Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 0**

**Инструкционно-технологическая карта**

Тема: Комментарии. Качество кода. Плохие и хорошие комментарии

Цель: Научиться писать корректные комментарии.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Теоретические сведения

Комментарии делятся на два типа: однострочные, начинающиеся с **//** и многострочные, вида **/\* ... \*/**.

Обычно они используются, чтобы описать код.

На первый взгляд, в комментариях нет ничего сложного, но новички в программировании часто применяют их неправильно.

## Плохие комментарии

Новички склонны использовать комментарии, чтобы объяснять, «что делает код». Например, так:

// Этот код делает это (...) и вот это (...)

// ...и кто знает, что ещё...

очень;

сложный;

код;

Но в хорошем коде количество «объясняющих» комментариев должно быть минимальным. Код должен быть таким, чтобы его можно было понять без комментариев.

Про это есть хорошее правило: «Если код настолько запутанный, что требует комментариев, то, может быть, его стоит переделать?»

#### Рецепт: выносите код в функции

Иногда выгодно заменить часть кода функцией, например, в таком случае:

function showPrimes(n) {

    nextPrime:

    for (let i = 2; i < n; i++) {

        // проверяем, является ли i простым числом

        for (let j = 2; j < i; j++) {

            if (i % j == 0) continue nextPrime;

        }

        alert(i);

    }

}

Лучший вариант – использовать отдельную функцию **isPrime**:

function showPrimes(n) {

    for (let i = 2; i < n; i++) {

        if (!isPrime(i)) continue;

        alert(i);

    }

}

function isPrime(n) {

    for (let i = 2; i < n; i++) {

        if (n % i == 0) return false;

    }

    return true;

}

Теперь код легче понять. Функция сама становится комментарием. Такой код называется **самодокументированным**.

#### Рецепт: создавайте функции

И если имеется такой длинный кусок кода:

// здесь мы добавляем виски

for (let i = 0; i < 10; i++) {

    let drop = getWhiskey();

    smell(drop);

    add(drop, glass);

}

// здесь мы добавляем сок

for (let t = 0; t < 3; t++) {

    let tomato = getTomato();

    examine(tomato);

    let juice = press(tomato);

    add(juice, glass);

}

  // ...

То будет лучше отрефакторить его с использованием функций:

addWhiskey(glass);

addJuice(glass);

function addWhiskey(container) {

    for (let i = 0; i < 10; i++) {

        let drop = getWhiskey();

        //...

    }

}

function addJuice(container) {

    for (let t = 0; t < 3; t++) {

        let tomato = getTomato();

        //...

    }

}

Здесь комментарии тоже не нужны: функции сами говорят, что делают. И ещё, структура кода лучше, когда он разделён на части. Понятно, что делает каждая функция, что она принимает и что возвращает.

В реальности мы нельзя полностью избежать «объясняющих» комментариев. Существуют сложные алгоритмы. И есть хитрые уловки для оптимизации. Но в целом нужно стараться писать простой и самодокументированный код.

## Хорошие комментарии

#### Описывание архитектуры

Следует делать высокоуровневый обзор компонентов, того, как они взаимодействуют, каков поток управления в различных ситуациях… Если вкратце – обзор кода с высоты птичьего полёта.

#### Документация параметров и использование функций

Есть специальный синтаксис **JSDoc** для документирования функций: использование, параметры, возвращаемое значение.

Например:

/\*\*

 \* Возвращает x, возведённое в n-ную степень.

 \*

 \* @param {number} x Возводимое в степень число.

 \* @param {number} n Степень, должна быть натуральным числом.

 \* @return {number} x, возведённое в n-ную степень.

 \*/

function pow(x, n) {

    ...

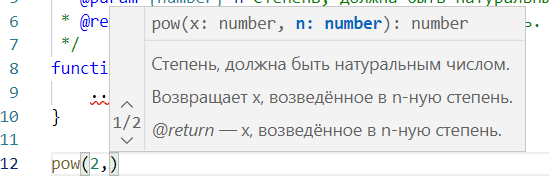
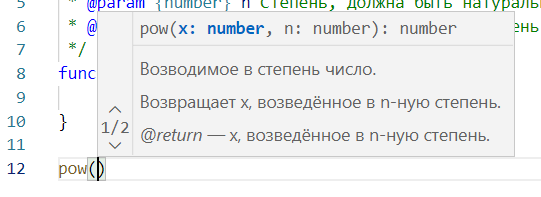
}

Подобные комментарии позволяют понимать назначение функции и правильно её использовать без необходимости заглядывать в код.

Многие редакторы, такие как WebStorm, прекрасно их распознают для того, чтобы выполнить автодополнение ввода и различные автоматические проверки кода.

Также существуют инструменты, например, JSDoc 3, которые умеют генерировать HTML-документацию из комментариев.

Результат применения JSDoc:



#### Почему задача решена именно таким способом?

Важно то, что написано. Но то, что не написано, может быть даже более важным, чтобы понимать происходящее. Почему задача решена именно этим способом? Код не даёт ответа.

Если есть несколько способов решить задачу, то почему вы выбрали именно этот? Особенно если ваш способ – не самый очевидный.

Без подобных комментариев возможна следующая ситуация:

1. Программист открывает написанный некоторое время назад код и видит, что в нём есть, что улучшить.
2. Он думает: «Каким глупым я раньше был и насколько умнее стал сейчас», и переписываете его на «более правильный и оптимальный» вариант.
3. …Желание переписать код – это хорошо. Но в процессе он понимаете, что «оптимальное» решение на самом деле не такое уж и оптимальное. Он даже смутно припоминает, почему, так как в прошлый раз вы уже его пробовал. Он возвращается к правильному варианту, потратив время зря.

Комментарии, объясняющие решение, очень важны. Они помогают продолжать разработку в правильном направлении.

# Контрольные вопросы

1. Что такое комментарии?
2. Что подразумевается под плохими комментариями?
3. Что такое самодокументированный код?
4. Что такое JSDoc?
5. Какие комментарии следует оставлять, а какие нет?

# Задания

Реализуйте задания по вариантам с **обязательным комментированием** кода после ознакомления с теоретической частью.

|  |  |
| --- | --- |
| Вар. | Задание |
|  | Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (например, по автору или по году издания), добавления книг в библиотеку, удаления книг из нее, сортировки книг по разным полям. |
|  | Составить описание класса прямоугольников со сторонами, параллельными осям координат. Предусмотреть возможность перемещения прямоугольников на плоскости, изменение размеров, построение наименьшего прямоугольника, содержащего два заданных прямоугольника, и прямоугольника, являющегося общей частью (пересечением) двух прямоугольников. |
|  | Создать класс для хранения комплексных чисел. Реализовать операции над комплексными числами: сложение, вычитание, умножение, деление, сопряжение, возведение в степень, извлечение корня. Предусмотреть возможность изменения формы записи комплексного числа: алгебраическая форма, тригонометрическая форма, экспоненциальная форма. |
|  | Составить описание класса для представления времени. Предусмотреть возможности установки времени и изменения его отдельных полей (час, минута, секунда) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения. Создать методы изменения времени на заданное количество часов, минут и секунд. |
|  | Составить описание класса для вектора, заданного координатами его концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами. |
|  | Описать класс, представляющий треугольник. Предусмотреть методы для создания объектов, вычисления площади, периметра и точки пересечения медиан. Описать свойства для получения состояния объекта. |
|  | Создать класс с двумя переменными. Добавить функцию вывода на экран и функцию изменения этих переменных. Добавить функцию, которая находит сумму значений этих переменных, и функцию которая находит наибольшее значение из этих двух переменных. |
|  | Описать класс, реализующий десятичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения, — и свойство, позволяющее получить его текущее состояние. Написать программу, демонстрирующую все возможности класса. |
|  | Создать класс с двумя переменными. Добавить конструктор с входными параметрами. Добавить конструктор, инициализирующий члены класса по умолчанию. Добавить деструктор, выводящий на экран сообщение об удалении объекта. |
|  | Создать класс, содержащий динамический массив и количество элементов в нем. Добавить конструктор, который выделяет память под заданное количество элементов, и деструктор. Добавить методы, позволяющие заполнять массив случайными числами, переставлять в данном массиве элементы в случайном порядке, находить количество различных элементов в массиве, выводить массив на экран. |
|  | Составить описание класса для определения одномерных массивов строк фиксированной длины. Предусмотреть контроль выхода за пределы массива, возможность обращения к отдельным строкам массива по индексам, выполнения операций поэлементного сцепления двух массивов с образованием нового массива, слияния двух массивов с исключением повторяющихся элементов, а также вывод на экран элемента массива по заданному индексу и всего массива. |
|  | Составить описание класса многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Предусмотреть методы для вычисления значения многочлена для заданного аргумента, операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена, вывод на экран описания многочлена. |
|  | Создайте класс с именем student, содержащую поля: фамилия и инициалы, номер группы, успеваемость (массив из пяти элементов). Создать массив из десяти элементов такого типа, упорядочить записи по возрастанию среднего балла. Добавить возможность вывода фамилий и номеров групп студентов, имеющих оценки, равные только 4 или 5. |
|  | Создайте класс с именем train, содержащую поля: название пункта назначения, номер поезда, время отправления. Ввести данные в массив из пяти элементов типа train, упорядочить элементы по номерам поездов. Добавить возможность вывода информации о поезде, номер которого введен пользователем. Добавить возможность сортировки массив по пункту назначения, причем поезда с одинаковыми пунктами назначения должны быть упорядочены по времени отправления. |
|  | Создать класс машина, имеющий марку (указатель на строку) , число цилиндров, мощность. Определить конструкторы, деструктор и функцию печати. Создать public- производный класс – грузовики, имеющий грузоподъемность кузова. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения марки и грузоподъемности. |

# Литература

1. **Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.
2. **Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.
3. **Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич