

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  
«Применение объектно-ориентированного подхода в программировании на языке JavaScript»

Минск  
2021

## Лабораторная работа № 27

### Тема работы: Применение объектно-ориентированного подхода в программировании на языке JavaScript

#### 1. Цель работы

Формирование умений применения объектно-ориентированного подхода в программировании на языке JavaScript.

#### 2. Задание

Реализовать ООП в сценарии на языке JavaScript.

#### 3. Оснащение работы

ПК, редактор исходного кода, браузер.

#### 4. Основные теоретические сведения

Объектно-ориентированное программирование (ООП) – это шаблон проектирования программного обеспечения, который позволяет решать задачи с точки зрения объектов и их взаимодействий. JavaScript реализует ООП через прототипное наследование.

Прототипное программирование – это модель ООП, которая не использует классы, а вместо этого сначала выполняет поведение класса и затем использует его повторно (эквивалент наследования в языках на базе классов), декорируя (или расширяя) существующие объекты прототипы.

Прототипное ООП отличается от традиционного ООП тем, что в нем нет никаких классов – только объекты, которые наследуются от других объектов. Каждый объект в JavaScript содержит ссылку на свой родительский (прототип) объект.

В JavaScript существует 4 способа создать объект:

- функция-конструктор (constructor function);
- класс (class);
- связывание объектов (object linking to other object, OLOO);
- фабричная функция (factory function).

Свойства и методы можно определять двумя способами:

- в экземпляре;
- в прототипе.

Конструкторами являются функции, в которых используется ключевое слово «this».

```
function Person(firstName, lastName) {  
    this.firstName = firstName  
    this.lastName = lastName  
};
```

Экземпляры создаются с помощью ключевого слова «new».

```
const anna = new Person('Anna', 'Ivanova');  
console.log(anna.firstName);  
console.log(anna.lastName);  
  
const ivan = new Person('Ivan', 'Ivanov');  
console.log(ivan.firstName);  
console.log(ivan.lastName);
```

Для определения свойства в экземпляре необходимо добавить его в функцию-конструктор.

```
function Person (firstName, lastName) {
  this.firstName = firstName
  this.lastname = lastName
  this.getfullname = function () {
    console.log(`${firstName} ${lastName}`)
  }
};

anna=new Person ('Anna','Ivanova');
Person {firstName: "Anna", lastname: "Ivanova", getfulln
  ame: f}
  firstName: "Anna"
  ▶ getfullname: f ()
  lastname: "Ivanova"
  ▼ __proto__:
    ▶ constructor: f Person(firstName, lastName)
    ▶ __proto__: Object
anna.getfullname();
```

Для добавления свойства в прототип используют prototype.

```
Person.prototype.getname = function () {
  console.log(`I'm ${this.firstName}`)
};

anna=new Person ('Anna','Ivanova');
Person {firstName: "Anna", lastName: "Ivanova"}
  firstName: "Anna"
  lastName: "Ivanova"
  ▼ __proto__:
    ▶ getname: f ()
    ▶ constructor: f Person(firstName, lastName)
    ▶ __proto__: Object
anna.getname();
```

Создание объектов с использованием классов выглядит следующим образом. Сперва описывается функция создания класса.

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

В данном случае constructor можно опустить, если нет необходимости присваивать начальные значения.

Экземпляры также создаются с помощью ключевого слова «new».

```
const anna = new Person('Anna', 'Ivanova');
console.log(anna.firstName);
console.log(anna.lastName);
```

Свойства экземпляра можно определить в constructor.

```
class Person {
  constructor (firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
```

```

    this.getname = function () {
        console.log(`I'm ${firstName}`)
    }
}
};

anna=new Person ('Anna', 'Ivanova');
    < Person {firstName: "Anna", lastName: "Ivanova", getna
      me: f} i
      firstName: "Anna"
      ▶ getname: f ()
      lastName: "Ivanova"
      ▼ __proto__:
        ▶ constructor: class Person
        ▶ __proto__: Object

```

Свойства прототипа определяются после constructor в виде обычной функции.

```

class Person {
    constructor (firstName, lastName) {
        this.firstName = firstName
        this.lastName = lastName
    }

    getfullname () {
        console.log(`${this.firstName} ${this.lastName}`)
    }
}

anna=new Person ('Anna', 'Ivanova');
    < ▼ Person {firstName: "Anna", lastName: "Ivanova"} i
      firstName: "Anna"
      lastName: "Ivanova"
      ▼ __proto__:
        ▶ constructor: class Person
        ▶ getfullname: f getfullname()
        ▶ __proto__: Object

```

При связывании объектов сперва определяется «родительский» объект. Затем с помощью метода (который, как правило, называется init, но это не обязательно, в отличие от constructor в классе) выполняется инициализация экземпляра.

```

const Person = {
    init(firstName, lastName) {
        this.firstName = firstName
        this.lastName = lastName
    }
};

```

Для создания экземпляра используется Object.create. После создания экземпляра инициализация выполняется вызовом init.

```

const anna = Object.create(Person);
anna.init('Anna', 'Ivanova');
console.log(anna.firstName);
console.log(anna.lastName);

```

Определение свойств и методов экземпляра при связывании объектов осуществляется путем добавления свойства/метода к this.

```

const Person = {
    init (firstName, lastName) {
        this.firstName = firstName
        this.lastName = lastName
    }
};

```

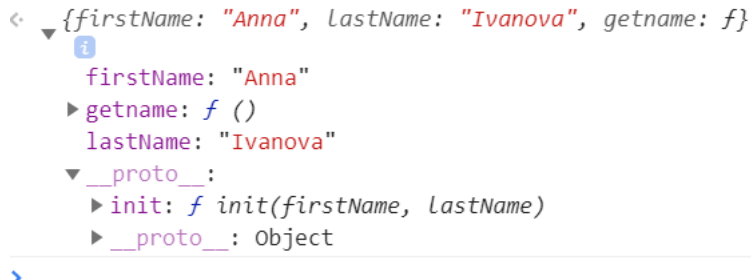
```

    this.getname = function () {
        console.log(` I'm ${firstName}`)
    }

    return this
}
};

```

```
anna=Object.create(Person).init('Anna','Ivanova');
```



```

{firstName: "Anna", lastName: "Ivanova", getname: f}
  |
  +-- firstName: "Anna"
  |
  +-- getname: f ()
  |
  +-- lastName: "Ivanova"
  |
  +-- __proto__:
      |
      +-- init: f init(firstName, lastName)
      |
      +-- __proto__: Object

```

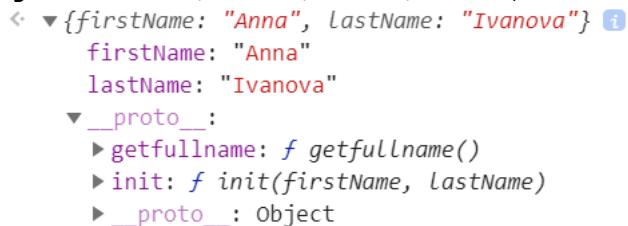
Метод прототипа определяется как обычный объект.

```

const Person = {
    init (firstName, lastName) {      this.firstName = firstName
        this.lastName = lastName
        return this},
    getfullname () {
        console.log(`${this.firstName} ${this.lastName}`)
    }
};

```

```
anna=Object.create(Person).init('Anna','Ivanova');
```



```

{firstName: "Anna", lastName: "Ivanova"}
  |
  +-- firstName: "Anna"
  |
  +-- lastName: "Ivanova"
  |
  +-- __proto__:
      |
      +-- getfullname: f getfullname()
      |
      +-- init: f init(firstName, lastName)
      |
      +-- __proto__: Object

```

Фабричная функция – это функция, возвращающая объект. Можно вернуть любой объект. Можно даже вернуть экземпляр класса или связывания объектов.

```

function Person (firstName, lastName) {
    return {
        firstName,
        lastName
    }
};

```

Для создания экземпляра ключевое слово «new» не требуется. Создание экземпляра осуществляется простым вызовом функции.

```

const anna = Person('Anna', 'Ivanova');
console.log(anna.firstName);
console.log(anna.lastName);

```

При необходимости свойства и методы могут быть включены в состав возвращаемого объекта.

```

function Person (firstName, lastName) {
    return {
        firstName,
        lastName,
        getfullname () {

```

```

        console.log(`${firstName} ${lastName}`)
    }
}
};
anna=Person ('Anna', 'Ivanova');
    {firstName: "Anna", lastName: "Ivanova", getfullname: f}
    i
    firstName: "Anna"
    ▶ getfullname: f getfullname()
    lastName: "Ivanova"
    ▶ __proto__: Object

```

Однако, определять свойства прототипа при использовании фабричных функций нельзя.

При выборе способа определения свойств и методов следует руководствоваться назначением определяемого свойства или метода. Если свойство или метод должны быть уникальными для каждого экземпляра, тогда они должны определяться в экземпляре. Если свойство или метод должны быть одинаковыми (общими) для всех экземпляров, тогда их следует определять в прототипе. В последнем случае при необходимости внесения изменений в свойство или метод достаточно будет внести их в прототип, в отличие от свойств и методов экземпляров, которые корректируются индивидуально.

## 5. Порядок выполнения работы

1. Имеется зоопарк, в котором имеются питомцы разных категорий: млекопитающие, животные, птицы. Которые, в свою очередь, бывают разных типов, например, животные: Львы, Тигры, Белки. Каждый из типов издает свои особенные звуки (львы – рычат, и т.д.) У всех обитателей зоопарка есть имена и возраст. У каждого животного могут быть дети. Реализовать создание экземпляра каждого питомца с использованием ООП.
2. Информация о питомцах должна храниться в массиве. При добавлении нового питомца информация о нем должна добавляться в массив.
3. Подсчитать количество питомцев зоопарка по каждому типу.

## 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. В чем суть прототипного наследования?
2. Перечислите известные Вам способы создания объектов в JavaScript?
3. Приведите пример наследования с использованием классов.

## 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.