

# Методы объектов

Занятие 8

# План занятия

- Методы объектов, `this`
- Преобразование объектов: `toString` и `valueOf`
- Создание объектов через «`new`»
- Дескрипторы, геттеры и сеттеры свойств
- Статические и фабричные методы



# Методы объектов



# Методы у объектов

```
var user = {  
  name: 'Василий',  
  
  // метод  
  sayHi: function() {  
    console.log( 'Привет!' );  
  }  
  
};  
  
// присвоили метод после создания объекта  
user.sayBye = function() {  
  console.log('Пока!');  
};  
  
// Вызов метода:  
user.sayHi();
```

# Доступ к объекту через this

```
var user = {  
  name: 'Василий',  
  
  sayHi: function() {  
    console.log( this.name );  
  }  
};
```

```
user.sayHi();  
// sayHi в контексте user
```

# Доступ к объекту через this

```
var user = {  
  name: 'Василий',  
  
  sayHi: function() {  
    console.log( user.name );  
    // приведёт к ошибке  
  }  
};
```

```
var admin = user;  
user = null;
```

```
admin.sayHi();
```

# Доступ к объекту через this

```
var user = {  
  name: 'Василий',  
  
  sayHi: function() {  
    // передать текущий объект в showName  
    showName(this);  
  }  
};
```

```
function showName(namedObj) {  
  console.log( namedObj.name );  
}
```

```
user.sayHi(); // Василий
```

# this у функции

```
function sayHi() {  
  console.log( this.firstName );  
}
```



# this у функции

```
var user = { firstName: 'Вася' };  
var admin = { firstName: 'Админ' };
```

```
function func() {  
    console.log( this.firstName );  
}
```

```
user.f = func;  
admin.g = func;
```

*// this равен объекту перед точкой:*

```
user.f(); // Вася  
admin.g(); // Админ
```

# Значение `this` без контекста

```
function func() {  
  console.log( this );  
  // выведет [object Window] или [object global]  
}  
  
func();
```

# Задача 8.1

устно

Что выведет этот код?

```
var arr = ['a', 'b'];
```

```
arr.push(function() {  
  console.log( this );  
});
```

```
arr[2]();
```

# Задача 8.1

решение

```
// "a","b",function
```

## Задача 8.2

Создайте объект `calculator` с тремя методами:

1. `read()` запрашивает `prompt` два значения и сохраняет их как свойства объекта
2. `sum()` возвращает сумму этих двух значений
3. `mul()` возвращает произведение этих двух значений



# Задача 8.2

## решение

```
var calculator = {  
  sum: function() {  
    return this.a + this.b;  
  },  
  
  mul: function() {  
    return this.a * this.b;  
  },  
  
  read: function() {  
    this.a = +prompt('Enter A?', 0);  
    this.b = +prompt('Enter B?', 0);  
  }  
};  
  
calculator.read();  
console.log( calculator.sum() );  
console.log( calculator.mul() );
```





# Преобразование объектов: `toString` и `valueOf`



# Логическое преобразование

```
if ({} && []) {  
    console.log( 'Все объекты - true!' );  
    // console.log работает  
}
```

# Строковое преобразование

```
var user = {  
  firstName: 'Василий'  
};  
  
alert( user ); // [object Object]
```

# Строковое преобразование

```
var user = {  
    firstName: 'Василий',  
    toString: function() {  
        return 'Пользователь ' + this.firstName;  
    }  
};  
  
alert( user ); // Пользователь Василий
```



# Строковое преобразование

```
alert( [1, 2] );
```

*// toString для массивов выводит список элементов "1,2"*

```
alert( new Date );
```

*// toString для дат выводит дату в виде строки*

```
alert( function() {} );
```

*// toString для функции выводит её код*

# Численное преобразование

```
var room = {  
  number: 777,  
  
  valueOf: function() { return this.number; },  
  toString: function() { return this.number; }  
};
```

`alert( +room );` // 777, вызвался *valueOf*

`delete room.valueOf;` // *valueOf* удалён

`alert( +room );` // 777, вызвался *toString*

# Задача 8.3

устно

Какими будут результаты alert?

```
var foo = {  
  toString: function() {  
    return 'foo';  
  },  
  valueOf: function() {  
    return 2;  
  }  
};
```

```
alert( foo );  
alert( foo + 1 );  
alert( foo + '3' );
```

# Задача 8.4

устно

Какими будут результаты у  
выражений?

1. **new** Date(0) - 0
2. **new** Array(1)[0] + ""
3. ({})(0)
4. [1] + 1
5. [1,2] + [3,4]
6. [] + **null** + 1
7. [[0]][0][0]
8. ({} + {})



# Создание объектов через «new»



# Конструктор

```
function Animal(name) {  
  this.name = name;  
  this.canWalk = true;  
}
```

```
var animal = new Animal('Elephant');
```

# Конструктор

Функция, запущенная через `new`, делает следующее:

1. Создаётся новый пустой объект.
2. Ключевое слово `this` получает ссылку на этот объект.
3. Функция выполняется. Как правило, она модифицирует `this`, добавляет методы, свойства.
4. Возвращается `this`.

# Конструктор

```
animal = {  
  name: 'Elephant',  
  canWalk: true  
}
```

# Конструктор

```
function Animal(name) {  
    // this = {};  
  
    // в this пишем свойства, методы  
    this.name = name;  
    this.canWalk = true;  
  
    // return this;  
}  
  
var animal = new Animal('Elephant');
```

# Правила обработки return

- При вызове return с объектом, будет возвращён он, а не this.
- При вызове return с примитивным значением, оно будет отброшено.



# Правила обработки return

```
function BigAnimal() {  
    this.name = 'Мышь';  
  
    return { name: 'Годзилла' }; // object  
}  
  
console.log( new BigAnimal().name );  
// Годзилла, получили объект вместо this
```

# Правила обработки return

```
function BigAnimal() {  
    this.name = 'Мышь';  
    return 'Годзилла'; // string  
}  
  
console.log( new BigAnimal().name );  
// Мышь, получили this (а Годзилла пропал)
```

# Создание методов в конструкторе

```
function User(name) {  
  this.name = name;  
  
  this.sayHi = function() {  
    console.log( 'Моё имя: ' + this.name );  
  };  
}
```

```
var ivan = new User('Иван');
```

```
ivan.sayHi(); // Моё имя: Иван
```

# Локальные переменные

```
function User(firstName, lastName) {  
  // вспомогательная переменная  
  var phrase = 'Привет';  
  
  // вспомогательная вложенная функция  
  function getFullName() {  
    return firstName + ' ' + lastName;  
  }  
  
  this.sayHi = function() {  
    // использование  
    console.log( phrase + ', ' + getFullName() );  
  };  
}  
  
var vasya = new User('Вася', 'Петров');  
vasya.sayHi(); // Привет, Вася Петров
```





# Дескрипторы, геттеры и сеттеры свойств



# Дескрипторы

**Object.defineProperty(obj, prop, descriptor)**



# Дескрипторы

- **value** — значение свойства, по умолчанию undefined
- **writable** — значение свойства можно менять, если true.  
По умолчанию false.
- **configurable** — если true, то свойство можно удалять, а также менять его в дальнейшем при помощи новых вызовов `defineProperty`. По умолчанию false.
- **enumerable** — если true, то свойство будет участвовать в переборе `for..in`. По умолчанию false.
- **get** — функция, которая возвращает значение свойства.  
По умолчанию undefined.
- **set** — функция, которая записывает значение свойства.  
По умолчанию undefined.

# Обычное свойство

```
var user = {};
```

*// 1. простое присваивание*

```
user.name = 'Вася';
```

*// 2. указание значения через дескриптор*

```
Object.defineProperty(user, 'name', { value: 'Вася' });
```

# СВОЙСТВО-КОНСТАНТА

```
'use strict';
```

```
var user = {};
```

```
Object.defineProperty(user, 'name', {  
  value: 'Вася',  
  writable: false, // запретить присвоение 'user.name='  
  configurable: false // запретить удаление 'delete user.name'  
});
```

*// Теперь попытаемся изменить это свойство.*

*// в strict mode присвоение 'user.name=' вызовет ошибку*

```
user.name = 'Петя';
```

# Свойство, скрытое для for..in

```
var user = {  
  name: 'Вася',  
  toString: function() { return this.name; }  
};
```

```
for(var key in user) console.log(key);  
// name, toString
```

# Свойство, скрытое для for..in

```
var user = {  
  name: 'Вася',  
  toString: function() { return this.name; }  
};
```

```
// помечаем toString как не подлежащий перебору в for..in  
Object.defineProperty(user, 'toString', {enumerable: false});
```

```
for(var key in user) alert(key); // name
```

# СВОЙСТВО-ФУНКЦИЯ

```
var user = {  
  firstName: 'Вася',  
  surname: 'Петров'  
};
```

```
Object.defineProperty(user, 'fullName', {  
  get: function() {  
    return this.firstName + ' ' + this.surname;  
  }  
});
```

```
console.log(user.fullName); // Вася Петров
```



# СВОЙСТВО-ФУНКЦИЯ

```
var user = {  
  firstName: 'Вася',  
  surname: 'Петров'  
};
```

```
Object.defineProperty(user, 'fullName', {  
  
  get: function() {  
    return this.firstName + ' ' + this.surname;  
  },  
  
  set: function(value) {  
    var split = value.split(' ');  
    this.firstName = split[0];  
    this.surname = split[1];  
  }  
});
```

```
user.fullName = 'Петя Иванов';  
console.log( user.firstName ); // Петя  
console.log( user.surname ); // Иванов
```

# get/set в литералах

```
var user = {  
  firstName: 'Вася',  
  surname: 'Петров',  
  
  get fullName() {  
    return this.firstName + ' ' + this.surname;  
  },  
  
  set fullName(value) {  
    var split = value.split(' ');  
    this.firstName = split[0];  
    this.surname = split[1];  
  }  
};
```

```
console.log( user.fullName );  
// Вася Петров (из геттера)
```

```
user.fullName = 'Петя Иванов';  
console.log( user.firstName );  
// Петя (поставил сеттер)  
console.log( user.surname );  
// Иванов (поставил сеттер)
```



# Статические и фабричные методы

# Статические свойства

```
function Article() {  
    Article.count++;  
}
```

*// статическое свойство-переменная*  
Article.count = 0;

*// статическое свойство-константа*  
Article.DEFAULT\_FORMAT = "html";

# Статические методы

```
function Article() {  
    Article.count++;  
  
    //...  
}  
Article.count = 0;  
  
Article.showCount = function() {  
    console.log( this.count ); // (1)  
};  
  
// использование  
new Article();  
new Article();  
Article.showCount(); // (2)
```

# Сравнение объектов

```
function Journal(date) {  
    this.date = date;  
    // ...  
}
```

```
// возвращает значение, большее 0,  
// если A больше B, иначе меньшее 0  
Journal.compare = function(journalA, journalB) {  
    return journalA.date - journalB.date;  
};
```

[bit.ly/1LtPg5t](http://bit.ly/1LtPg5t)



# Статические методы

```
function Journal() { /*...*/ }
```

```
Journal.formatDate = function(date) {  
    return date.getDate() + '.' + (date.getMonth()+1) + '.' +  
    date.getFullYear();  
};
```

```
// ни одного объекта Journal нет, просто форматируем дату  
alert( Journal.formatDate(new Date) );
```



# Фабричные методы

- `new Date()`
- `new Date(milliseconds)`
- `new Date(year, month, day ...)`
- `new Date(datestring)`

## Полиморфная функция конструктор

```
function User(userData) {  
  if (userData) {  
    this.name = userData.name;  
    this.age = userData.age;  
  } else {  
    this.name = 'Аноним';  
  }  
  
  this.sayHi = function() {  
    console.log(this.name)  
  };  
}
```

*// Использование*

```
var guest = new User();  
guest.sayHi(); // Аноним
```

```
var knownUser = new User({  
  name: 'Вася',  
  age: 25  
});  
knownUser.sayHi(); // Вася
```

## Фабричные методы

```
function User() {  
  this.sayHi = function() {  
    console.log(this.name)  
  };  
}  
  
User.createAnonymous = function() {  
  var user = new User;  
  user.name = 'Аноним';  
  return user;  
};  
  
User.createFromData = function(userData) {  
  var user = new User;  
  user.name = userData.name;  
  user.age = userData.age;  
  return user;  
};  
  
// Использование  
var guest = User.createAnonymous();  
guest.sayHi(); // Аноним  
  
var knownUser = User.createFromData({  
  name: 'Вася',  
  age: 25  
});  
knownUser.sayHi(); // Вася
```

# Преимущества фабричных методов

- Лучшая читаемость кода.
- Лучший контроль ошибок.
- Удобная расширяемость.

# План занятия

- Методы объектов, `this`
- Преобразование объектов: `toString` и `valueOf`
- Создание объектов через «`new`»
- Дескрипторы, геттеры и сеттеры свойств
- Статические и фабричные методы



**2015**