SoftServe

Empowering your Business
through Software Development

Module 4:
OOP in JavaScript

D. Petin

06/2014

Agenda

- Custom objects
- Constructors
- Context and "***this***"
- Operator "***new***"

Custom Object

Object creation

You know that we can create a simple object in JavaScript. We use JSON for this.

```
var cat = {
    name: "Znizhok",
    color: "white"
};
```

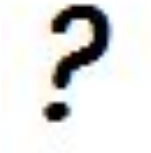
[1]

Object or Hash Table

But this way it looks like hash table creation. What is the difference between hash table and object, then?

```
var hash = {
    key: value,
    key: value
};
```

[1]



```
var object = {
    key: value,
    key: value
};
```

Object or Hash Table

Typically we use hash table if we want to represent some collection, and we use an object to describe some system or entity.

```
var cats = {
    first: murzyk,
    second: barsyk
};
```

[1]

```
var cat = {
    name: barsik,
    color: white
};
```

Difference in use

There are some differences in using of hash tables and objects as a result. For example:

```
cats["first"];   // good way
```

[1]

To access elements of hash table we use indexer **[ ]** with key inside. But it's incorrect for objects! For objects Operator "**.**" should be used :

```
cat["name"];   // incorrect!
cat.name;      // good way
```

[2]

Constructors

Constructors

Sometimes we need to create more than one single object. It is not a good idea to use the literal way for this. It will be better create a *scenario* for objects reproducing.

Constructor is a function that implements this scenario in JavaScript.

Constructor consists of declaration attributes and methods that should be added into each new object with presented structure.

Constructors: example

```
function Cat (name) {
    this.name = name;
    this.run = function () {
        console.log(this.name + " run!");
    };
    return this;
}
```

[1]

```
var murzyk = new Cat("Murzyk");
```

[2]

Context and "this"

Context

Let's imagine two identical objects.

They are created by **Cat** constructor:

```
var murzyk = new Cat("Murzyk"),
    barsyk = new Cat("Barsyk");
```

[1]

Context

If we call method **run()** for both cats, we'll take correct results:

murzyk.run();

In console:

Murzyk run!

[1]

barzyk.run();

In console:

Barsyk run!

How does the interpreter distinguish whose name should be printed?

Context

It works because we use the next form of access to attribute name: *this.name*.

this contains inside a reference to object on whose behalf was called method *run*.

Such a reference is called a **context**.

The context determined automatically after the method calling and can't be changed by code.

Loss of context

Be careful! There are situations when you can lose a context. For example:

```
setTimeout(murzyk.run, delay);
```

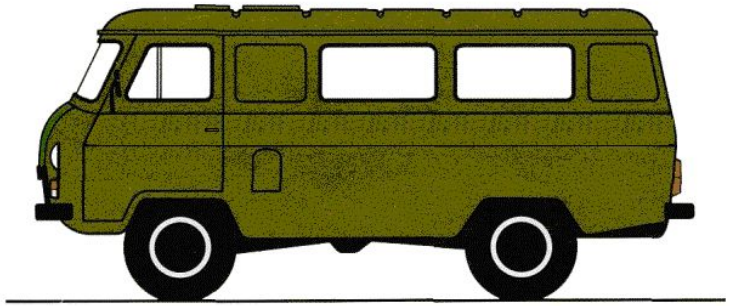
In console: [1]
undefined run!

murzyk.run is a reference to method. And **only reference** was saved in setTimeout. When the method was called by saved reference, object *window* will be used as a context and *this.name* (equal to *window.name*) was not found.

Operator new

Pre-example

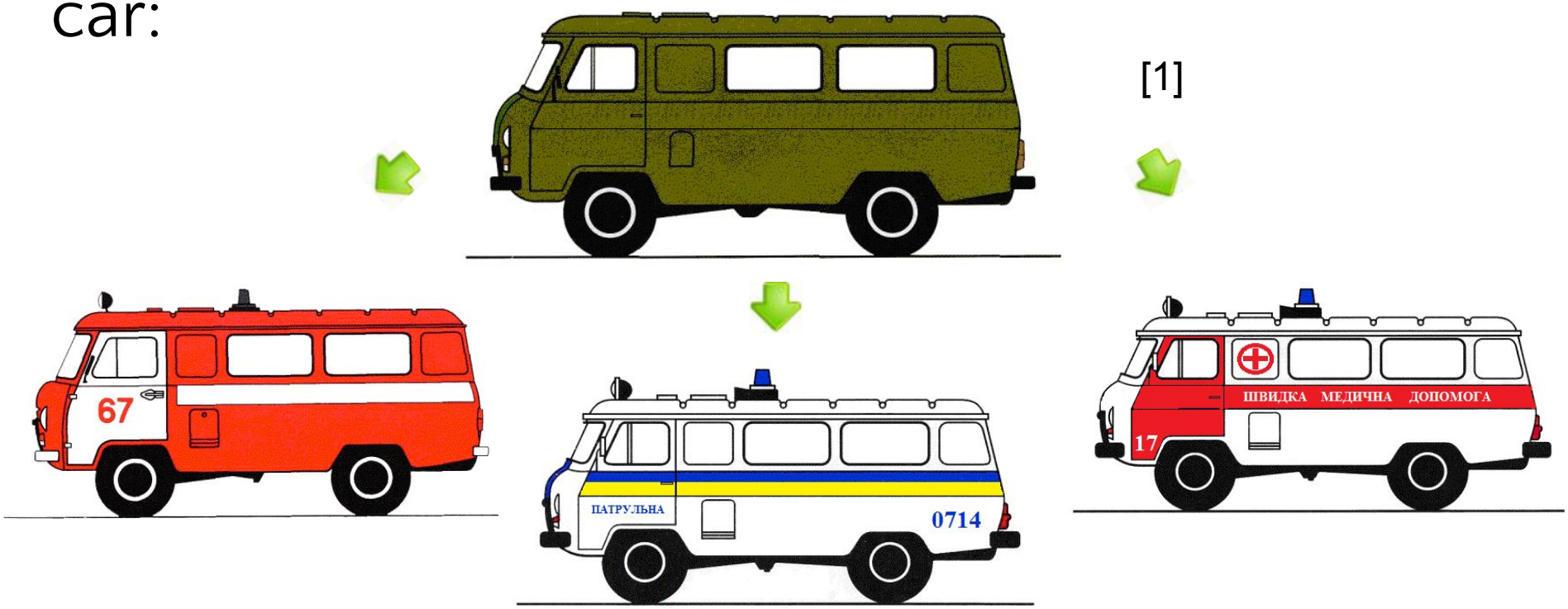
Imagine that some abstract factory produces cars. All cars are absolutely identical:



[1]

Pre-example

But there are some emergency services and each of them has an own color scheme for a car:

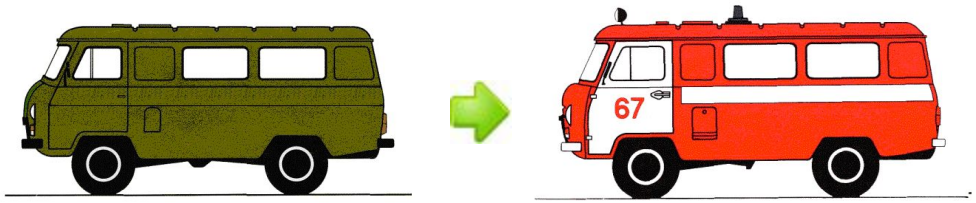


[1]

New: scenario of work

new processing has a similar scenario:

- creation of default object [1]
- calling of constructor with just created object context [2]
- modification of default object [3]
- returning and saving the reference to modified object [4]



[5]

New: example

```
var murzyk = new Cat("Murzyk");
```

[1]

- creation of default object [2]

```
var _temporary_ref = new Object();
```

[3]

Interpreter creates some variables for temporary storing of reference to new object. Now it's a default object.

New: example

```
var murzyk = new Cat("Murzyk");
```

[1]

- calling of constructor with just created object context [2]

```
_temporary_ref.Cat();
```

[3]

_temporary_ref set as a context for constructor Cat.

this inside the Cat refers to as yet default object.

New: example

```
var murzyk = new Cat("Murzyk");
```

[1]

- modification of default object

[2]

```
this.name = "Murzyk";
this.run = function () { . . . };
```

[3]

Interpreter extends the default object inside the constructor. If a key is not found, it will be created, as it occurs with hashes and arrays.

New: example

```
var murzyk = new Cat("Murzyk");
```

[1]

returning and saving the reference to modified object

[2]

```
var murzik = _temporary_ref;
```

[3]

At last the reference to modified object returned and saved in a user variable.


THANK
YOU