Лабораторная работа 2. Разделы документа SDD

Содержание

[Примечание 3](#_Toc90157038)

[Структура проекта клиентского модуля 3](#_Toc90157039)

[UML-диаграмма классов уровня логики приложения 4](#_Toc90157040)

[Особенности реализации и использования классов 5](#_Toc90157041)

[Описание control flow основных частей проекта 8](#_Toc90157042)

[Описание использования и работы функции confirm 8](#_Toc90157043)

[Описание использования и работы функции prompt 9](#_Toc90157044)

[Описание использования и работы функции alert 10](#_Toc90157045)

[Контрольные вопросы + ответы: 11](#_Toc90157046)

# Примечание

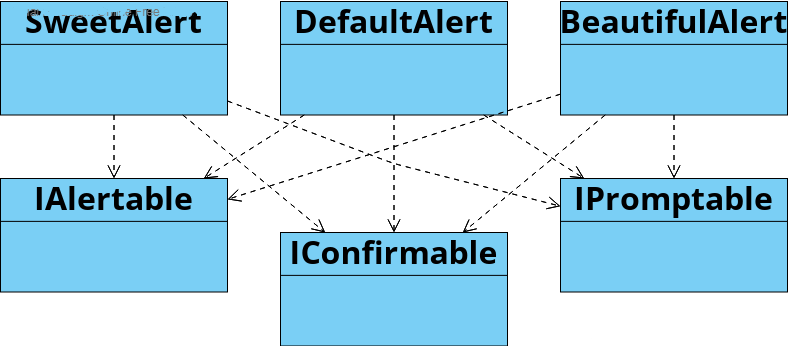
Многие функции, реализующие методы, подобные тем, которые изначально есть в js должны иметь такие же названия. Большая часть переменных должны иметь осмысленные названия(например в случае заголовка - $title, а не $t).

# Структура проекта клиентского модуля

**Логика приложения:**

|  |  |
| --- | --- |
| BeautifulAlert | Вывод в браузер с помощью стилизированного окна, которое можно кастомизировать, передавая различные значения аргументов в функцию/конструктор класса |
| DefaultAlert | Класс, выводящий сообщения с помощью дефолтных функций js |
| SweetAlert | Класс, выводящий сообщения с помощью плагина SweetAlert |
| IAlertable | Интерфейс, который показывает то, что класс способен использовать функцию подобную alert |
| IConfirmable | Интерфейс, который показывает то, что класс способен использовать функцию подобную confirm |
| IPromptable | Интерфейс, который показывает то, что класс способен использовать функцию подобную prompt |

# UML-диаграмма классов уровня логики приложения



# Особенности реализации и использования классов

**DefaultAlert**

Класс для вывода сообщений в браузер.

DefaultAlert должен выводить записи на экран пользователя без какой-либо стилизации. Программист, использующий фреймворк не должен иметь возможности при помощи функций данного класса задавать свой стиль вывода.

Сообщения должны выводиться с помощью функций alert, confirm, prompt.

Alert – простой вывод сообщения.

Confirm – вывод диалогового окна с выбором между «ок» и «cancel»

Prompt - вывод диалогового окна с возможностью ввода текста.

class DefaultAlert implements IAlertable, IConfirmable, IPromptable

{

    function \_\_construct(string $txt, string $dataType = 'html', string $type = 'POST')

    {

        $this->txt = $txt;

        $this->dataType = $dataType;

        $this->type = $type;

    }

    public function alert()

    {

        ?>

        <script>

             let txt = "<?php echo $this->txt; ?>";

             alert(txt);

        </script>

        <?php

    }

    public function confirm(string $url, $responseIsNeeded = false, $trueResponse = 'OK', $falseResponse = 'OK')

    {

    ?>

        <script>

        let txt = "<?php echo $this->txt; ?>";

        let dataType = "<?php echo $this->dataType; ?>";

        let type = "<?php echo $this->type; ?>";

        let url = "<?php echo $url; ?>";

        let responseIsNeeded = "<?php echo $responseIsNeeded; ?>";

        let trueResponse = "<?php echo $trueResponse; ?>";

        let falseResponse = "<?php echo $falseResponse; ?>";

        var confirm = confirm(txt);

        $.ajax({

            url: url,

            dataType: dataType,

            type: type,

            data: "confirm=" + encodeURIComponent(confirm),

        })

        if (responseIsNeeded) {

            if(confirm)

        {

            alert(trueResponse);

        }

        else

        {

            alert(falseResponse);

        }

        }

        </script>

    <?php

    }

    public function prompt(string $url, $responseIsNeeded = false, $response = 'OK')

    {

    ?>

        <script>

        let txt = "<?php echo $this->txt; ?>";

        let dataType = "<?php echo $this->dataType; ?>";

        let type = "<?php echo $this->type; ?>";

        let url = "<?php echo $url; ?>";

        let responseIsNeeded = "<?php echo $responseIsNeeded; ?>";

        let response = "<?php echo $response; ?>";

        var prompt = prompt(txt);

        $.ajax({

            url: url,

            dataType: dataType,

            type: type,

            data: "prompt=" + encodeURIComponent(prompt),

        })

        if (responseIsNeeded) {

        if(prompt)

        {

            alert(response);

        }

        }

        </script>

    <?php

    }

    public function \_\_set($property, $value)

    {

        $this->$property = $value; // устанавливаем значение

    }

}

# Описание control flow основных частей проекта

## Описание использования и работы функции confirm

**вызов функции:**

…

$defaultAlert ->confirm(‘script.php’);

…

**Передача параметров из php в js:**

        let txt = "<?php echo $this->txt; ?>";

        let dataType = "<?php echo $this->dataType; ?>";

        let type = "<?php echo $this->type; ?>";

        let url = "<?php echo $url; ?>";

        let trueResponseIsNeeded = "<?php echo $trueResponseIsNeeded; ?>";

        let falseResponseIsNeeded = "<?php echo $falseResponseIsNeeded; ?>";

        let trueResponse = "<?php echo $trueResponse; ?>";

        let falseResponse = "<?php echo $falseResponse; ?>";

**Вызов confirm:**

        var confirm = confirm(txt);

**использование** **$.ajax:**

        $.ajax({

            url: url,

            dataType: dataType,

            type: type,

            data: "confirm=" + encodeURIComponent(confirm),

        })

**Вывод сообщения, если программист решил, что он нужен(если параметры trueResponseIsNeeded и falseResponseIsNeeded == true ):**

        if(confirm)

        {

           if(trueResponseIsNeeded) alert(trueResponse);

        }

        else

        {

            if(falseResponseIsNeeded) alert(falseResponse);

        }

## Описание использования и работы функции prompt

**вызов функции:**

…

$defaultAlert ->prompt(‘script.php’);

…

**Передача параметров из php в js:**

        let txt = "<?php echo $this->txt; ?>";

        let dataType = "<?php echo $this->dataType; ?>";

        let type = "<?php echo $this->type; ?>";

        let url = "<?php echo $url; ?>";

        let trueResponseIsNeeded = "<?php echo $trueResponseIsNeeded; ?>";

        let falseResponseIsNeeded = "<?php echo $falseResponseIsNeeded; ?>";

        let trueResponse = "<?php echo $trueResponse; ?>";

        let falseResponse = "<?php echo $falseResponse; ?>";

**Вызов prompt:**

        var prompt = prompt(txt);

**использование $.ajax:**

       $.ajax({

            url: url,

            dataType: dataType,

            type: type,

            data: "prompt=" + encodeURIComponent(prompt),

        })

**Вывод сообщения, если программист решил, что он нужен(если параметры trueResponseIsNeeded и falseResponseIsNeeded == true ):**

        if(prompt)

        {

           if(trueResponseIsNeeded) alert(trueResponse);

        }

        else

        {

            if(falseResponseIsNeeded) alert(falseResponse);

        }

## Описание использования и работы функции alert

**Вызов функции:**

**///…**

$defaultAlert->alert();

///…

**Передача текста, предназначенного для функции, из конструктора класса:**

             let txt = "<?php echo $this->txt; ?>";

**Вывод текста с помощью одноимённой функции js:**

             alert(txt);

# Контрольные вопросы + ответы:

1. В чем отличие предварительного проектирования от детального?

Целью **детального проектирования** является полноценный проект, **по** которому может быть создана программа

При предварительном проектировании только разрабатывают его дизайн программного продукта.

1. Перечислите и кратко опишите архитектурные системные паттерны.

Репозиторий

*Описание.* Все совместно используемые подсистемами данные хранятся в центральной базе данных, доступной всем подсистемам. Репозиторий является пассивным элементом, а управление им возложено на подсистемы.

Паттерн Клиент/сервер

*Описание.*Данные и процессы системы распределены между несколькими процессорами. Паттерн имеет три основных компонента: набор автономных серверов (предоставляют сервисы другим подсистемам), набор подсистем — клиентов (которые вызывают сервисы, предоставляемые серверами) и сеть (служит для доступа клиентов к сервисам). Клиенты должны знать имена серверов и сервисов, в то время как серверам не надо знать имена клиентов и их количество. Клиенты получают доступ к сервисам, предоставляемым серверами посредством удаленного вызова процедур.

Паттерн обьектно-ориентированный

Бизнес-логика крайне сложна, имеется множество правил и условий, оговаривающих различные варианты поведения системы.

Паттерн Многоуровневая система (Layers) или Абстрактная машина

*Описание*: В соответствии с паттерном Многоуровневая система структурные элементы системы организуются в отдельные уровни со взаимосвязанными обязанностями таким образом, чтобы на нижнем

3. Перечислите и кратко опишите паттерны управления.

Паттерны централизованного управления

|  |
| --- |
| *Вызов программных процедур осуществляется "сверху - вниз", то есть управление начинается на вершине иерархии процедур и через вызовы передается на нижние уровни иерархии.* |
|  |

**Паттерн Диспетчер**

|  |  |
| --- | --- |
| **Описание** | *Один системный компонент назначается диспетчером и управляет запуском и завершением других процессов системы и координирует эти процессы. Процессы могут протекать параллельно.* |
|  |  |

**Передача сообщений**

|  |  |
| --- | --- |
| **Описание** | В рамках данного паттерна событие представляет собой передачу сообщения всем подсистемам. Любая подсистема, которая обрабатывает данное событие, отвечает на него. |
|  |  |

4. Что такое «связность модуля»? Перечислите и кратко опишите типы связности модуля с указанием значения силы связности для каждого типа.

Связность модуля (cohesion) – внутренняя характеристика модуля, характеризующая меру прочности соединения функциональных и информационных объектов внутри одного модуля.

7 типов связности и их силы:

* Функциональная связность - 10
* Последовательная связность - 9
* Информационная связность - 7
* Процедурная связность - 5
* Временная связность - 3
* Логическая связность - 1
* Связность по совпадению - 0

**Функционально связный модуль** содержит объекты, предназначенные для решения одной единственной задачи. Примерами функционально связанных модулей являются модули проверки орфографии, вычисления заработной платы сотрудника, вычисления логарифма функции.

В **последовательно связном модуле** его объекты охватывают подзадачи, для которых выходные данные одной из подзадач являются входными для другой (открыть файл – прочитать запись – закрыть файл).

**Информационно связный модуль** содержит объекты, использующие одни и те же входные или выходные данные. Так, по ISBN книги, можно узнать ее название, автора и год издания. Эти три процедуры (определить название, определить автора, определить год издания) связаны между собой тем, что все они работают с одним и тем же информационным объектом – ISBN.

**Процедурно связный модуль** – это такой модуль, объекты которого включены в различные (возможно, несвязанные) подзадачи, в которых управление переходит от одной подзадачи к следующей (сделать зарядку, принять душ, позавтракать, одеться, отправится на работу). В отличие от последовательно связанного модуля, в котором осуществляется передача данных, в процедурно связанном модуле выполняется передача управления.

**Модуль с временной связностью** – это такой модуль, в котором объекты модуля привязаны к конкретному промежутку времени. Примером может являться модуль, осуществляющий инициализацию системы. Элементы данного модуля почти не связаны друг с другом за исключением того, что должны выполняться в определенное время.

**Модуль с логической связностью** – это такой модуль, объекты которого содействуют решению одной общей подзадачи, для которой эти объекты отобраны во внешнем по отношению к модулю мире. Так, например, альтернативы: поехать на автомобиле, на метро, на автобусе – являются средством достижения цели: добраться в како-то определенное место, из которых нужно выбрать одну.

**Модуль со связностью по совпадению** содержит объекты, которые слабо связаны друг с другом (сходить в кино, поужинать, посмотреть телевизор, проверить электронную почту).

5. Что такое «сцепление модулей»? Перечислите и кратко опишите типы сцепления модулей.

Сцепление модулей – это мера относительной независимости модулей, которая определяет их читабельность и сохранность.

Типы сцепления + описание:

**Независимое сцепление**возможно, если модули не вызывают друг друга и не обрабатывают одну и ту же информацию.

Модули **сцеплены по данным,**если они имеют общие простые элементы данных, которые передаются от одного модуля к другому как параметры.

(пр, структура – массив, а передается в качестве параметра элемент массива; при этом изменения в структуре данных не повлияют на другой модуль). Модули со сцеплением по данным не имеют общей области данных (глобальных переменных).

Модули **сцеплены по образцу**, если в качестве параметров используются структуры данных (например, в качестве параметра передается массив). Недостаток:оба модуля должны содержать информацию о др модуле(внутренней структуре данных).т.е изменения должны отображ и в др. модуле.

Модули **сцеплены по общей области,** если они разделяют одну и ту же глобальную структуру данных.

Модули **сцеплены по управлению**, если какой-либо из них управляет решениями внутри другого с помощью передачи флагов, переключателей и т.д.

Модуль **сцеплен по внешним ссылкам**, если у него есть доступ к данным другого модуля через внешнюю точку входа.

Модули **сцеплены по кодам**, если коды их команд объединены друг с другом. , использ общий участок памяти.

Если модули косвенно обращаются друг к другу (ч/з другие модули), то между ними также существует сцепление.

6. Опишите следующие фундаментальные паттерны проектирования: делегирование, неизменяемый объект, интерфейс, MVC.

**Делегирование** — это передача задач или компетенций от руководителя к подчиненному.

В программировании неизменяемым называется объект, состояние которого не может быть изменено после создания.

Интерфе́йс — программная/синтаксическая структура, определяющая отношение между объектами, которые разделяют определённое поведенческое множество и не связаны никак иначе.

Model-View-Controller — схема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.