# Прототипы

# Объекты разных типов

```
const student = {
    name: 'Billy',
    getName() {
        return this.name;
    },
    sleep() {}
};
```

```
const lecturer = {
    name: 'Sergey',
    getName() {
        return this.name;
    },
    talk() {}
};
```

# Объекты разных типов

```
const student = {
    name: 'Billy',
    getName() {
        return this.name;
    },
    sleep() {}
};
```

```
const lecturer = {
    name: 'Sergey',
    getName() {
        return this.name;
    },
    talk() {}
};
```

```
const person = {
    getName() {
        return this.name;
    }
};
```

# Прототип

```
const person = {
    getName() {
        return this.name;
    }
};

const student = { name: 'Billy' };
const lecturer = { name: 'Sergey' };

Object.setPrototypeOf(student, person);
Object.setPrototypeOf(lecturer, person);

student.getName(); // Billy
lecturer.getName(); // Sergey
```

# create

```
const person = {
    getName() {
        return this.name;
    }
};

const student = Object.create(person, {
    name: { value: 'Billy' }
});
```

# super

```
const person = {
    getName() {
        return this.name;
    }
};

const student = {
    name: 'Billy',
    getName() {
        return 'Student ' + super.getName();
    }
};

Object.setPrototypeOf(student, person);
student.getName(); // Student Billy
```

# Объекты одного типа

```
const billy = {
    name: 'Billy',
    getName() {
        return this.name;
    }
};
```

```
const willy = {
    name: 'Willy',
    getName() {
        return this.name;
    }
};
```

# Конструирование объектов

# Фабрика

```
function createStudent(name) {
    return {
        name,
        getName() {
            return this.name;
        }
    };
}

const billy = createStudent('Billy');
```

# Фабрика

```
const studentProto = {
    getName() {
        return this.name;
    }
};

function createStudent(name) {
    const student = { name };

    Object.setPrototypeOf(student, studentProto);

    return student;
}
```

# Фабрика

```
const studentProto = {
    getName() {}
};

function createStudent(name) {
    return Object.create(studentProto, {
        name: {
            value: name
        }
    });
}
```

# Фабрика

```
const studentProto = {
    getName() {}
};

function createStudent(name) {
    return Object.create(studentProto, {
        name: {
            value: name,
            enumerable: true,
            writable: true
        }
    });
}
```

# Фабрика

```
const studentProto = {
    getName() {}
};

function createStudent(name) {
    const student = Object.create(studentProto);

    student.name = name;

    return student;
}
```

# Фабрика

```
const studentProto = {
    getName() {}
};

function createStudent(name) {
    const student = Object.create(studentProto);

    return Object.assign(student, { name });
}
```

# Фабрика

```
const personProto = {
    getName() {}
};

const studentProto = Object.create(personProto);

function createStudent(name) {
    const student = Object.create(studentProto);

    return Object.assign(student, { name });
}
```

# Фабрика

```
const personProto = {
    getName() {}
};

const studentProto = Object.create(personProto);

studentProto.getName = function () {
    return 'Student ' + this.name;
}

function createStudent(name) {
    const student = Object.create(studentProto);

    return Object.assign(student, { name });
}
```

# Фабрика

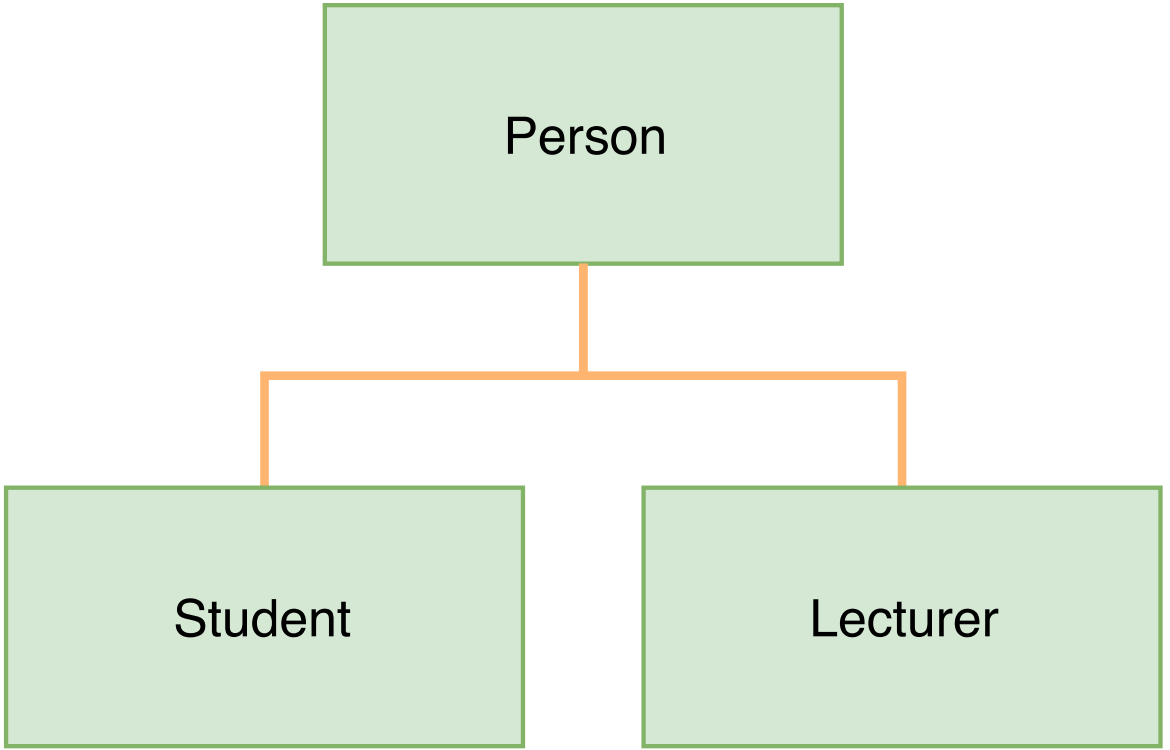
```
const personProto = {
    getName() {}
};

const studentProto = Object.create(personProto);

studentProto.getName = function () {
    return 'Student ' + super.getName(); Error
}

function createStudent(name) {
    const student = Object.create(studentProto);

    return Object.assign(student, { name });
}
```

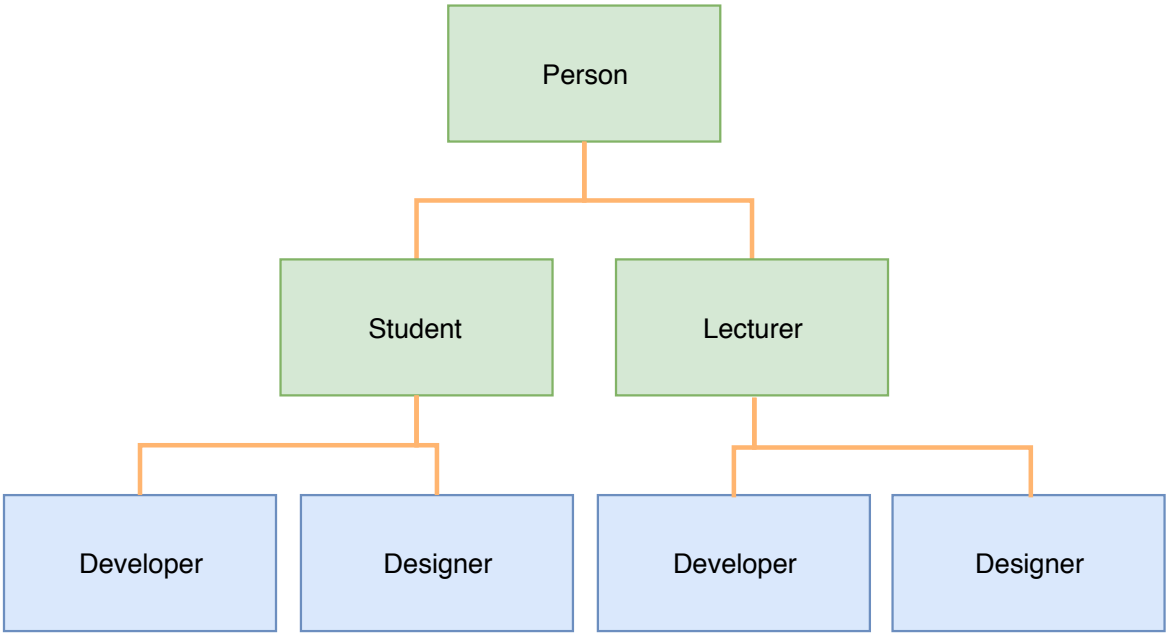
# Фабрика

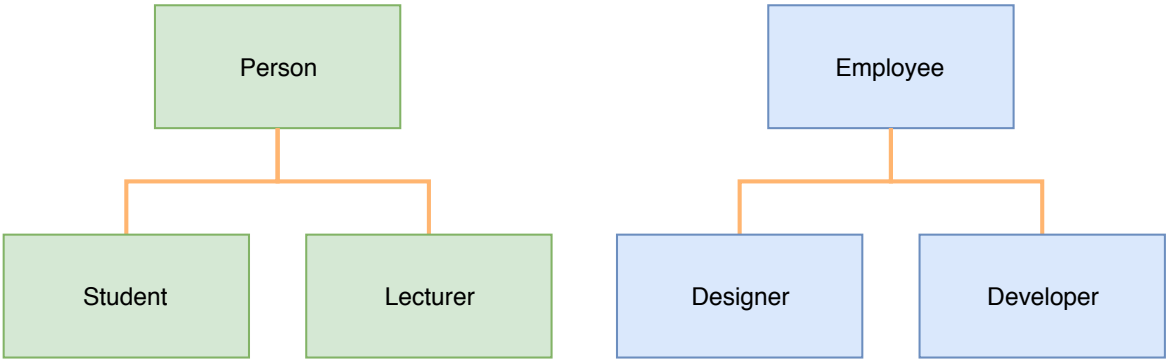
```
const personProto = {
    getName() {}
};

const studentProto = {
    getName() {
        return 'Student ' + super.getName();
    }
};

Object.setPrototypeOf(studentProto, personProto);

function createStudent(name) {
    // ...
}
```

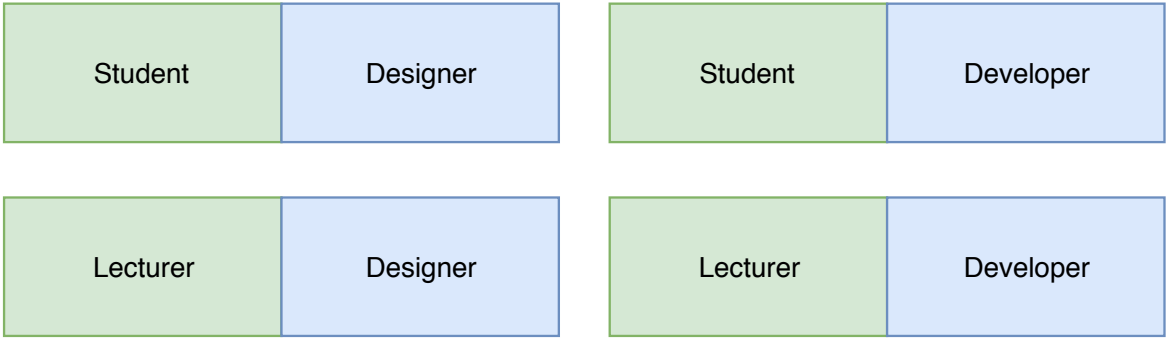
# Фабрика

```
const { createServer } = require('http');

const server = createServer((req, res) => {
    res.end('Hello, World!');
})

server.listen(8080);
```


Person
Student
Lecturer


Person
Student
Lecturer
Developer
Designer
Developer
Designer


Person
Student
Lecturer
Employee
Designer
Developer


Student
Designer
Student
Developer
Lecturer
Designer
Lecturer
Developer

Favor object composition over class inheritance

# Миксины

```
const studentProto = {
    getName() {}
};

const developerProto = {
    getSalary() {}
};

const proto = Object.assign({}, studentProto, developerProto);

function create({ name, salary }) {
    const instance = Object.create(proto);

    return Object.assign(instance, { name, salary });
}
```

Object.assign не переносит свойства полей, неперечисляемые поля, а также set/get

# Копирование полей со свойствами

```
const objTo = {};
const properties = Object.getOwnPropertyNames(objFrom);

for (const property of properties) {
    Object.defineProperty(
        objTo,
        property,
        Object.getOwnPropertyDescriptor(objFrom, property);
    );
}
```

# Конструкторы

```
function createStudent(name) {
    this.name = name;
}

const billy = new createStudent('Billy');
```

Функция, вызванная оператором new,
работает как конструктор объектов

this внутри конструкторов ссылается на создаваемый объект

# Конструкторы

```
function createStudent(name) {
    // const this = {};
    this.name = name;
    // return this;
}

const billy = new createStudent('Billy');
```

# Конструкторы

```
function Student(name) {
    // const this = {};
    this.name = name;
    // return this;
}

const billy = new Student('Billy');
```

Чтобы отличить функции-конструкторы от простых функций принято именовать их с заглавной буквы

# Конструкторы

```
function Student(name) {
    this.name = name;
}

const billy = Student('Billy');
```

# Конструкторы

```
'use strict';

function Student(name) {
    this.name = name;
}

const billy = Student('Billy');
```

# Конструкторы

```
function Student(name) {
    this.name = name;

    return {
        name: 'Willy'
    };
}

const billy = new Student('Billy');

console.info(billy.name); // Willy
```

# Конструкторы

```
function Student(name) {
    this.name = name;

    return 1703;
}

const billy = new Student('Billy');

console.info(billy.name); // Billy
```

# .prototype

```
function Student(name) {
    this.name = name;
}

Student.prototype = {
    getName() {}
}

const billy = new Student('Billy');

billy.getName(); // Billy
```

# .prototype

```
function Student(name) {
    // const this = {};
    // Object.setPrototypeOf(this, Student.prototype);
    this.name = name;
    // return this;
}

Student.prototype = {
    getName() {}
}

const billy = new Student('Billy');

billy.getName(); // Billy
```

# .constructor

```
function Student(name) {
    this.name = name;
}

Student.prototype.constructor === Student; // true
```

# .prototype

```
function Student(name) {
    // const this = {};
    // Object.setPrototypeOf(this, Student.prototype);
    this.name = name;
    // return this;
}

Object.assign(Student.prototype, {
    getName() {}
});

const billy = new Student('Billy');

billy.getName(); // Billy
```

# .prototype

```
function Student(name) {
    // const this = {};
    // Object.setPrototypeOf(this, Student.prototype);
    this.name = name;
    // return this;
}

Student.prototype = {
    constructor: Student,
    getName() {}
};

const billy = new Student('Billy');

billy.getName(); // Billy
```

# Object.prototype

```
// const obj = {};

function Object() {}

Object.prototype = {
    constructor: Object,
    hasOwnProperty() {},
    toString() {}
};

const obj = new Object();
```

# Конструкторы

```
function Person() {}

Object.assign(Person.prototype, {
    getName() {}
});

function Student(name) {
    this.name = name;
}

Object.assign(Student.prototype, {
    getCourse() {}
});

Object.setPrototypeOf(Student.prototype, Person.prototype);
```

# «Классы»

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

    getName() {
        return this.name;
    }
}

typeof Student; // function

Student.prototype.hasOwnProperty('getName'); // true
```

«Классы» нельзя использовать без new

«Классы» не всплывают

# «Классы» не всплывают

```
const student = new Student();

function Student() {}

const lecturer = new Lecturer(); Error

class Lecturer {}
```

ReferenceError: Lecturer is not defined

# get/set

```
class Student {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    get fullName() {
        return this.firstName + ' ' + this.lastName;
    }
}
```

# Статичный метод

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

    static createAdmin(name) {
        return new User('admin', name);
    }

    static createGuest(name) {
        return new User('guest', name);
    }
}
User.createAdmin('Billy');
```

Все методы не перечислимы

Все методы поддерживают super

Все методы работают в строгом режиме

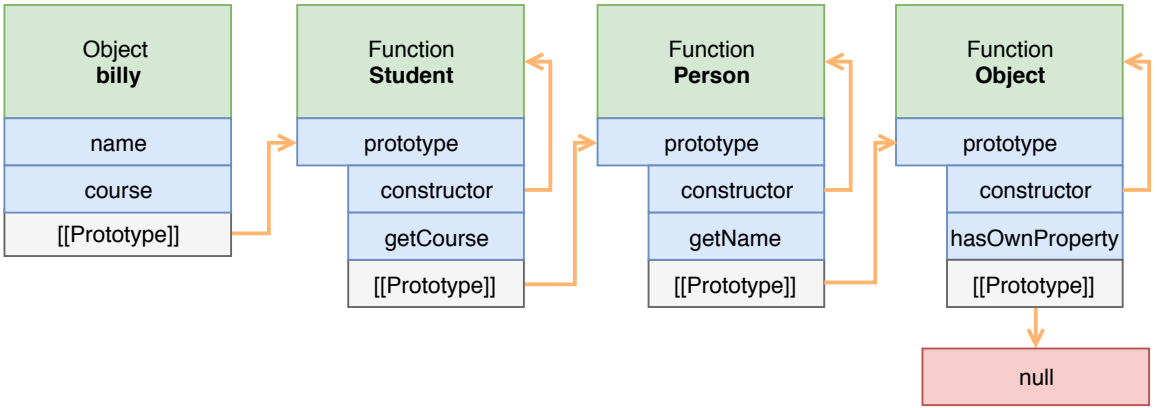
# extends

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

class Student extends Person {
    constructor(name, course) {
        super(name);
        this.course = course;
    }
}

const billy = new Student('Billy', 4);
```

# extends

```
class Student extends Person {
    constructor(name, course) {
        this.course = course;
        super(name);
    }
}

const billy = new Student('Billy', 4); Error
```

ReferenceError: Must call super constructor in derived class before accessing or returning from derived constructor


Object
billy
name
course
[[Prototype]]
Function
Student
prototype
constructor
getCourse
[[Prototype]]
Function
Person
prototype
constructor
getName
[[Prototype]]
Function
Object
prototype
constructor
hasOwnProperty
[[Prototype]]
null

Наследоваться можно либо от другого конструктора («класса»), либо от null

# extends

```
class Student extends null {}
```

# isPrototypeOf

```
class Student {}

const billy = new Student();

Student.prototype.isPrototypeOf(billy); // true
```

# instanceof

```
class Student {}

const billy = new Student();

billy instanceof Student; // true
```

# instanceof

```
class Student extends Person {}

const billy = new Student();

billy instanceof Student; // true
billy instanceof Person; // true
```

# billy instanceof Person

```
billy.[[Prototype]] === Person.prototype; // false

// Может, там null?

billy.[[Prototype]] === null; // false

// Идём дальше по цепочке

billy.[[Prototype]].[[Prototype]] === Person.prototype; // true

// Возвращаем true
```

# Развитие JavaScript

TC39 (Technical Committee 39)

Microsoft, Google, IBM, Intel, Yahoo, Facebook, Airbnb, Netflix, PayPal, Hewlett Packard

Переодически встречаются и
рассматривают предложения (proposals)

Заметки со встреч размещаются на Github, а решения принимаются большинством голосов

## 0. Идея (Strawman)

Идея может быть предложена командой TC39 или любым человеком зарегистрированным как контрибутор

1. Предложение (Proposal)

Выбирается ответственный за предложение из TC39. Подготавливается подробное описание с примерами. Пишутся полифилы.

## 2. Черновик (Draft)

Подготавливается описание на языке спецификации ECMAScript. Пишутся две реализации (одна из них для транспилера).

## 3. Кандидат (Candidate)

Описание финализируется и проходит ревью других членов TC39. Пишутся две полноценные реализации.

## 4. Спецификация (Finished)

Пишутся тесты Test262. Реализации проходят все тесты. Редакторы ECMAScript подписыват спецификацию.

## Стандарт

Раз в год в июле обновлённая спецификация становится новым стандартом

# Class fields + Private methods (Candidate)

```
class Student {
    #name = null;

    constructor(value) {
        this.#name = value;
    }

    getName() {
        return 'Student ' + this.#name;
    }
}
```

# Декораторы методов (Draft)

```
class Student {
    @enumerable(true)
    getName() {
        return this.name;
    }
}

function enumerable(value) {
    return (target, propertyName, descriptors) => {
        // target === Student.prototype
        descriptors.enumerable = value;
    };
}
```

# Декораторы классов (Draft)

```
@debug
class Student {}

function debug(Class) {
    return (...args) => {
        console.trace(Class.name, args);

        return new Class(...args);
    };
}
```

# Babel

```
npm install -g babel-cli
babel-node --plugins transform-decorators script.js
```

# Почитать

Speaking JavaScript
Chapter 17. Objects and Inheritance.
Layer 3: Constructors—Factories for Instances

Speaking JavaScript
Chapter 17. Objects and Inheritance
Layer 4: Inheritance Between Constructors

Exploring ES6
15. Classes

# Почитать

Eric Elliott
Common Misconceptions About Inheritance in JavaScript

Лекции 2015 года
Про this

# Почитать

Современный учебник Javascript
ООП в прототипном стиле

Современный учебник Javascript
Современные возможности ES-2015
Классы

# Почитать

Dr. Axel Rauschmayer
The TC39 process for ECMAScript features