (здесь - символом)
  // ...
}
let x=new MyClass(3);
x.chislo=5;
x.vpow(); //125
```

При добавлении к объекту нового свойства, создаётся новое собственное свойство (own property).

Ключевые слова, реализующие классы: "class", "constructor", "static", "extends" и "super".

Метод **constructor** – специальный метод, необходимый для создания и инициализации объектов, созданных, с помощью класса.

Ключевое слово **super** можно использовать в методе **constructor** для вызова конструктора родительского класса (в примере выше используется для инициализации сторон Квадрата).

Ключевое слово **static**, определяет статический метод для класса. Статические методы **не могут** быть вызваны у экземпляров (instance) класса. Статические методы, часто используются для создания служебных функций для приложения.

Ключевое слово **extends** используется в объявлениях классов и выражениях классов для создания класса, дочернего относительно другого класса.

5. Порядок выполнения работы

1. Создайте объект в соответствии с вариантом 3-мя различными способами:

Вариант	Задания
1	Создайте объект Сотрудник, который будет хранить сведения о сотруднике некоторой фирмы, такие как Имя, Отдел, Телефон, Зарплата. И метод, который выводит название отдела и номер телефона, в котором работает сотрудник.
2	Создайте объект Книга, который будет хранить сведения о книге, такие как Название, Автор, Год Издания, Издательство. И метод, который выводит информацию о книгах в таблицу.
3	Создайте объект Кафе, который будет хранить сведения о кафе, такие как Название, Адрес, Ценовая категория, Количество столиков. И метод, который запрашивает имя пользователя и выводит персональное сообщение с адресом Кафе.
4	Создайте объект Блюдо, который будет хранить сведения о блюде, такие как Название, Ингредиенты, Калорийность, Кухня, Время приготовления. И метод, который будет запрашивать ингредиент и выводить сообщение – имеется ли такой ингредиент в блюде.
5	Создайте объект Человек, который будет хранить сведения о человеке, такие как Имя, Возраст, Пол, Социальный статус. И метод, который будет в зависимости от пола, будет запрашивать информацию: например, м- вы занимаетесь спортом?, ж – вы занимаетесь фитнесом?

2. Для созданного объекта, используя флаги конфигурации, для первого свойства задайте запрет на изменение значения свойства, для второго свойства добавьте запрет на удаление свойства, а для третьего свойства запретите вывод в циклах.

3. Создать класс «Сказочный герой», у которого имеется свойство с названием сказки (из которой герой), и метод, который выводит, что указанный герой является персонажем указанной сказки. Создать класс «Персонаж», который является наследником класса «Сказочный герой» и имеет свойство, которое содержит свойство: отрицательный или положительный персонаж. Кроме того, добавьте в класс метод с одноименным названием из родительского класса, который выводит сообщение, с указанием имени, характера (отрицательный/положительный) и названием сказки. Создайте 3 положительных персонажа из одной сказки и 3 отрицательных персонажа из разных сказок.

4. Выведите описание созданных персонажей 2-мя разными способами.

5*. Сохраните информацию о созданных объектах в массиве. Выполните вывод информации о персонажах, хранящихся в массиве с помощью функции из задания 4 с использованием управления контекстом вызова.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

7. Контрольные вопросы и задания

1. Что представляет собой объект в JavaScript?
2. Какой тип данных можно использовать в качестве ключей свойств?
3. В чем отличие операторов доступа к свойствам объектов?
4. Какие способы проверки наличия свойства в объекте вы знаете?
5. Как можно создать новый объект и повторить структуру уже имеющегося объекта?
6. Что представляет собой контекст вызова this? Чем определяется значение this?
7. Приведите пример использования ключевых слов "class", "constructor", "static", "extends" и "super" при описании класса.

8. Рекомендуемая литература

1. **JAVASCRIPT.RU** [Электронный ресурс] / Современный учебник JavaScript – 2007—2020 Илья Кантор. – Режим доступа: https://learn.javascript.ru. – Дата доступа: 04.03.2020.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. 4-е изд. – СПб.: Питер, 2018.
3. **Симпсон, К.** ES6 и не только / К. Симпсон. – СПб.: Питер, 2017.
4. **Хавербеке, М.** Выразительный JavaScript. Современное веб-программирование / М. Хавербеке – СПб.: Питер, 2019.