ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Paradigms in **JavaScript**

Alex Thalhammer

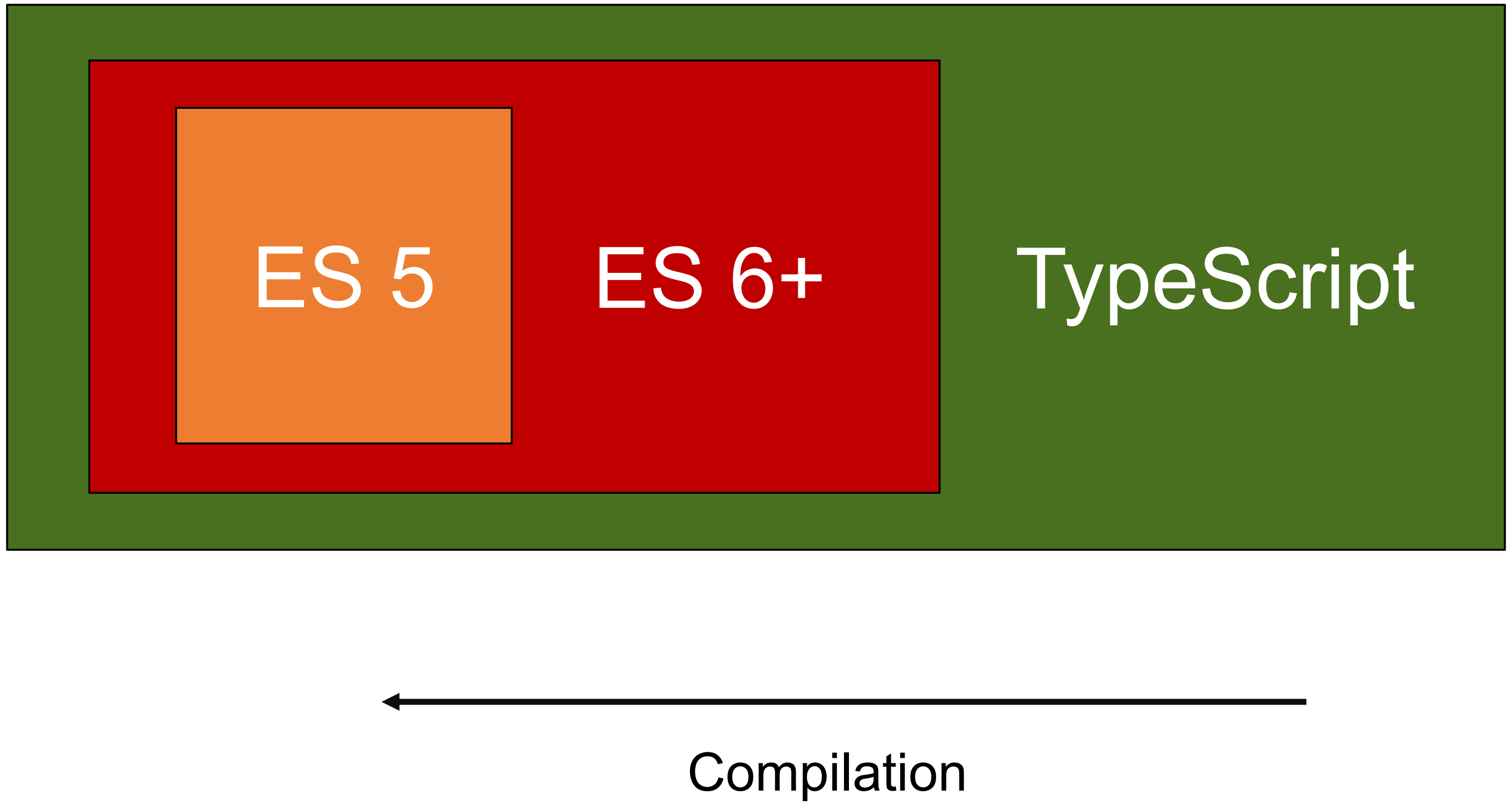
Outline

- Overview
    - Procedural Paradigm
    - Functional Paradigm
    - Object oriented Paradigm
    - DEMO
- More details
    - Functions and this
    - Data types in JS
    - Exceptions
    - Prototypes
    - Spread operator
    - Modular Paradigm
    - DEMO

Overview

ES 5 < ES 6 < TypeScript

ES 6+: official:
ES 2015, 2016, 2017



Procedural Paradigm

The procedual paradigm

```
function calcInterest(k, p, t) {
    var result = k * p * t / 36000;
    return result;
}

var result = calcInterest(200, 2, 360);
alert("Result: " + result);
```

Selected predefined procedures

```
var two = parseInt("2");

var twoPointTwo = parseFloat("2.2");

var isSevenNaN = isNaN("seven");
```

Functional Paradigm

The functional paradigm I

```
function forEach(ary, action) {
    for (var i = 0; i < ary.length; i++) {
        action(ary[i]);
    }
}
```

The functional paradigm II

```
function forEach(ary, action) {
    for (var i = 0; i < ary.length; i++) {
        action(ary[i]);
    }
}

function showItem(item) { alert(item); }
var myInts = [1, 2, 3, 4];
forEach(myInts, showItem);
```

The functional paradigm III

```
function forEach(ary, action) {
    for (var i = 0; i < ary.length; i++) {
        action(ary[i]);
    }
}

function showItem(item) { alert(item); }
var myInts = [1, 2, 3, 4];
forEach(myInts, showItem);

forEach(myInts, function (item) {
    alert(item);
});
```

Lambda statements with arroy syntax (ES 6)

```
forEach(myInts, (item) => {
    alert(item);
});
```

Arrow syntax (since ES 6)

```
forEach(myInts, item => {   // alternatively for 1 param w/o parenthesis
    alert(item);
});

forEach(myInts, item => alert(item) ); // just 1 row => return statement
```

DEMO

functions

Object oriented Paradigm

The object oriented paradigm

```
let flightBooking = {
    from: "Graz",
    to: "Mallorca",
    passengers: [
        {
            firstname: "Max", lastname: "Muster"
        },
        {
            firstname : "Susi", lastname: "Schuster"
        }
    ],
    payment: {
        type: "creditCard", amount: 250, paid: true
    }
};
```

Object literals

Constructor functions (ES 5, still working)

```
function Person(id, firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;

    this.fullName = function () {
        return this.id + ": " + this.firstname + " " + this.lastname;
    }
}


var rudi = new Person(47, "Rudolf", "Rentier");
alert(rudi.firstname);
alert(rudi.lastname);
alert(rudi.fullName());
```

Classes since ES6

```
class Person {
      id;
      firstname;
      lastname;

      constructor(id, firstname, lastname) {
         this.id = id;
         this.firstname = firstname;
         this.lastname = lastname;
      }

      fullName() {
         return this.id + ": " +this.firstname + " " + this.lastname;
      }
}
```

DEMO

class

More details

Functions and this

This

- *this* in function refers to current "context"
- Caller sets context

Context

- obj.method()
  - this => obj
- func.call(x, y, z)
  - this => x, parameter: y, z
- new Func() (constructor)
  - this => new „empty“ Object
- outside function
  - this => global object („*window*“ in browser)

Thought experiment

- What does this refer to in doStuff?
- obj.doStuff();
- var m = obj.doStuff;

Function

- Every function is represented by a function object
- Methods:
  - func.call(thisArg, arg1, arg2, …)
  - func.apply(thisArg, aryArray)
  - func2 = func.bind(thisArg)

Lambda statements bind this

```
forEach(myInts, function (item) {
    console.debug(this);   // Caller (= forEach sets this)
});

var that = this;
forEach(myInts, function (item) {
    console.debug(that);
});

forEach(myInts, (item) => {
    console.debug(this);
});
```

DEMO

this

Data types

Overview data types in JS

- number
  - var num = 3.14;
  - var i = 0;
- boolean
  - var ok = true;
- string
  - var name = 'Max';
  - var multiline = `
        Hallo ${name}!
    `; // ES6
- array
  - var ary = [1, 2, 3];
- object
  - var obj = { x: 1, y: 2 };
- function
  - var f = function () { … }
- null
  - var maybe = null;
  - "attribute has no value"
- undefined
  - var maybe;
  - "attribute doesn't exist / wasn't set"

typeof

- Returns data type as string
- if (typeof value === "undefined") { … }

Comparisons

- == and != perform type conversions
  - "1" == 1 // true
- === and !== also require equality in types
  - "1" === 1 // false

- Always prefer the second with three "="!

Booleans

- Falsy
  - false, null, undefined, 0, "", NaN
- Truthy
  - !falsy

- var emptyObject = {};
- var emptyArray = [];
- if (emptyObject && emptyArray) // true

Objects are dictionaries

- rudi.name === rudi['name']
- First one can be optimized better (also better for static typing in TS)

- But
- var key = 'name'
  rudi[key] === rudi.name

Iterate keys of an object

-
  ```
  for (let key of Object.keys(rudi)) {      // of: ES6
      console.debug(key, rudi[key]);
  }
  ```

-
  ```
  for (let key in rudi) {
      console.debug(key, rudi[key]);
  }
  ```

Declarations

- var x;
  - Scope: Whole function, valid from beginning of the function (hoisting)
- let y;
  - Scope: Current block, valid from declaration (like other languages)
- const z = 3.14;
  - Like let but constant (readonly)

Details: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects

Globals Objects (Excerpt)

Number	Boolean	String
Date	Array	RexExp

Exceptions

Exceptions

```
// structure

try {
   …
}
catch (e) {
   …
}
finally {
   …
}
```

```
// throw

throw 17;
throw "error!";
throw new Error("Error");
```

Error also serves as a base class for your own exception types

Exceptions are not part of method signatures!

DEMO

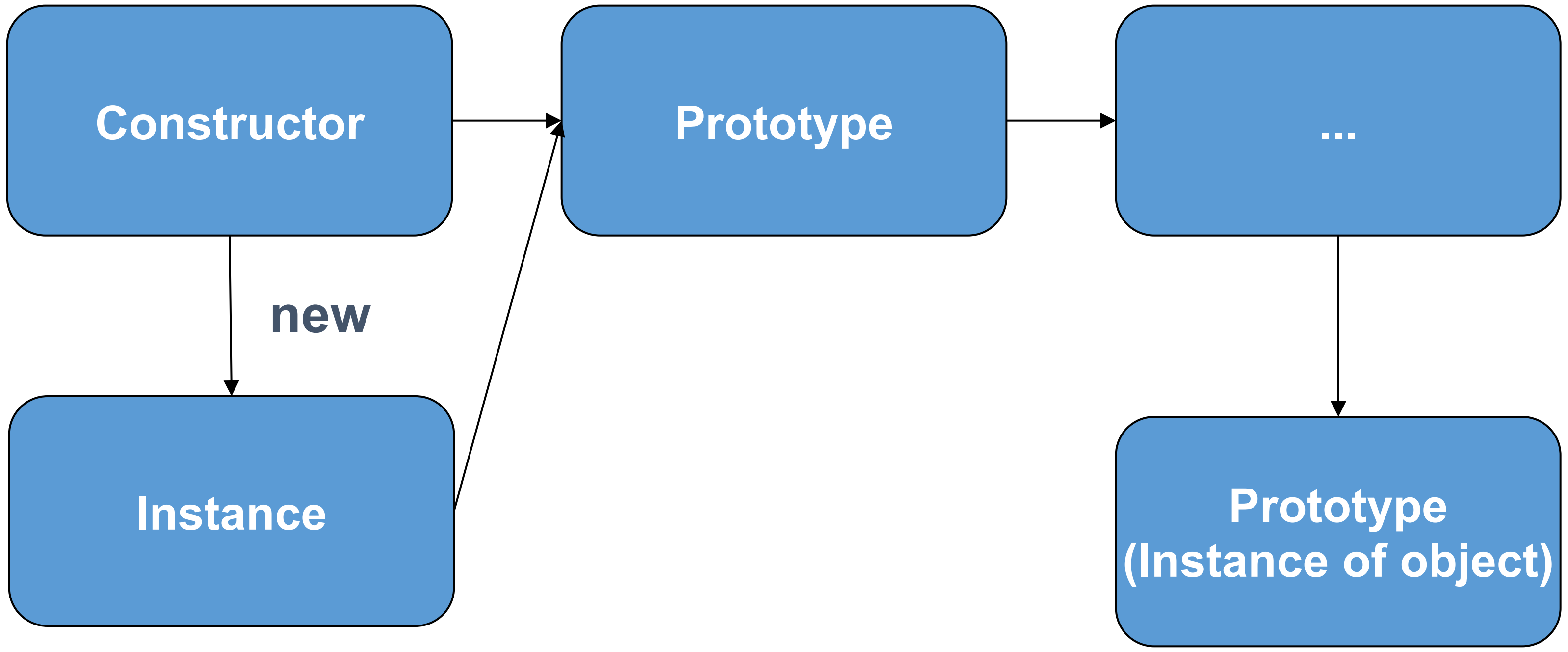
Exception for invalid parameters

Prototypes

Prototypes

- Every object has a Prototype
- Properties (Methoden), not found in the object, JavaScript searches in the prototype

Prototypen


Constructor
Prototype
...
new
Instance
Prototype
(Instance of object)

Example without prototype

```
function Person(id, firstname, lastname) {
    this.id = id;
    this.firstname = firstname;
    this.lastname = lastname;

    this.fullName = function() {
        return this.firstname + " " + this.lastname;
    }
}
```

Example of prototypes

```
function Person(id, firstname, lastname) {
    this.id = id;
    this.firstname = firstname;
    this.lastname = lastname;
}

Person.prototype.fullName = function () {
   return this.firstname + " " + this.lastname;
}
```

Example of prototypes

```
function Employee(id, firstname, lastname, department) {
    this.department = department;
}
```

Example of prototypes

```
function Employee(id, firstname, lastname, department) {
    this.department = department;
}

Employee.prototype = new Person();
```

Example of prototypes

```
function Employee(id, firstname, lastname, department) {
    Person.call(this, id, firstname, lastname);
    this.department = department;
}

Employee.prototype = new Person();
```

Example of prototypes

```
function Employee(id, firstname, lastname, department) {
    Person.call(this, id, firstname, lastname);
    this.department = department;
}

Employee.prototype = new Person();

Employee.prototype.switch = function(newDepartment) {
    console.debug(this.fullName() + " switches to " + newDepartment);

    this.department = newDepartment;
}
```

Example of prototypes

```
var dn = new Employee(1, "Max", "Muster", "Management");
console.debug('Employee', dn);
dn.switch("Dev");
console.debug(After switch', dn);
```

DEMO

Subclass

Spread operator

Examples of spreading

```
const dn2 = { ...dn, firstname: 'Maria' };

const myIntegersExtended = [ ...myIntegers, 4 ];
```

DEMO

spreading

Modules

The modular paradigm

```
(function () {
    var info = "Hello world";
    function sum(a, b) { return a + b; }
    function alertInfo() { alert(info); }
})();
```

IIFE: Immediately-invoked function expression

The modular paradigm

```
var tools = tools || {};  // <-- "empty" object

(function (root) {
    var info = "Hello world";
    root.sum = function(a, b) { return a + b; }
    root.sayHello = function() { alert(info); }
})(tools);

var sum = tools.sum(1,2);
alert(sum);
tools.sayHello();

var sumFunc = tools.sum; // import tools.sum;
sum = sumFunc(1,3);
```

EcmaScript 6 module system

- Since EcmaScript 6
- Every (.js) file is a module
- Files can *export* content for other files
- Those other files can *import* this contents

export and import

```
// a.js
function calcPriceInternal(flightId, discount) { … }
export function calcPrice(flightId) { … }


// b.js
import { calcPrice } from './a';
calcPrice(17);
```