МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

**Факультет программной инженерии и компьютерной техники**

**Направление (специальность) — 09.04.02 Информационные системы и технологии Специализация — Веб-технологии**
**Дисциплина — Проектирование и анализ языков веб-решений**

# Курсовой проект (работа)

**ТЕМА: Сравнительный анализ реализации ООП на языках JavaScript и PHP**

**ВЫПОЛНИЛ**
**Студент группы** **41072** _____ **А. Н. Мавлянов**
№ группы подпись, дата ФИО

**ПРОВЕРИЛ** _____ _____ **И. Б. Государев**
ученая степень, должность подпись, дата ФИО

**САНКТ-ПЕТЕРБУРГ**

**2020 г.**

# Оглавление

# Введение

На сегодняшний день количество прикладных языков программирования, реализующих объектно-ориентированный подход, является наибольшим по отношению к другим подходам программирования.

Объектно-ориентированный подход (ООП) – это методология программирования, которая основана на шаблоне проектирования программного обеспечения. Данный подход позволяет думать о проблемах с точки зрения объектов и их взаимодействий. Настоящий подход обычно выполняется с классами или с прототипами. Большинство языков программирования, реализующих данный подход, используют наследование на основе классов. Объектно-ориентированный подход нашел большое применение на таких языках программирования как PHP, JavaScript (ECMAScript 2015), Java, C#. В зависимости от специфики и синтаксиса языка программирования, реализация объектно-ориентированное подхода на разных языках программирования осуществляется про разному. Поскольку в данной работе объектно-ориентированный подход рассматривается с точки зрения двух таких языков программирования как JavaScript и PHP, сравнительный анализ проводился соответственно по этим двум языкам программирования.

В настоящей работе отражён сравнительный анализ объектно-ориентированного подхода на языках JavaScript и PHP. Критериями сравнительного анализа являются синтаксис языков, частота использования объектно-ориентированных конструкций и модель реализации объектно-ориентированного подхода. В результате сравнительного анализа, были выявлены как аналогичные, так и противоположные особенности реализации объектно-ориентированного подхода на рассматриваемых языках программирования.

На настоящем этапе высокие требования предъявляются к внутренней языковой мобильности разработчиков в составе «full stack» команд, что обусловливает **актуальность данной работы**, посвящённой объединяющим и различающим чертам имплементации объектно-ориентированного подхода в JavaScript и PHP.

**Целью курсовой работы** является сравнительный анализ реализации объектно-ориентированного подхода к программированию на языках JavaScript и PHP, позволяющий разработчикам свободнее ориентироваться в кодовой базе при переходе от одной экосистемы к другой.

**Объектом курсовой работы** являются объектно-ориентированный подход в проектировании веб-приложений. **Предметом курсовой работы** являются результаты сравнительного анализа объектно-ориентированного подхода к программированию на рассматриваемых языках программирования.

В задачи курсового проекта входило:

- описание концепции объектно-ориентированного подхода к программированию, его преимуществ и недостатков на языке PHP;
- описание ключевых особенностей ООП на языке JavaScript;
- выявление критерий для сравнительного анализа ООП на рассматриваемых языках программирования;
- сравнительный анализ настоящего подхода на основе выявленных критерии;
- обзор результатов сравнительного анализа.

**Теоретической основой** проводимого исследования выступили работы российских (М.С. Фленов, В.В. Мухортов, С. В. Зыков, и др.) и зарубежных (S. Sanders, D.M. Brett, Z. Matt) учёных, посвященные к основным идеям объектно-ориентированного подхода к программированию, а также сравнительному анализу данного подхода на языках JavaScript и PHP.

Для решения задач настоящей работы использовался комплекс теоретических **методов** исследования, таких как сравнительный анализ, обобщение, синтез, метод индукции и дедукции.

Основной **информационной базой** исследования послужили www.elibrary.ru, www.cyberleninka.ru, www.scholar.google.ru, www.medium.com и www.habr.com.

**Теоретический результат.** Проведенный сравнительный анализ ООП на языках JavaScript и PHP позволил:

— описать основные понятия объектно-ориентированного подхода, выделить его преимущества и недостатки в JavaScript и PHP;

— выявить две модели в качестве критерия для сравнительного анализа ООП на рассматриваемых языках программирования;

— рассмотреть частоту использования объектно-ориентированных конструкций в фреймворках JavaScript и PHP;

— сопоставить результаты сравнительного анализа друг с другом.

# 1 Общие сведения о языках JavaScript и PHP

## 1.1 Язык программирования JavaScript

В исследовании [6] отмечается, что изначально в структуре HTML-документа применялись лишь средства оформления текста (теги), ссылки на прочие веб-документы или графические файлы (изображения). Позднее появилась возможность вставки и проигрывания звуковых файлов и видео-клипов. Однако помимо размещения мультимедиа-контента на веб-странице существовала проблема динамической обработки экспозиции и управления просмотром. Возможностей стандартного HTML не хватало и потому возникла необходимость в привлечении иного языка программирования (по своим характеристикам приближенного к стандартным средствам разработки ПО).

Все изменилось в 1995 году, когда Брендан Айк, отец JavaScript из фирмы Netscape представил общественности новый браузер Netscape Navigator с поддержкой языка LiveScript. Функционал LiveScript был весьма ограничен, но для своего времени он оказался довольно практичным и прогрессивным. Позднее имя LiveScript заменили на JavaScript.

JavaScript – это лёгкий, интерпретируемый, объектно-ориентированный язык с функциями первого класса, самый известный скриптовый язык для веб-страниц, но также используется во многих не браузерных окружениях [15]. В работе [3] трактуется, что JavaScript также является прототипно-ориентированным языком программирования.

Прототипное программирование – стиль объектно-ориентированного программирования, при котором отсутствует понятия класса, а наследования производится путём клонирования существующего экземпляра объекта – прототипа [3].

## 1.2 Язык программирования PHP

В данной работе [5] говорится о том, что PHP прошёл долгий путь за последние несколько лет, становясь одним из наиболее популярных языков веб-разработки. Истоки PHP лежат в старом продукте, имевшем название PHP/FI. PHP/FI был создан Расмусом Лердорфом в 1995 году и представлял собой набор Perl-скриптов для ведения статистики посещений его резюме. Развитие веб еще только начиналось, никаких специальных средств для решения этих задач не было, и к автору хлынул поток сообщений с вопросами. Лердорф начал бесплатно раздавать свой инструментарий, названный «Personal Homepages Tools» (PHP) - («Инструменты для персональных домашних страниц»). Очень скоро потребовалась большая функциональность и Расмус пишет новую, намного более обширную версию на C, работающую с базами данных и позволяющую пользователям разрабатывать простейшие веб-приложения. Расмус Лердорф решил выложить исходный код PHP/FI на всеобщее обозрение, исправление ошибок и дополнение.

PHP/FI (Personal Home Page / Forms Interpreter – Персональная Домашняя страница / Интерпретатор Форм) включал в себя базовую функциональность сегодняшнего PHP. Он имел переменные в стиле Perl, автоматическую интерпретацию форм и возможность встраиваться в html-код. Собственно синтаксис языка имел много общего с Perl, хотя и был намного проще и ограниченнее.

В 1997 выходит PHP/FI 2.0. Вторая версия C-имплементации обозначила группу пользователей: несколько тысяч людей по всему миру, с примерно 50,000 доменами, что составляло около 1% всего числа доменов Интернета. Несмотря на то, что разработкой занималось уже несколько людей, PHP/FI 2.0 все еще оставался крупным проектом одного человека.

Официально PHP/FI 2.0 вышел только в ноябре 1997 года, после проведения большей части своей жизни в бета-версиях. Вскоре после выхода его заменили альфа-версии PHP 3.0.

PHP 3.0 был первой версией, напоминающей PHP, каким знают его сегодня. Посчитав PHP/FI 2.0 все еще неэффективным и недостаточно функциональным для использования в коммерческих приложениях, разрабатываемых для их университетского проекта, Энди Гутманс и Зив Сураски из Тель-Авива начали еще раз заново переписывать парсер в 1997 году. Связавшись с Расмусом, они обсудили различные аспекты текущей реализации и их новой разработки PHP. Для улучшения движка и использования уже существующей базы пользователей PHP/FI, Энди, Расмус и Зив решили сотрудничать в развитии нового, независимого языка программирования. Этот совершенно новый язык был выпущен под новым именем, без упоминания о персональном использовании, как в PHP/FI 2.0. Он был назван просто «PHP» – аббревиатура, означающая рекурсивный акроним – PHP: Hypertext Preprocessor.

Одной из сильнейших сторон PHP 3.0 была возможность расширения ядра. Кроме обеспечения пользователей надежной инфраструктурой из множества различных баз данных, протоколов и API, расширяемость PHP 3.0 привлекла к нему множество сторонних разработчиков, желающих добавить к языку свои модули. Возможно, это и был главный ключ к успеху, но стоит добавить, что немаловажным шагом оказалась поддержка ООП синтаксиса и намного более мощного и последовательного синтаксиса самого языка.

К зиме 1998 года, практически сразу после официального выхода PHP 3.0, Энди Гутманс и Зив Сураски начали переработку ядра PHP. В задачи входило увеличение производительности сложных приложений и улучшение модульности кодовой базы PHP. PHP 3.0 дал возможность подобным приложениям успешно работать с набором баз данных и поддерживать большое

количество различных API и протоколов, но PHP 3.0 не имел качественной поддержки модулей и приложения работали неэффективно.

Новый движок, названный «Zend Engine» (от имен создателей: Zeev и Andi), успешно справлялся с поставленными задачами и впервые был представлен в середине 1999 года. PHP 4.0, основанный на этом движке и принесший с собой набор дополнительных функций, официально вышел в мае 2000 года, почти через два года после выхода своего предшественника. Кроме значительного улучшения производительности, PHP 4.0 имел еще несколько ключевых нововведений, таких как поддержка намного большего числа веб-серверов, поддержка HTTP сессий, буферизация вывода, более безопасные способы обработки, вводимой пользователем информации и несколько новых языковых конструкций [4].

После долгой разработки и нескольких пре-релизов в июле 2004 был выпущен PHP 5. В основном он управляется ядром Zend Engine 2.0 с новой объектной моделью и множеством различных других нововведений.

Одним из самых значительных событий, произошедших в мире PHP в 2015 году, стал выпуск PHP 7.  Релиз PHP 7.0 стал в своем роде прорывом, так как он основывался на phpng – экспериментальном и активно развивающимся проекте PHP, название которого расшифровывается как «PHP Next Generation» – «Следующее поколение PHP». Данную технологию можно было использовать и в предыдущих версиях PHP, однако именно в PHP 7 она была введена как основополагающая. При ее разработке была поставлена цель повысить производительность PHP, но при этом не потерять совместимость. PHP 7 основан на третьей версии Zend Engine, в которой был развит проект phpng [4].

Команда разработчиков PHP включает в себя десятки разработчиков, а также десятки других организаций, работающих над связанными с PHP и его поддержкой проектами, такими как PEAR, PECL и документацией, а также базовую инфраструктуру сети более чем из ста серверов на шести из семи

континентах мира. Основываясь на статистике прошлых лет, можно с уверенностью предположить, что PHP теперь установлен на десятки или даже, возможно, сотни миллионов доменов по всему миру.

# 2 Основные критерии сравнительного анализа

## 2.1 Модели реализации ООП в JavaScript и PHP

Модель реализации ООП на рассматриваемых языках программирования является первом критерием, на основе которого можно провести сравнительный анализ в настоящей работе.

В JavaScript ООП, который реализуется на основе прототипной модели не использует классы, а вместо этого сначала выполняет поведение класса и затем использует его, декорируя (или расширяя) существующие объекты «прототипы» (Также называемое бесклассовое, прототипно-ориентированое, или экземплярно-ориентированное программирование) [3]. Данное утверждение может быть обосновано на примере, который показан на Рисунке 1. На данным примере представлен процесс создания объектов «Audi», «Nissan», «Volvo» и их наследования от вышестоящих объектов посредством прототипной цепочки. Нижнюю ступень цепочки занимают объекты «Audi», «Nissan», «Volvo», чьи свойства «__proto__» равны прототипу («Car.prototype») функции-конструктора «Car». Следовательно, все свойства зарегистрированные в «Car.prototype» доступны всем трем объектам («Audi», «Nissan», «Volvo»). Свойство «__proto__» функции-конструктора «Car» равно прототипу («Function.prototype») глобальной функции «Function». Следовательно, все свойства зарегистрированные в «Function.prototype» доступны функции-конструктора «Car». Свойство «__proto__» глобальной функции «Function» не равно «Object.prototype», который является прототипным объектом объекта «Object», встроенного в ядро JavaScript. Каждый объект в JavaScript является экземпляром объекта «Object», следовательно наследует все его свойства и методы. В данным случае глобальная функция не является исключением. Но как данная функция может получить доступ к свойствам и методам «Object», если свойство «__proto__» глобальной

функции «Function» не равно «Object.prototype»? Глобальная функция «Function» получает доступ к свойствам и методам «Object» с помощью свойства «Function.prototype.__proto__», которое равно прототипу «Object.prototype» объекта «Object». Из этого примера следует, что в JavaScript ООП, который реализуется на основе прототипной модели не использует классы, а использует прототипы.

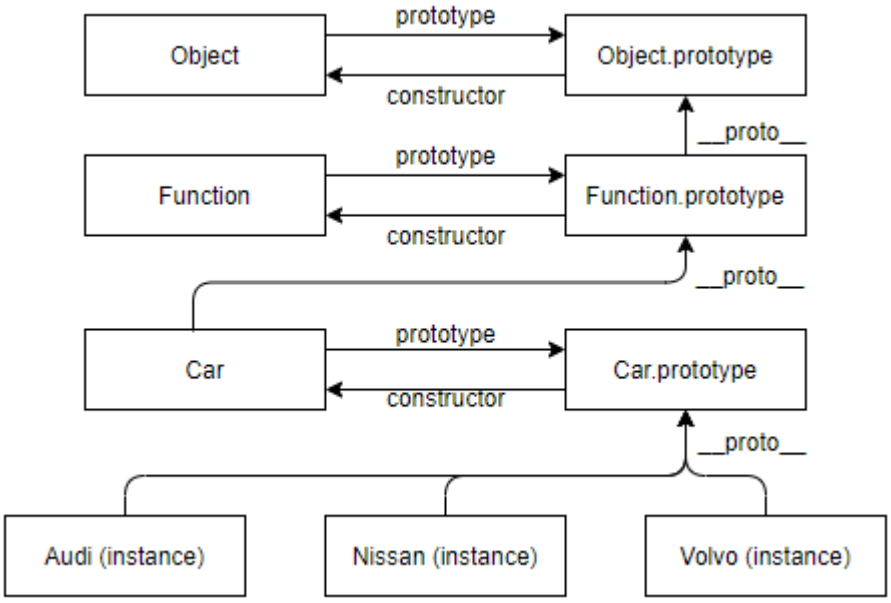


Рисунок 1 – Иллюстрация создания объектов (экземпляров) на основе прототипной модель в JavaScript

С другой стороны, в JavaScript версии 6 (ES6) ключевое слово «class» стало так называемым синтаксическим сахаром для создания множества объектов по прототипу.

В отличие от прототипной модели JavaScript, ООП в PHP имплементируется с помощью модели на основе классов [13]. На Рисунке 2 представлена иллюстрация создания экземпляров (объектов) класса машины. На данным примеры классом является первая машина «Car», которая нарисована пунктирными линиями. Объектами, которые создаются на основе класса «Car»

являются машины с названиями «Audi», «Nissan» и «Volvo». Эти машины закрашены в черный цвет. Из этого примера следует, что, создав один класс, можно создавать бесконечно множества объектов (экземпляров) на основе данного класса. Данный пример четко иллюстрирует взаимосвязь между классом и объектом в PHP.

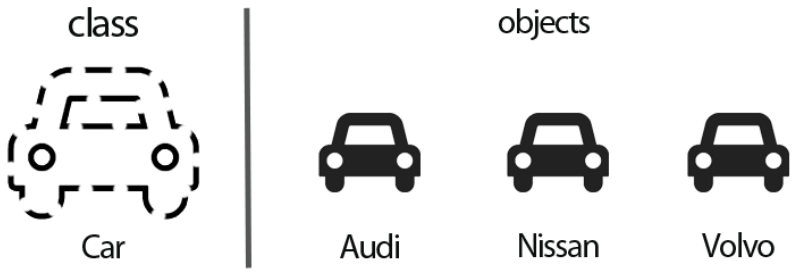


Рисунок 2 – Иллюстрация создания объектов на основе класса в PHP

## 2.2 Синтаксис реализации ООП в JavaScript и PHP

Поскольку JavaScript является прототипно-ориентированным языком программирования, в нём нет оператора «class», который имеет место в PHP или Java. Иногда это сбивает с толку программистов, привыкших к языкам с оператором «class». Вместо этого JavaScript использует функции как конструкторы классов [9]. Объявить класс так же просто как объявить функцию. На Рисунке 3 представлен пример объявления нового класс «Person» с пустым конструктором.

```
1 | var Person = function () {};
```

Рисунок 3 – Пример объявления класса «Person» в JavaScript

С другой стороны, в PHP каждое определение классов начинается с ключевого слово «class», затем следует имя класса, и далее пара фигурных

скобок, которые заключают в себе определение свойств и методов этого класса [12]. На Рисунке 4 представлен пример объявления нового пустого класса «Person», использую ключевое слово class.

```php
<?php
// Создается пустой класс Person
class Person {

}

?>
```

Рисунок 4 – Пример объявления класса «Person» в PHP

В JavaScript для создания нового экземпляра объекта «obj» используется оператор «new obj», присваивая результат (который имеет тип «obj») в переменную [16]. На Рисунке 5 представлен пример создания двух экземпляров (объектов) «person1» и «person2» класса «Person».

```javascript
var person1 = new Person();
var person2 = new Person();
```

Рисунок 5 – Пример создания экземпляров (объектов) класса «Person» в JavaScript

В PHP создания нового экземпляра класса практически не отличается от создания нового экземпляра класса в JavaScirpt. Например, для создания нового экземпляра (объекта) класса «Person» используется оператор «new», присваивая результат в переменную. На Рисунке 6 представлен пример создания двух экземпляров («person»1 и «person2») класса «Person» в PHP.

```php
$person1 = new Person();
$person2 = new Person();

```

Рисунок 6 – Пример создания экземпляров (объектов) класса «Person» в PHP

Свойства – это переменные, содержащиеся в классе; каждый экземпляр объекта имеет эти свойства. Свойства устанавливаются в конструкторе (функции) класса, таким образом они создаются для каждого экземпляра.

Ключевое слово «this», которое ссылается на текущий объект, позволяет вам работать со свойствами класса. Доступ (чтение и запись) к свойствам снаружи класса осуществляется синтаксисом «InstanceName.Property», так же как в C++ Java и в некоторых других языках [18]. (Внутри класса для получения и изменения значений свойств используется синтаксис «this.Property») На Рисунке 6 представлен пример, в которым определяется свойство «firstName» для класса «Person» при создании экземпляра в JavaScript.

```
var Person = function (firstName) {
  this.firstName = firstName;
  console.log('Person instantiated');
};

var person1 = new Person('Alice');
var person2 = new Person('Bob');

// Выводит свойство firstName в консоль
console.log('person1 is ' + person1.firstName); // выведет "person1 is Alice"
console.log('person2 is ' + person2.firstName); // выведет "person2 is Bob"
```

Рисунок 7 – Пример создания свойств класса «Person» в JavaScript

Что касается свойств класса в PHP, при их описании необходимо указывать модификатор доступа – ключевое слово, которое будет определять область видимости поля, к которому оно относится. В PHP есть три модификатора доступа: «public», «protected» и «private». Спецификатор «public» обеспечивает доступ к полю из любого места, «protected» – только из классов, стоящих в той же цепочки наследования и «private» запрещающий доступ ото всюду, кроме

самого класса. Данные модификаторы доступа позволяет реализовать принцип работы инкапсуляции, определяя разные уровни доступа к свойствам и методам классов [11]. На Рисунке 8 представлен пример создания двух свойств («$firstName», «$lastName») класса с модификатором доступа «public».

```php
<?php
// Создается класс Person с полями firstName и lastName;
class Person {
     public $firstName = "John";
     public $lastName = "Doe";
}
```

Рисунок 8 – Пример создания свойств класса «Person» в PHP

В PHP для доступа к полям класса используется символ «–>». Имя поля, к которому необходимо обратиться пишется без знака доллара. Поля доступны извне класса в этом случае, потому что используется спецификатор доступа «public» [12]. На Рисунке 9 представлен пример обращения к полям (свойствам) класса «Person».

```php
<?php
// Создается класс Person с полями firstName и lastName;
class Person {
     public $firstName = "John";
     public $lastName = "Doe";
}

// Создается объект person1
$person1 = new Person();

echo("$person1->firstName <br/>"); // Выведет "John"
echo("$person1->lastName"); // Выведит "Doe"
?>
```

Рисунок 9 – Пример обращения к свойствам класса «Person» в PHP

Методы – это функции (и определяются как функции), но, с другой стороны, следуют той же логике, что и свойства. В JavaScript вызов метода похож на доступ к свойству, но добавляется «()» на конце имени метода, возможно, с

аргументами. Чтобы объявить метод, функция присваивается в именованное свойство свойства «prototype» класса. Потом можно вызвать метод объекта под тем именем, которое выло присвоено функции [14]. На Рисунке 10 представлен пример определения и использования метода «sayHello()» для класса «Person».

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

Person.prototype.sayHello = function() {
  console.log("Hello, I'm " + this.firstName);
};

var person1 = new Person("Alice");
var person2 = new Person("Bob");

// вызываем метод sayHello() класса Person
person1.sayHello(); // выведет "Hello, I'm Alice"
person2.sayHello(); // выведет "Hello, I'm Bob"
```

Рисунок 10 – Пример определения и использования методов класса «Person» в JavaScript

Как говорилось ранее, методы описывают поведения экземпляров класса, то есть действия, которые они могут совершать. В PHP описания метода в классе, как и описание поля, начинается со спецификатора доступа, затем следует ключевое слово «function», имя метода и список параметров в круглых скобках. Внутри метода доступ к текущему экземпляру класса можно получить при помощи ключевого слова «$this», которое, в отличии от многих других языков программирования, в PHP пишется со знаком доллара. За пределами класса вызов методов производится с указанием имени экземпляра класса. Как и для доступа к полям, для вызова методов используется символ «–>» [20]. На Рисунке 11 приведен пример создания метода «printFullName» класса «Person», который

производит некоторые манипуляции с полями класса и выводит определенный результат.

```php
<?php
// Создается класс Person с полями firstName и lastName;
class Person {
     public $firstName;
     public $lastName;

     // Создается метод printFullName
     public function printFullName() {
        return $this->firstName." ".$this->lastName;
     }
}

// Создается объект person1
$person1 = new Person();

// Присваиваются значения к полям firtName и lastName
$person1->firstName = "John";
$person1->lastName = "Doe";

echo $person1->printFullName(); // Выведется "John Doe"
?>
```

Рисунок 11 – Пример определения и создания методов класса «Person» в PHP

Наследование – это способ создать класс как специализированную версию одного или нескольких классов. В JavaScript поддерживается только одиночное наследование. Специализированный класс, как правило, называют потомком, а другой класс родителем. В JavaScript наследование осуществляется присвоением экземпляра класса родителя классу потомка. В современных браузерах можно реализовать наследование с помощью «Object.create» [16].

На Рисунке 13 класс Student определяется как потомка класса Person. Потом переопределяется метод sayHello() и добавляется метод addGoodBye(). Классу «Student» нет необходимости знать о реализации метода «walk()» класса «Person», но он может его использовать. Класс «Student» не должен явно определять этот метод, до тех пор, пока не захочется его изменить. Это называется

«инкапсуляция», благодаря чему каждый класс собирает данные и методы в одном блоке.

Сокрытие информации распространённая особенность, часто реализуемая в других языках программирования как приватные и защищённые методы или свойства. Однако в JavaScript можно лишь импортировать нечто подобное, это не является необходимым требованием объектно-ориентированного программирования [18].

Относительно строки «Student.prototype = Object.create(Person.prototype);», в старых движках JavaScript, в которых нет «Object.create» можно использовать полифилл (ещё известный как «shim») или функцию которая достигает тех же результатов. На Рисунке 12 представлена реализация данного функционала.

```
function createObject(proto) {
    function ctor() { }
    ctor.prototype = proto;
    return new ctor();
}

// Пример использования:
Student.prototype = createObject(Person.prototype);
```

Рисунок 12 – Пример замены «Object.create» с простой функцией

```
// Определяем конструктор Person
var Person = function(firstName) {
  this.firstName = firstName;
};

// Добавляем пару методов в Person.prototype
Person.prototype.walk = function(){
  console.log("I am walking!");
};

Person.prototype.sayHello = function(){
  console.log("Hello, I'm " + this.firstName);
};

// Определяем конструктор Student
function Student(firstName, subject) {
  // Вызываем конструктор родителя, убедившись (используя Function#call)
  // что "this" в момент вызова установлен корректно
  Person.call(this, firstName);

  // Инициируем свойства класса Student
  this.subject = subject;
};

// Создаём объект Student.prototype, который наследуется от Person.prototype.
// Примечание: Рспространённая ошибка здесь, это использование "new Person()", чтобы создать
// Student.prototype. Это неверно по нескольким причинам, не в последнюю очередь
// потому, что нам нечего передать в Person в качестве аргумента "firstName"
// Правильное место для вызова Person показано выше, где мы вызываем
// его в конструкторе Student.
Student.prototype = Object.create(Person.prototype); // Смотрите примечание выше

// Устанавливаем свойство "constructor" для ссылки на класс Student
Student.prototype.constructor = Student;

// Заменяем метод "sayHello"
Student.prototype.sayHello = function(){
  console.log("Hello, I'm " + this.firstName + ". I'm studying "
              + this.subject + ".");
};

// Добавляем метод "sayGoodBye"
Student.prototype.sayGoodBye = function(){
  console.log("Goodbye!");
};

// Пример использования:
var student1 = new Student("Janet", "Applied Physics");
student1.sayHello();   // "Hello, I'm Janet. I'm studying Applied Physics."
student1.walk();       // "I am walking!"
student1.sayGoodBye(); // "Goodbye!"

// Проверяем, что instanceof работает корректно
console.log(student1 instanceof Person);  // true
console.log(student1 instanceof Student); // true
```

Рисунок 13 – Пример наследования в JavaScript

Наследование в PHP – это хорошо зарекомендовавший себя принцип программирования, и PHP использует этот принцип в своей модели на основе классов [21]. Этот принцип влияет на то, как многие классы и объекты связаны друг с другом. Например, при расширении класса дочерний класс наследует все общедоступные и защищенные методы из родительского класса. До тех пор, пока эти методы не будут переопределены, они будут сохранять свою исходную функциональность. Это полезно для определения и абстрагирования функциональности и позволяет реализовать дополнительную функциональность в похожих объектах без необходимости реализовывать всю общую функциональность. На Рисунке 14 представлен пример наследования в PHP.

```php
<?php
class Foo {
    public function printItem($string) {
        echo 'Foo: ' . $string . PHP_EOL;
    }

    public function printPHP() {
        echo 'PHP просто супер.' . PHP_EOL;
    }
}

class Bar extends Foo {
    public function printItem($string) {
        echo 'Bar: ' . $string . PHP_EOL;
    }
}

$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Выведет: 'Foo: baz'
$foo->printPHP();       // Выведет: 'PHP просто супер'
$bar->printItem('baz'); // Выведет: 'Bar: baz'
$bar->printPHP();       // Выведет: 'PHP просто супер'
?>

```

Рисунок 14 – Пример наследования в PHP

Что касается абстракции в JavaScript, это механизм, который позволяет смоделировать текущий фрагмент рабочей проблемы, с помощью наследования (специализации) или композиции. JavaScript достигает специализации

наследованием, а композиции возможностью экземплярам класса быть значениями атрибутов других объектов [3].

С другой стороны, в PHP существует абстрактный класс, который не может быть реализован, то есть, нельзя создать объект класса, если он абстрактный. Вместо этого создаются дочерние классы от него и спокойно создаете объекты от этих дочерних классов. Абстрактные классы представляют собой шаблоны для создания классов [12]. На Рисунке 15 представлен пример реализации абстрации веб-сайта, в котором есть как участники форума, так и покупатели онлайн-магазина, который является частью сайта. Так как и участники форума и покупатели - люди, можно создать абстрактный класс «Person», в котором будут какие-то поля и методы, общие для всех пользователей сайта.

```php
<?php
abstract class Person {
  private $firstName = "";
  private $lastName = "";

  public function setName( $firstName, $lastName ) {
    $this->firstName = $firstName;
    $this->lastName = $lastName;
  }

  public function getName() {
   return "$this->firstName $this->lastName";
  }

  abstract public function showWelcomeMessage();
}

class Member extends Person {
  public function showWelcomeMessage() {
    echo "Hi " . $this->getName() . ", welcome to the forums!<br>";
  }

  public function newTopic( $subject ) {
    echo "Creating new topic: $subject<br>";
  }
}

class Shopper extends Person {
  public function showWelcomeMessage() {
    echo "Hi " . $this->getName() . ", welcome to our online store!<br>";
  }

  public function addToCart( $item ) {
    echo "Adding $item to cart<br>";
  }
}
?>
```

Рисунок 15 – Пример реализации абстракции в PHP

Из примера видно, что создается абстрактный класс, добавив в его описание ключевое слово abstract. В этом классе есть несколько свойств, общих

для всех людей, - «$frstName» и «$lastName» - а также методы для инициализации и чтения значений этих полей. В классе также есть абстрактный метод «showWelcomeMessage()». Этот метод выводит приветствие, когда пользователь входит на сайт. Опять же, добавляется ключевое слово «abstrac»t в описание данного метода, чтобы сделать его абстрактным. Так как он абстрактный, в нем нет ни строчки кода, это просто его объявление. Тем не менее, любой дочерний класс обязан добавить и описать метод «showWelcomeMessage()».

Так как все методы и свойства определяются внутри свойства prototype, различные классы могут определять методы с одинаковыми именами; методы находятся в области видимости класса, в котором они определены, пока два класса не имеют связи родитель-потомок (например, один наследуется от другого в цепочке наследований) [3]. Данное утверждение характеризует функционал полиморфизма в JavaScript.

Полиморфизм в PHP – это поддержка нескольких реализаций на основе одного общего интерфейса [25]. Если сказать несколько иначе, это взаимозаменяемость объектов, реализующих один интерфейс. На Рисунке 16 рассматривается вышеуказанное утверждение. На первом этапе создается интерфейс «Animal» с методом «giveSound()». Далее, создаются два класса «Dog» и «Cat», которые реализуют данный интерфейс и метод «giveSound()». Каждый метод реализует данный метод по-своему. Метод «giveSound()», который реализован в классе «Dog», возвращает строку «Гаф, гаф!». Что касается метода «giveSound()», который реализован в классе «Cat», возвращает строку «Мяу мяу!».

```php
<?php

interface Animal {
    public function giveSound();
}

class Dog implements Animal {
    public function giveSound() {
        return "Гаф, гаф!";
    }
}

class Cat implements Animal {
    public function giveSound() {
        return "Мяу, мяу!";
    }
}

$dog1 = new Dog();
$cat1 = new Cat();

echo $dog1->giveSound(); // Выведется "Гаф, гаф!"
echo $cat1->giveSound(); // Выведется "Мяу, мяу!"
```

Рисунок 16 – Пример реализации полиморфизма в PHP

Таким образом, на лицо взаимозаменяемость объектов, реализующих один и тот же интерфейс. Конечно, указанный пример – очень прост, но так или иначе он отлично показывает, что собой представляет полиморфизм в объектно-ориентированном программировании PHP.

## 2.3 Частота использования ООП библиотек и фреймворков

В настоящее время существует множества библиотек и фреймворков, которые реализуют объектно-ориентированный подход в своих программно-интерфейсных приложениях (Application Programming Interface) как для JavaScript, так и для PHP.

Следующим критерием, на основе которого можно провести сравнительный анализ реализации объектно-ориентрованного подхода, является частота

использования объектно-ориентированных конструкций. Для этого рассматриваются популярные фреймворки и библиотеки JavaScript и PHP.

В современной экосистеме представлены такие JavaScript-библиотеки и фреймворки как React.js, Angular.js, Vue.js, Ember.js, Polymer, Aurelia, Sail.js которые реализуют объектно-ориентированные конструкции. Наряду с этими инструментами существует и такие фреймворки как Express.js, Koa.js, Hapi.js, Данные фреймворки не реализует объектно-ориентированный подход, а реализует подход промежуточного программного обеспечения («middleware»). Данный подход имплементируется на основе принципов функционального программирования, где используются чистые функции и функции высшего порядка, а не классы или экземпляры (объекты) данных классов [16].

Практически все PHP-фреймворки являются объектно-ориентированными, в отличие от JavaScript фреймворков [21]. Например, такие популярные PHP-фреймворки как Laravel, Yii, Symfony, Phalcon, CodeIgniter, CakePHP, Zend, FuelPHP, PHPixie, Slim основываются на принципах объектно-ориентированного подхода к программированию при решении тех или иных предметных задач.

# 3 Результаты сравнительного анализа

В настоящей работе при проведении исследовании были выявлены следующие критерии для проведения сравнительного анализа:

1. Модель реализации ООП
2. Синтаксис языков
3. Частота использования ООП конструкций

Все вышеперечисленные критерии, на основе которых проводился сравнительная анализ, способствовали выявлению как аналогичных, так и противоположных аспектов реализации ООП на рассматриваемых языках программирования. Например, в результате сравнительного анализа, который проводился на основе модели реализации ООП в JavaScript и PHP, удалось выделить две следующие модели:

1. Прототипная модель
2. Модель на основе классов

Для того чтобы реализовать ООП на языке JavaScript главном образом используется первая модель, а на языке PHP вторая. Результаты сравнительного анализа показали, что первая модель отличается от второй тем, что в первый модели не используются классы, а используются так называемые прототипы. Во второй модели используются исключительно классы для реализации ООП в PHP.

Результаты сравнительного анализа, проведенного на основе синтаксиса языков показали, что реализация ООП в JavaScript во многом зависит от функции-конструктор, который позволяет создавать объекты (экземпляры). С другой стороны, в PHP используется классы для создания объектов (экземпляров) данного класса. Также в JavaScript есть объект prototype, который является одним из ключевых объектов при реализации ООП функционала. Кроме этого, результаты сравнительного анализа показывают, что в JavaScript функционал

наследования достигается присвоением экземпляра класса родителя классу потомка. В современных браузерах наследование можно реализовать с помощью Object.create. С другой стороны, в синтаксисе PHP наследование осуществляется ключевым словом extends, которое позволяет классам-потомкам унаследовать свойства и методы суперкласса.

Что касается результатов, полученных при проведении сравнительного анализа на основе частоты использования ООП конструкций, большинство PHP библиотек и фреймворков реализуют ООП в своих функционалах. В индустрии насчитывается более двадцати известных PHP фреймворков, которые реализуют ООП. С другой стороны, в экосистеме JavaScript существуют как функциональные, так и ООП библиотеки и фреймворки для данного языка программирования. Количество известных ООП библиотек и фреймворков для JavaScript заметно меньше, чем для PHP, так как для него существуют не мало функциональных библиотек и фреймворков.

# Заключение

В настоящей исслодвательской работе на первом этапе были освящены общие сведения о языках программирования JavaScript и PHP, ключевые особенности объектно-ориентированного подход на данных языках программированию. Более того, был приведён обзор истории развития рассматриваемых языков программирования.

На втором этапе были вявлены такие криетрии как модель реализации ООП, синтаксис языков и частота использования ОПП конструкций, на основе которых проводился сравнтиельный анализ ООП на яызках программированя. Более того, каждый критерий был рассмотрен отдельно от других критерии. Это позволило сфокусирваться на одном аспекты реализации ООП и получить достаточно показательные результаты. Кроме этого, при провдении сравнительного ананлиза были использованы фрагменты кода реализации ООП на языках и болщая разновидность иллюстративных материалов. Выявленные критерии позволили провести очень показательный сравнительный ананлиз, благодаря которому удалось всецело понять как анналогичные так и противоположные аспекты реализации ООП на языках JavaScript и PHP.

На третим этапе, более детально были рассмотрены результаты, проведенного стравнительно ананлиза. Результаты сравнительного ананлиза показали, что будучи двумя основными языками веб-разработки, JavaScript и PHP развиваются конвергентно, дополняя друг друга. PHP тяготеет к приоритету объектно-ориентированного подхода, но по мере развития обогащается функциональными чертами.

В дальнейшем на языках JavaScript и PHP предполагается разработать инновационное веб-приложение, с помощью которого можно будет понять практическую значимость данной курсовой работы.

# Список использованных источников


1. eLibrary – Научная электронная библиотека [Электронный ресурс]. Режим доступа: https://elibrary.ru/ (Дата обращения: 20.03.2020).
2. Академия Google [Электроный ресурс]. Режим доступа: https://scholar.google.ru/ (Дата обращения: 21.03.2020).
3. Вступление в объектно-ориентированный JavaScript [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript (Дата обращения: 17.03.2020).
4. История PHP [Электронный ресурс]. Режим доступа: https://www.php.net/manual/ru/history.php.php (Дата обращения: 24.03.2020).
5. История развития PHP [Электронный ресурс]. Режим доступа: http://www.php.su/php/?history (Дата обращения: 30.03.2020).
6. История создания языка программирования JavaScript [Электронный ресурс]. Режим доступа: https://web.informatics.ru/works/17-18/web_online/barabanov_n_v/language_js.html (Дата обращения: 12.03.2020).
7. КиберЛенинка – Научная электронная библиотека [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/ (Дата обращения: 29.03.2020).
8. Краткая история JavaScript. Част 1 [Электронный ресурс]. Режим доступа: https://habr.com/ru/company/livetyping/blog/324196/ (Дата обращения: 12.03.2020)

9. Краткое руководство по ОПП в JS [Электронный ресурс]. Режим доступа: https://medium.com/nuances-of-programming/краткое-руководство-по-ооп-в-js-1f54f9d50067/ (Дата обращения: 15.03.2020).
10. М.А. Кузнецов, И.Н. Симдянов PHP 7. – Санкт-Петербург, 2018. – №1 С. 24-60.
11. М.С. Фленов PHP глазами хакера. – Санкт-Петербург, 2018. – №3 С. 44-52.
12. ООП и PHP. Часть 1 - наследование [Электронный ресурс]. Режим доступа: https://habr.com/ru/sandbox/2532/ (Дата обращения: 16.03.2020).
13. Основы PHP и ООП [Электронный ресурс]. Режим доступа: https://habr.com/ru/sandbox/47247/ (Дата обращения: 01.04.2020).
14. Фундаментальные приципы объектно-ориентированного программирования на JavaScript [Электронный ресурс]. Режим доступа: https://tproger.ru/translations/oop-js-fundamentals/ (Дата обращения: 06.04.2020).
15. Что такое JavaScript ? [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript/%D0%9E_JavaScript (Дата обращения: 02.04.2020).
16. A Guide to Object-Oriented Programming in JavaScript [Электронный ресурс]. Режим доступа: https://medium.com/better-programming/object-oriented-programming-in-javascript-b3bda28d3e81 (Дата обращения: 07.04.2020).
17. D.M. Brett Head first object-oriented analysis and design // Eighth IEEE International Symposium on Network Computing and Applications. — US IEEE, 2010. — P. 124-139.
18. D. Crockford JavaScript: The Good Parts, First Edition – US. – 2008 Vol. 228. – P. 92 – 115.
19. JavaScript: ООП, функциональный стиль [Электронный ресурс]. Режим доступа: https://habr.com/ru/sandbox/123183/ (Дата обращения: 17.03.2020).

20. Z. Matt PHP Objects, Patterns and Practice, Second Edition – US Computers & Education. — 2008. — Vol. 80. — P. 92–115.
21. Object-Oriented PHP – An Easy Approach [Электронный ресурс]. Режим доступа: https://medium.com/ph-devconnect/object-oriented-php-an-easy-approach-6774c425854a (Дата обращения: 02.04.2020).
22. Object-Oriented Programming Concepts in PHP – Part 1 [Электронный ресурс]. Режим доступа: https://medium.com/valuebound/object-oriented-programming-concepts-in-php-part-1-9332c230f1f2 (Дата обращения: 03.04.2020).
23. PHP is bad for Object-Oriented Programming (aka OOP) [Электронный ресурс]. Режим доступа: https://medium.com/cook-php/php-is-bad-for-object-oriented-programming-oop-b9295a90bf00 (Дата обращения: 04.04.2020).
24. S. Sanders PHP Design Patterns – US Computers & Education. — 2018. — Vol. 224. — P. 72–76.
25. Understanding Polymorphism in PHP [Электронный ресурс]. Режим доступа: https://medium.com/@iamjoestack/understanding-polymorphism-in-php-3d2670deb6e1 (Дата обращения: 05.04.2020).