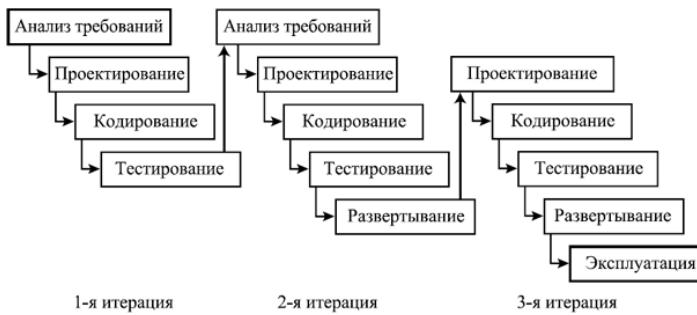


Билет №1

1. Ітеративний процес розробки. Переваги та недоліки



Ітеративний подхід в розробці п.о. — це выполнение работ паралельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы. Проект при этом подходит в каждой фазе развития проходит повторяющийся цикл: Планирование — Реализация — Проверка — Оценка

Преимущества итеративного подхода:

- снижение воздействия серьёзных **рисков** на ранних стадиях проекта, что ведет к минимизации затрат на их устранение;
- организация эффективной обратной связи проектной команды с заказчиком и создание продукта, реально отвечающего его потребностям;
- акцент усилий на наиболее важные и критичные направления проекта;
- непрерывное итеративное тестирование, позволяющее оценить успешность всего проекта в целом;
- раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта;
- более равномерная загрузка участников проекта;
- реальная оценка текущего состояния проекта и, как следствие, большая уверенность заказчиков и непосредственных участников в его успешном завершении.
- затраты распределяются по всему проекту, а не группируются в его конце

Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.

2. Арх. Шаблон «Тонкий клиент»

Тонкий клиент – это арх. шаблон, который предоставляет клиенту (браузеру) ограниченные возможности по управлению конфигурацией. При этом вся бизнес-логика (реализация предметной области в информационной системе) выполняется на сервере в процессе обработки запроса на получение страницы, полностью сгенерированной на сервере.



В основе поведения этого архитектурного шаблона лежит следующий принцип: бизнес-логика используется только в ответ на запрос Web-страницы клиентом. Клиент взаимодействует с системой, используя протокол *HTTP* для получения страниц с Web-сервера. Если запрашиваемая

страница является файлом *HTML* файловой системы Web-сервера, то последний просто извлекает страницу и пересыпает ее соответствующему клиенту.

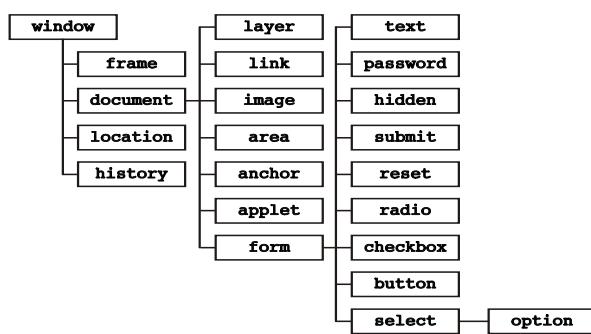
Если страница содержит сценарий, т.е. интерпретируемый код, который должна быть обработан до отправки клиенту, то все необходимые для этого действия выполняются сервером приложения.

Преимущества:

1. Возможность использования старых комп. (не нагружается ресурсы клиентского комп)
2. Возможность централизованного упр (часто использ в интернет-магаз с больший кол польз)

Недостатки:

- 1.Высокие требования к серверному обесп
- 2.Бедный интерф пользователя



в виде узла дерева.

```
x = prompt("Введите Ваше имя", "Имя");
document.getElementById("dialog").style.visibility = "visible";
document.getElementById("name").innerHTML = x;
document.getElementById("header").innerHTML = "Диалог";
```

3. Пример использования модели DOM

DOM, или объектная модель документов (Document Object Model), является способом моделирования HTML-документов. В рамках этой модели обеспечивается возможность доступа, навигации и манипулирования HTML-документами.

Объектная модель документов представляет документ в виде дерева. Структура такого дерева полностью описывает весь HTML-документ, представляя каждый тег и его текстовое содержимое

Каскадная модель (англ. *waterfall model*) — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки. В качестве источника названия часто указывают статью, опубликованную У. У. Ройсом (*W. W. Royce*) в 1970 году; при том, что сам Ройс использовал итеративную модель разработки.

Содержание модели

В 1970 году в своей статье Ройс описал в виде концепции то, что сейчас принято называть «каскадная модель», и обсуждал недостатки этой модели. Там же он показал как эта модель может быть доработана до итеративной модели.

В оригинальной каскадной модели Ройса, следующие фазы шли в таком порядке:

1. Определение требований
2. Проектирование
3. Конструирование (также «реализация» либо «кодирование»)
4. Воплощение
5. Тестирование и отладка (также «верификация»)
6. Инсталляция
7. Поддержка

Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.

Тем самым, каскадная модель подразумевает, что переход от одной фазы разработки к другой происходит только после полного и успешного завершения предыдущей фазы, и что переходов назад либо вперёд или перекрытия фаз — не происходит.

Тем не менее, существуют модифицированные каскадные модели (включая модель самого Ройса), имеющие небольшие или даже значительные вариации описанного процесса.

Критика каскадной модели и гибридные методологические решения

Методику «Каскадная модель» довольно часто критикуют за недостаточную гибкость и объявление самоцелью формальное управление проектом в ущерб срокам, стоимости и качеству.^[1] Тем не менее, при управлении большими проектами формализация часто являлась очень большой ценностью, так как могла кардинально снизить многие риски проекта и сделать его более прозрачным. Поэтому даже в PMBOK 3-ей версии формально была закреплена только методика «каскадной модели» и не были предложены альтернативные варианты, известные как итеративное ведение проектов.

Начиная с PMBOK 4-ой версии удалось достичь компромисса между методологами, приверженными формальному и поступательному управлению проектом, с методологами, делающими ставку на гибкие итеративные методы.^[2] Таким образом, начиная с 2009 года, формально Институтом Проектного Менеджмента (PMI) предлагается как стандарт гибридный вариант методологии управления проектами, сочетающий в себе как плюсы от методики «Водопада», так и достижения итеративных методологов.

Каскадный подход

Если не углубляться в детали, то каскадные методологии разработки ПО подразумевают, что разработка ПО делится на фазы, каждая из которых характеризуется своим набором работ. Сначала происходит выявление всех требований к проекту и их анализ. Затем проектная группа приступает к проектированию системы (чаще всего сверху вниз, разбив создаваемую систему на подсистемы и далее детализируя их до уровня программных процедур и функций). После этого начинаются разработка кода и модульное тестирование. Затем наступает очередь сборки и системного тестирования. И так далее — вплоть до передачи системы заказчику.

Преимуществами каскадного подхода считаются: минимальный объем переработок написанного кода, возможность тщательно спроектировать архитектуру системы и однократное тестирование системы.

Состав команды

При каскадном подходе в проекте от начала и до конца принимает участие разве что менеджер проекта. Аналитики, зафиксировав требования, уступают место разработчикам, а те, в свою очередь, — специалистам по тестированию. Конечно, ведущий аналитик обычно продолжает присматривать за проектом, давая необходимые пояснения архитектору и программистам, а те, в свою очередь, исправляют обнаруженные тестировщиками дефекты. Но это уже, как правило, неполная занятость. Да и занимаются этим не все участовавшие в проекте на предыдущей фазе.

При итерационной разработке команда проекта оказывается значительно более стабильной. Она может сохраняться в течение нескольких итераций, активно изменяясь лишь в самом начале и в самом конце проекта.

Толстый или **Rich-клиент**^[1] в архитектуре клиент-сервер — это [приложение](#), обеспечивающее (в противовес [тонкому клиенту](#)) расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь [хранилищем данных](#), а вся работа по обработке и представлению этих данных переносится на машину клиента.

ДОСТОИНСТВА

Толстый клиент обладает широким функционалом в отличие от [тонкого](#).

- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Высокое быстродействие (зависит от аппаратных средств клиента)

Недостатки

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.
- Наличие бизнес-логики

Объекты в JavaScript

Во многих статьях встречается фраза «В JavaScript — всё объект». Технически это не совсем верно, однако производит должное впечатление на новичков :)

Действительно, многое в языке является объектом, и даже то, что объектом не является, может обладать некоторыми его возможностями.

Важно понимать, что слово «объект» употребляется здесь не в смысле «объект некоторого класса». Объект в JavaScript — это в первую очередь просто коллекция свойств (если

кому проще, может называть это ассоциативным массивом или списком), состоящая из пар ключ-значение. Причем ключом может быть только строка (даже у элементов массива), а вот значением — любой тип данных из перечисленных ниже.

Итак, в JavaScript есть 6 базовых типов данных — это Undefined (обозначающий отсутствие значения), Null, Boolean (булев тип), String (строка), Number (число) и Object (объект).

При этом первые 5 являются *примитивными* типами данных, а Object — нет. Кроме того, условно можно считать, что у типа Object есть «подтипы»: массив (Array), функция (Function), регулярное выражение (RegExp) и другие.

Это несколько упрощенное описание, но на практике обычно достаточно.

Кроме того, примитивные типы String, Number и Boolean определенным образом связаны с не-примитивными «подтипами» Object: String, Number и Boolean соответственно.

Это означает, что строку 'Hello, world', например, можно создать и как примитивное значение, и как объект типа String.

Если вкратце, то это сделано для того, чтобы программист мог и в работе с примитивными значениями использовать методы и свойства, как будто это объекты. А подробнее об этом можно будет прочитать в соответствующем разделе данной статьи.

JavaScript предоставляет разработчикам возможность создавать объекты и работать с ними. Для этого существуют следующие приёмы:

- Оператор new
- Литеральная нотация
- Конструкторы объектов
- Ассоциативные массивы

Используем оператор new

Это, наверное, самый легкий способ создания объекта. Вы просто создаете имя объекта и приравниваете его к новому объекту Javascript.

```
// Создаем наш объект
var MyObject = new Object();
// Переменные
MyObject.id = 5; // Число
MyObject.name = "Sample"; // Стока
// Функции
MyObject.getName = function()
{
    return this.name;
}
```

Минус данного способа заключается в том, что вы можете работать только с одним вновь созданным объектом.

```
//Используем наш объект
alert(MyObject.getName());
```

Литеральная нотация

Литеральная нотация является несколько непривычным способом определения новых объектов, но достаточно легким для понимания. Литеральная нотация работает с версии Javascript 1.3.

```
//Создаем наш объект с использованием литеральной нотации
MyObject = {
    id : 1,
    name : "Sample",
    boolval : true,
    getName : function()
    {
        return this.name;
    }
}
```

Как видите, это довольно просто.

```
Объект = {
    идентификатор : значение,
    ...
}
```

И пример использования:

```
alert(MyObject.getName());
```

Конструкторы объектов

Конструкторы объектов — это мощное средство для создания объектов, которые можно использовать неоднократно. Конструктор объекта — это, по сути, обычная функция Javascript, которой так же можно передавать различные параметры.

```
function MyObject(id, name)
{
}

}
```

Только что мы написали конструктор. С помощью него мы и будем создавать наш объект.

```
var MyFirstObjectInstance = new MyObject(5, "Sample");
var MySecondObjectInstace = new MyObject(12, "Othe Sample");
```

Таким образом мы создали различные экземпляры объекта. Теперь мы можем работать отдельно с каждым экземпляром объекта MyObject, не боясь того, что, изменяя свойства одного экземпляра, мы затронем свойства другого экземпляра.

Как и в ООП, у MyObject могут быть методы и различные свойства. Свойствам можно присвоить значения по умолчанию, либо значения, переданные пользователем в конструкторе объекта.

```
function MyObject(id, name)
{
    //Значения переданные пользователем
    this._id = id;
    this._name = name;
    //Значение по умолчанию
    this.defaultValue = "MyDefaultValue";
}
```

Аналогичным образом мы можем создавать и функции.

```
function MyObject(id, name)
{
    this._id = id;
    this._name = name;
    this.defaultValue = "MyDefaultValue";

    //Получение текущего значения
    this.getDefaultValue = function()
    {
        return this.defaultValue;
    }

    //Установка нового значения
    this.setDefaultValue = function(newvalue)
```

```

{
    this.defaultvalue = newvalue;
}

//Произвольная функция
this.sum = function(a, b)
{
    return (a+b);
}
}

```

Ассоциативные массивы

Подобный метод будет полезен упорядочивания большого числа однотипных объектов.

```

var MyObject = new Number();
MyObject["id"] = 5;
MyObject["name"] = "SampleName";

```

Для обхода таких объектов можно использовать такой цикл:

```

for (MyElement in MyObject)
{
    //Код обхода
    //В MyElement - идентификатор записи
    //В MyObject[MyElement] - содержание записи
}

```

Каскадная модель ([англ. waterfall model](#)) — [модель](#) процесса [разработки программного обеспечения](#), в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки. В качестве источника названия часто указывают статью, опубликованную У. У. Ройсом (*W. W. Royce*) в [1970 году](#); при том, что сам Ройс использовал [итеративную модель разработки](#).

Содержание модели

В 1970 году в своей статье Ройс описал в виде концепции то, что сейчас принято называть «каскадная модель», и обсуждал недостатки этой модели. Там же он показал как эта модель может быть доработана до итеративной модели.

В оригинальной каскадной модели Ройса, следующие фазы шли в таком порядке:

8. Определение требований
9. Проектирование

10. Конструирование (также «реализация» либо «кодирование»)
11. Воплощение
12. Тестирование и отладка (также «верификация»)
13. Инсталляция
14. Поддержка

Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.

Тем самым, каскадная модель подразумевает, что переход от одной фазы разработки к другой происходит только после полного и успешного завершения предыдущей фазы, и что переходов назад либо вперёд или перекрытия фаз — не происходит.

Тем не менее, существуют модифицированные каскадные модели (включая модель самого Ройса), имеющие небольшие или даже значительные вариации описанного процесса.

Критика каскадной модели и гибридные методологические решения

Методику «Каскадная модель» довольно часто критикуют за недостаточную гибкость и объявление самоцелью формальное управление проектом в ущерб срокам, стоимости и качеству.^[1] Тем не менее, при управлении большими проектами формализация часто являлась очень большой ценностью, так как могла кардинально снизить многие [риски](#) проекта и сделать его более [прозрачным](#). Поэтому даже в [PMBOK](#) 3-ей версии формально была закреплена только методика «каскадной модели» и не были предложены альтернативные варианты, известные как [итеративное ведение проектов](#).

Начиная с [PMBOK](#) 4-ой версии удалось достичь компромисса между [методологиями](#), приверженными формальному и поступательному управлению проектом, с методологиями, делающими ставку на [гибкие итеративные методы](#).^[2] Таким образом, начиная с 2009 года, формально [Институтом Проектного Менеджмента \(PMI\)](#) предлагается как стандарт гибридный вариант методологии управления проектами, сочетающий в себе как плюсы от методики «Водопада», так и достижения итеративных методологов.

Каскадный подход

Если не углубляться в детали, то каскадные методологии разработки ПО подразумевают, что разработка ПО делится на фазы, каждая из которых характеризуется своим набором работ. Сначала происходит выявление всех требований к проекту и их анализ. Затем проектная группа приступает к проектированию системы (чаще всего сверху вниз, разбив создаваемую систему на подсистемы и далее детализируя их до уровня программных процедур и функций). После этого начинаются разработка кода и модульное тестирование. Затем наступает очередь сборки и системного тестирования. И так далее — вплоть до передачи системы заказчику.

Преимуществами каскадного подхода считаются: минимальный объем переработок написанного кода, возможность тщательно спроектировать архитектуру системы и однократное тестирование системы.

Состав команды

При каскадном подходе в проекте от начала и до конца принимает участие разве что менеджер проекта. Аналитики, зафиксировав требования, уступают место разработчикам, а те, в свою очередь, — специалистам по тестированию. Конечно, ведущий аналитик обычно продолжает присматривать за проектом, давая необходимые пояснения архитектору и

программистам, а те, в свою очередь, исправляют обнаруженные тестировщиками дефекты. Но это уже, как правило, неполная занятость. Да и занимаются этим не все участвовавшие в проекте на предыдущей фазе.

При итерационной разработке команда проекта оказывается значительно более стабильной. Она может сохраняться в течение нескольких итераций, активно изменяясь лишь в самом начале и в самом конце проекта.

Толстый или **Rich-клиент**^[1] в архитектуре клиент-сервер — это [приложение](#), обеспечивающее (в противовес [тонкому клиенту](#)) расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь [хранилищем данных](#), а вся работа по обработке и представлению этих данных переносится на машину клиента.

Достоинства

Толстый клиент обладает широким функционалом в отличие от [тонкого](#).

- Режим многопользовательской работы.
- Предоставляет возможность работы даже при обрывах связи с сервером.
- Высокое быстродействие (зависит от аппаратных средств клиента)

Недостатки

- Большой размер дистрибутива.
- Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
- При работе с ним возникают проблемы с удаленным доступом к данным.
- Довольно сложный процесс установки и настройки.
- Сложность обновления и связанная с ней неактуальность данных.
- Наличие бизнес-логики

Объекты в JavaScript

Во многих статьях встречается фраза «В JavaScript — всё объект». Технически это не совсем верно, однако производит должное впечатление на новичков :)

Действительно, многое в языке является объектом, и даже то, что объектом не является, может обладать некоторыми его возможностями.

Важно понимать, что слово «объект» употребляется здесь не в смысле «объект некоторого класса». Объект в JavaScript — это в первую очередь просто коллекция свойств (если кому проще, может называть это ассоциативным массивом или списком), состоящая из пар ключ-значение. Причем ключом может быть только строка (даже у элементов массива), а вот значением — любой тип данных из перечисленных ниже.

Итак, в JavaScript есть 6 базовых типов данных — это Undefined (обозначающий отсутствие значения), Null, Boolean (булев тип), String (строка), Number (число) и Object (объект).

При этом первые 5 являются *примитивными* типами данных, а Object — нет. Кроме того, условно можно считать, что у типа Object есть «подтипы»: массив (Array), функция (Function), регулярное выражение (RegExp) и другие.

Это несколько упрощенное описание, но на практике обычно достаточно.

Кроме того, примитивные типы String, Number и Boolean определенным образом связаны с не-примитивными «подтипами» Object: String, Number и Boolean соответственно. Это означает, что строку 'Hello, world', например, можно создать и как примитивное значение, и как объект типа String. Если вкратце, то это сделано для того, чтобы программист мог и в работе с примитивными значениями использовать методы и свойства, как будто это объекты. А подробнее об этом можно будет прочитать в соответствующем разделе данной статьи.

JavaScript предоставляет разработчикам возможность создавать объекты и работать с ними. Для этого существуют следующие приёмы:

- Оператор new
- Литеральная нотация
- Конструкторы объектов
- Ассоциативные массивы

Используем оператор new

Это, наверное, самый легкий способ создания объекта. Вы просто создаете имя объекта и приравниваете его к новому объекту Javascript.

```
//Создаем наш объект
var MyObject = new Object();
//Переменные
MyObject.id = 5; //Число
MyObject.name = "Sample"; //Строка
//Функции
MyObject.getName = function()
{
    return this.name;
}
```

Минус данного способа заключается в том, что вы можете работать только с одним вновь созданным объектом.

```
//Используем наш объект
alert(MyObject.getName());
```

Литеральная нотация

Литеральная нотация является несколько непривычным способом определения новых

объектов, но достаточно легким для понимания. Литеральная нотация работает с версии Javascript 1.3.

```
//Создаем наш объект с использованием литеральной нотации
MyObject = {
    id : 1,
    name : "Sample",
    boolval : true,
    getName : function()
    {
        return this.name;
    }
}
```

Как видите, это довольно просто.

```
Объект = {
    идентификатор : значение,
    ...
}
```

И пример использования:

```
alert(MyObject.getName());
```

Конструкторы объектов

Конструкторы объектов — это мощное средство для создания объектов, которые можно использовать неоднократно. Конструктор объекта — это, по сути, обычная функция Javascript, которой так же можно передавать различные параметры.

```
function MyObject(id, name)
{
}
```

Только что мы написали конструктор. С помощью него мы и будем создавать наш объект.

```
var MyFirstObjectInstance = new MyObject(5, "Sample");
var MySecondObjectInstace = new MyObject(12, "Othe Sample");
```

Таким образом мы создали различные экземпляры объекта. Теперь мы можем работать отдельно с каждым экземпляром объекта MyObject, не боясь того, что, изменяя свойства одного экземпляра, мы затронем свойства другого экземпляра.

Как и в ООП, у MyObject могут быть методы и различные свойства. Свойствам можно присвоить значения по умолчанию, либо значения, переданные пользователем в конструкторе объекта.

```
function MyObject(id, name)
{
    //Значения переданные пользователем
    this._id = id;
    this._name = name;
    //Значение по умолчанию
    this.defaultValue = "MyDefaultValue";
}
```

Аналогичным образом мы можем создавать и функции.

```
function MyObject(id, name)
{
    this._id = id;
    this._name = name;
    this.defaultValue = "MyDefaultValue";

    //Получение текущего значения
    this.getDefaultValue = function()
    {
        return this.defaultValue;
    }

    //Установка нового значения
    this.setValue = function(newvalue)
    {
        this.defaultValue = newvalue;
    }

    //Произвольная функция
    this.sum = function(a, b)
    {
        return (a+b);
    }
}
```

Ассоциативные массивы

Подобный метод будет полезен упорядочивания большого числа однотипных объектов.

```
var MyObject = new Number();
MyObject["id"] = 5;
MyObject["name"] = "SampleName";
```

Для обхода таких объектов можно использовать такой цикл:

```
for (MyElement in MyObject)
{
    //Код обхода
    //В MyElement - идентификатор записи
    //В MyObject[MyElement] - содержание записи
}
```

Билет 6

1. Шаблон Controller. Призначення.

Шаблон - це іменована пара "проблема / рішення", що містить рекомендації для застосування в різних конкретних ситуаціях, яку можна використовувати в різних контекстах

Шаблони не містять нових ідей

Вони покликані систематизувати існуючі знання, ідіоми та принципи

Шаблони мають імена

☒ Дозволяє зафіксувати поняття в пам'яті

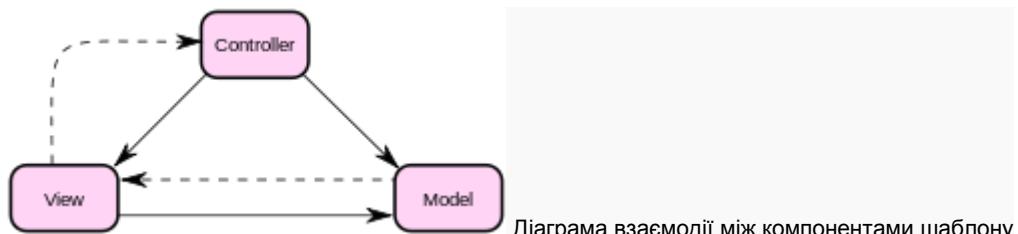
☒ Полегшує спілкування

Шаблон Controller - [архітектурний шаблон](#), який використовується під час проєктування та розробки [програмного забезпечення](#).

Цей шаблон поділяє систему на три частини: [модель даних](#), вигляд даних та [керування](#).

Застосовується для відокремлення даних (модель) від [інтерфейсу користувача](#) (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах призводить до певної впорядкованості їх структури і робить їх зрозумілішими завдяки зменшенню складності.



Діаграма взаємодії між компонентами шаблону

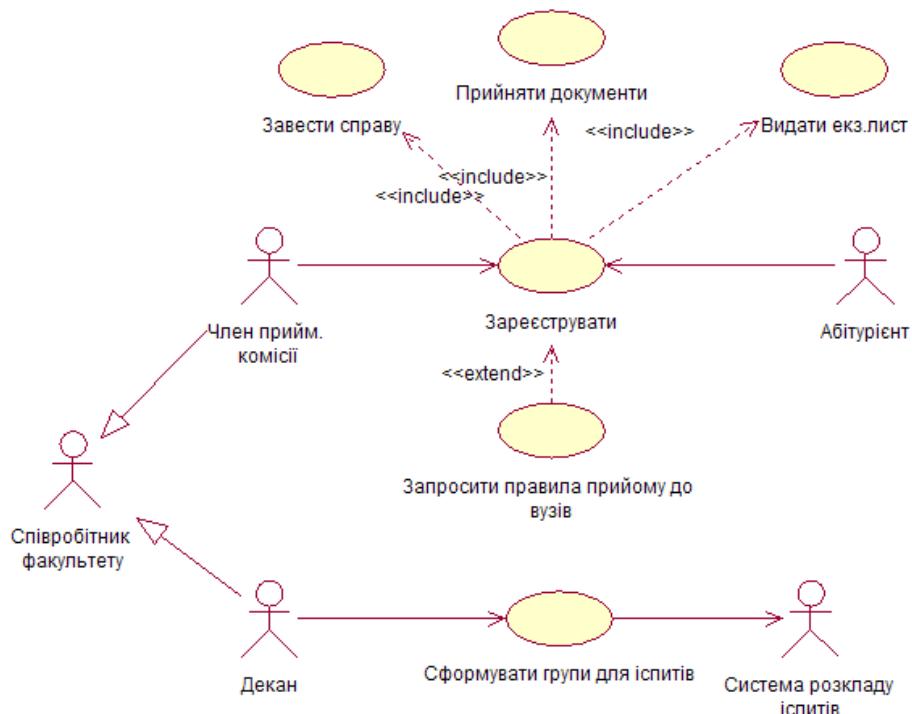
Типы контроллеров

- зовнішній контролер
 - 1. представляє всю "систему", пристрій або підсистему
 - 2. забезпечує головну точку виклику всіх служб з інтерфейсу користувача та обігу до решти верствам
- контролер прецеденту
 - 1. вводиться в тому випадку, якщо існуючий контроллер занадто "роздувається" при покладанні на нього додаткових обов'язків

1. Статичні та динамічні представлення програмної системи. Приклади. Призначення

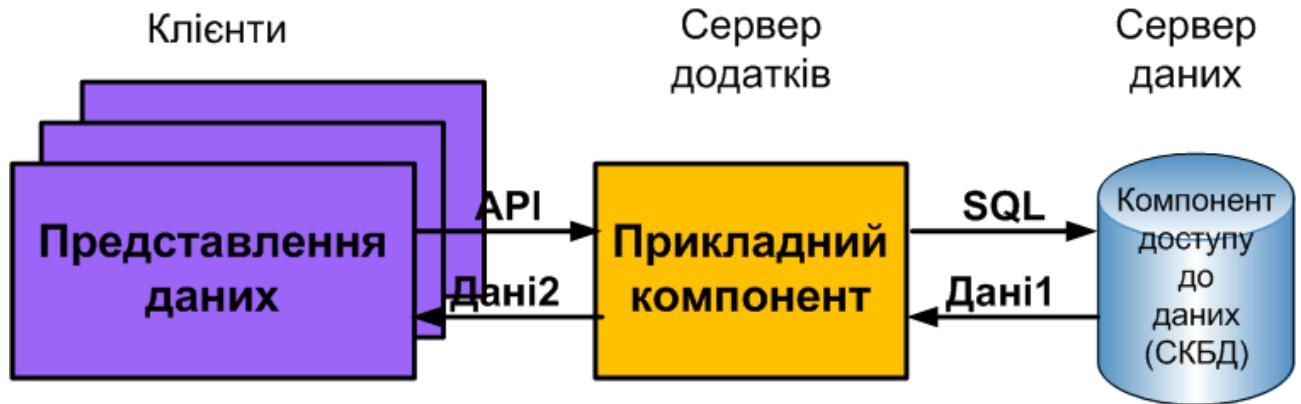
В UML **інтегровані** різноманітні відомі засоби візуального моделювання, які добре зарекомендували себе на практиці, зокрема, забезпечується можливість опису двох визначальних видів об'єктних моделей:

- **структурних** (або **статичних**) **моделей** – описується структура сутностей системи, включаючи класи, інтерфейси, відношення, атрибути;
- **моделей поведінки** (або **динамічних моделей**) – описується поведінка (функціонування) об'єктів системи, включаючи методи, взаємодію, процес зміни станів окремих компонент чи всієї системи.



Тип представлень стосовно найвищого рівня абстракції	Представлення, що підтримуються UML	Діаграми	Основні концепції
Представлення структури	Представлення статичне	Діаграма класів	Клас, асоціація, узагальнення, залежність, реалізація, інтерфейс
	Представлення використання	Діаграма прецедентів	Прецедент, діяч, асоціація, узагальнення, розширення, включення
	Представлення програмної реалізації	Діаграма компонентів	Компонент, інтерфейс, залежність, реалізація
	Представлення розгортання	Діаграма розгортання	Вузол, компонент, залежність, розташування
Представлення (динамічної) поведінки	Представлення скінченим автомatem	Діаграма станів	Стан, подія, перехід, дія
	Представлення діяльності	Діаграма діяльності	Стан, діяльність, завершення переходу, розгалуження, злиття
	Представлення взаємодії	Діаграма послідовності Діаграма кооперації	Взаємодія, об'єкт, повідомлення, активізація Кооперація, взаємодія, роль у кооперації, повідомлення
	Представлення управління моделями	Діаграма класів	Пакет, підсистема, модель

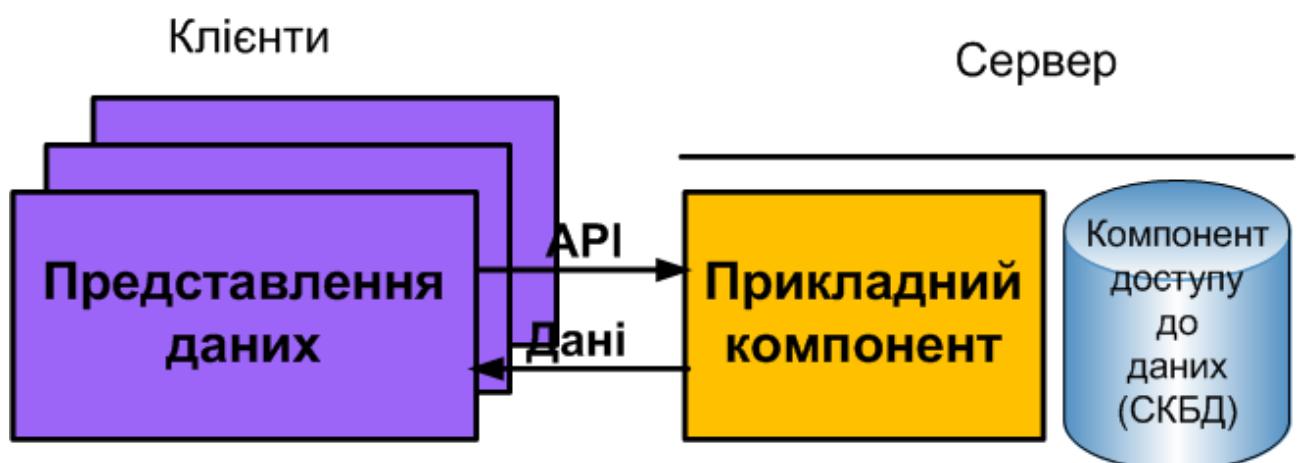
1. Проілюструвати взаємодію з сервером бази даних засобами серверу додатків.
Щоб рознести вимоги до обчислювальних ресурсів сервера у відношенні швидкодії і пам'яті за різними машинам, використовується *модель сервера* - модель зберігає сильні сторони DBS-моделі.



Використання цієї моделі дозволяє розвантажити робочі станції, тобто перейти до «тонких» клієнтів.

Особливості AS-моделі:

- перенесення прикладного компонента АІС на спеціалізований сервер;
- на клієнтських машинах - тільки інтерфейсна частина системи;
- виклики функцій обробки даних спрямовуються на сервер додатків;
- низькорівневі операції з даними виконує SQL-сервер.



Билет 7

Лист N 9

① Що робить інженерний підсумок Офісу зважу та буде зробити

Інженерний підсумок в разіборі ПО - це високий рівень паралельної інженерної аналітичної оптимізації результатів і коригуванням посередником оточувальних роботів. Побудова буде високою у винесеній. Кількість коригувальних розмірників високими (наприклад 1000, або 4 тисяч), але результативність їх менша. Кожна інженерія використовує свої власні етапи: аналіз, висновки, пропонування, реалізація і застосування мезонівальника, інженерія і супроводження роботів системи.

Програми для інженерування підсумку в складі: програми розрахунку кривих кінцевих точок: Підсумування - Розрахування - Таблиця - Офіс.

Підсумок інженерування підсумку

- зменшення погрешності середніх підсумків за рахунок сполучення що веде до підвищення точності підсумків за їх збільшення;
- зменшення криволінійного фасонування підсумків кривими кінцевих точок: інженерування підсумку, реалізація та застосування цього методу;
- зменшення залежності від найменших точок і криволінійного кривої.

Комп'ютерне відображення інженерування підсумку разом з RUP (Rational Unified Process) - універсальними програмами розробки.

Деяльність інженерування може використовувати:

- пакети на функціональну базову використання при розробці систем;
- комп'ютерні пакети із функціональною використанням: числовим та фізичними вимірюваннями в навчанні та експлуатації, при яких використовуються комп'ютерні системи операції від варіант до варіант;
- пакети розробки як інженерування криволіній, то юнітів використовується як пакети: інженерування підсумків, пакети: висновки та обсяги, висновки та обсяги. Універсальні програми висновка комп'ютера разом з пакетами при високому рівні функціональної ефективності використовують.

Ці фази: "Начало" (Initial, Inception), "Розробка" (Planning, Planning), "Конструювання" (Конструювання, Construction), "Розробка" (Designing, Design).

③ Регуляри вирази між Javascript.

Регулярні вирази - це додаткові засоби написання і звільнення від коду з мінімальною кількістю логіки та коду. Ідея регулярного виразу - зробити їх відносно відкритими для підтримки нових символів, які будуть надані тензором. Ідея є та, що співставлення з символами і символічними виразами виконується. Регулярні вирази використовуються для підтримки нових методів якщо не в здатності інтерпретатора для підтримки як підтримки.

Способи використання регулярних виразів:

- Лобія форма заміни: var reg = new RegExp(регулярний вираз);
- Генерата форма заміни(інтерпретаторна форма): var reg = /re вираз/

Атрибути:

- pattern - змінний параметр (під час реалізації)
- flags - змінний параметр як флаги
- g - відповідає кількості замін
- i - не реагує на великі малі літери
- n - інтерпретаторний параметр

Якщо вираз використовується як змінний, то використовується вираз, який використовується в змінній.

Методи:

Метод	Опис
exec	використовується як обєкт RegExp, що повертає відповідь на питання. Регулярний вираз в змінній
test	використовується як обєкт RegExp, що повертає результат як true, якщо відповідь відносно відповідності регулярного виразу в змінній
match	використовується як обєкт RegExp, що повертає відповідь в змінній
search	використовується як обєкт RegExp, що повертає відповідь в змінній
replace	використовується як обєкт RegExp, що повертає нову строку String
split	використовується як обєкт RegExp, що повертає масив на окремі підрозділи

Функція для перевірки чи є номер:

```

function isNum(phoneNum) {
    var ok = num.search(/\d{3}-\d{4}/);
    if (ok == 0)
        return true;
    else
        return false;
}

```

Приклад:

```

var tel = tel.replace(/\d{3}-\d{4}\d{3}"/, "$1-$2-$3");
tel = tel.replace(/\d{3}-\d{4}\d{3}"/, "($1)-$2-$3");
tel = tel.replace(/\d{3}-\d{4}\d{3}"/, "($1)-$2-$3");

```

Приклад:

Вираз	Опис
444-9999	Відповідає
444-1234	Не відповідає
444-2345	Не відповідає

② Розгляніть обєкти якіми реалізована та структурувана DOM (Document Object Model) - об'єкт якого реалізується принциповою прокладеною інтерфейсу та роботи з структурованим документом. Розгляніть як реалізована концепція W3C. З погляду ООП DOM виглядає як клас, якимо то атрибути чи методи є лише у звичайних реалізаціях та крізь інтерфейс які не є реальними. Все це призводить до того що погані мовчання використання DOM.

Програма дозволяє на функціональному рівні реалізувати принципи, які є відповідними для реального DOM.

Через те що структура документа прокладеною та вже відповідає, можна дізнатися будь-якого елемента якщо звернутися до його відповідного методу. Важливо зазначити що відповідно до принципу "Документу не відповідає" він має зберегти у собі інформацію про походження та власність якогось елемента.

Важливо зазначити що відповідно до принципу "Документу не відповідає" він має зберегти у собі інформацію про походження та власність якогось елемента.

- Документ (Document) - реальний зразок, представлений вимогами HTML

```
<document type="text/html" encoding="UTF-8">
<html>
  <head>
    <title> Розклад на ІІІ </title>
  </head>
  <body>
    <h1> ІІІ курсантка </h1>
    <h2> Мат. дисципліни </h2>
  </body>
</html>
```

Документ (Document) - реальний зразок, представлений вимогами HTML

Схема дерево структури DOM

```

graph TD
    Document[Document] --> HTML[HTML]
    HTML --> Head[Head]
    HTML --> Body[Body]
    Head --> Title[Title]
    Body --> H1[H1]
    Body --> H2[H2]
    H1 --> Text1[Text]
    H2 --> Text2[Text]
    
```

Документ (Document) - реальний зразок, представлений вимогами HTML

Документ (Document) - реальний зразок, представлений вимогами HTML

ЕКЗАМЕНАЦІЙНИЙ БЛЄТ № 8

- Шаблони High Cohesion та Low Coupling. Їх місце у багаторівневій архітектурі.

Низька зв'язаність (Low Coupling) і Високе зачеплення (High Cohesion)

Мабуть в будь-якій літературі з об'єктно-орієнтованого проектування зустрічаються ці два поняття. Вважається, що будь-яка спроектована система, повинна задовольняти принципам низькою зв'язності і високого зачеплення модулів. Відповідність даним шаблонами дозволяє легко модифікувати і супроводжувати програмний код а також підвищуює ступінь його повторного використання.

Розглянемо поняття міри зв'язності модулів і заходи зачеплення модуля. Міра зв'язності модулів визначається кількістю інформації яку має один модуль про природу іншого. У свою чергу, міра зачеплення модуля визначається ступенем сфокусованності його обов'язків.

Варто відзначити, що існують методології, згідно з якими заходи зв'язності і зачеплення можна оцінити за шкалою від 1 до 10 для конкретного випадку. Однак, в рамках даної статті вони на розглядаються.

Прикладом хорошого дизайну системи може служити набір утиліт GNU Binutils для Linux. У якому, кожна утиліта (якщо її розглядати як модуль) виконує лише мінімальні обов'язки (високе зачеплення) і майже нічого не знає про природу інших утиліт (низька зв'язність), у зв'язку з чим може бути легко замінена на аналог в деякому варіанті використання.

.....

High Cohesion — это принцип, который задаёт свойство сильного зацепления *внутри* подсистемы. Классы (подсистемы) таким образом получаются сфокусированными, управляемыми и понятными. Зацепление (*cohesion*) (или более точно, функциональное зацепление) — это мера связности и сфокусированности обязанностей класса. Считается, что объект (подсистема) обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет огромных объемов работы.

Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению следующих проблем:

- Трудность понимания.
- Сложность при повторном использовании.
- Сложность поддержки.
- Ненадежность, постоянная подверженность изменениям.

Классы с низкой степенью зацепления, как правило, являются слишком «абстрактными» или выполняют обязанности, которые можно легко распределить между другими объектами.

.....

Low Coupling — это принцип, который позволяет распределить обязанности между объектами таким образом, чтобы степень связности между системами оставалась низкой. Степень связности (*coupling*) — это мера, определяющая, насколько жестко один элемент связан с другими элементами, либо каким количеством данных о других элементах он обладает. Элемент с низкой степенью связности (или слабым связыванием) зависит от не очень большого числа других элементов и имеет следующие свойства:

- Малое число зависимостей между классами (подсистемами).
- Слабая зависимость одного класса (подсистемы) от изменений в другом классе (подсистеме).
- Высокая степень повторного использования подсистем.

2. Призначення та взаємозв'язок Web-сервера та серверу додатків.

Сервер приложений — это программная платформа ([software framework](#)), предназначенная для эффективного выполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API ([Интерфейс прикладного программирования](#)), который определен самой платформой.

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Веб-сервером называют как [программное обеспечение](#), выполняющее функции веб-сервера, так и непосредственно [компьютер](#) (см.:[Сервер \(аппаратное обеспечение\)](#)), на котором это программное обеспечение работает.

Веб-сервер - довольно узкое понятие, обозначающее программу, отдающую данные клиентам по протоколу [HTTP](#).

Сервер приложений - это, грубо говоря, программа, позволяющая централизованно на стороне сервера хранить и выполнять приложения, управляя при этом распределением нагрузки, обеспечивая работу с БД и т.д. Если сервер приложений работает с клиентами по протоколу [HTTP](#), то он одновременно [является](#) и [веб-сервером](#).

3. Призначення суперглобальних масивів PHP.

Суперглобальные массивы - это встроенные переменные, которые всегда доступны во всех областях видимости.

Некоторые предопределённые переменные в PHP являются "суперглобальными", что означает, что они доступны в любом месте скрипта. Нет необходимости использовать синтаксис **global \$variable;** для доступа к ним в функциях и методах.

Суперглобальными переменными являются:

- **`$_SERVER`**
- **`$_GET`**
- **`$_POST`**
- **`$_SESSION`**

В PHP существует ряд предварительно определенных переменных, которые не меняются при выполнении всех приложений в конкретной среде. Их также называют переменными окружения или переменными среды. Они отражают установки среды Web-сервера Apache, а также информацию о запросе данного браузера. Есть возможность получить значения URL, строки запроса и других элементов HTTP-запроса.

В кратце: сохраняет, передает, получает важные данные в этих переменных(массивах)

1. Шаблони High Cohesion та Low Coupling. Їх місце у багаторівневій архітектурі.

GRASP = General Responsibility Assignment Software Patterns = Общие шаблоны распределения обязанностей в программных системах. Это шаблоны, используемые в объектно-ориентированном проектировании для решения общих задач по назначению обязанностей классам и объектам объектно-ориентированном. Таким образом, GRASP-шаблоны — это хорошо документированные, стандартизованные и проверенные временем принципы объектно-ориентированного анализа, а не попытка принести что-то принципиально новое.

Low Coupling — это принцип, который позволяет распределить обязанности между объектами таким образом, чтобы степень связанности между системами оставалась низкой.

Степень связанности (coupling) — это мера, определяющая, насколько жестко один элемент связан с другими элементами, либо каким количеством данных о других элементах он обладает.

/* Связывания объектов бывает: - Объект TypeX содержит атрибут (переменную-член), который ссылается на экземпляр объекта TypeY или сам объект TypeY.

-Объект TypeX вызывает службы объекта TypeY.

-Объект TypeX содержит метод, который каким-либо образом ссылается на экземпляр объекта TypeY или сам объект TypeX (обычно это подразумевает использование TypeY в качестве типа параметра, локальной переменной или возвращаемого значения).

-Объект TypeX является прямым или непрямым подклассом объекта TypeY.

-Объект TypeY является интерфейсом, а TypeX реализует этот интерфейс. */

Шаблон Low Coupling поддерживает независимость классов, что, в свою очередь, повышает возможности повторного использования и обеспечивает более высокую эффективность приложения.

Отношение наследования повышает степень связанности классов.

Элемент с низкой степенью связанности (или слабым связыванием) зависит от не очень большого числа других элементов. Изменения компонентов мало сказываются на других объектах. Принципы работы и функции компонентов можно понять, не изучая другие объекты. Таким образом, мы имеем

- Малое число зависимостей между классами (подсистемами).
- Слабую зависимость одного класса (подсистемы) от изменений в другом классе (подсистеме).
- Высокую степень повторного использования подсистем.

High Cohesion — это принцип, который задаёт свойство сильного зацепления *внутри* подсистемы. Классы (подсистемы) таким образом получаются сфокусированными, управляемыми и понятными.

Зацепление (cohesion) (или более точно, функциональное зацепление) — это мера связанности и сфокусированности обязанностей класса.

Считается, что объект (подсистема) обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет огромных объемов работы. Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к следующим проблемам: трудность понимания, сложность при повторном использовании, сложность поддержки, ненадежность, постоянная подверженность изменениям.

/* Степени зацепления:

-Очень слабое зацепление: Только один класс отвечает за выполнение множества операций в самых различных функциональных областях.

-Слабое зацепление. Класс несет единоличную ответственность за выполнение сложной задачи из одной функциональной области.

-Среднее зацепление. Класс имеет несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой.

-Сильное зацепление. Класс имеет среднее количество обязанностей из одной функциональной области и для выполнения своих задач взаимодействует с другими классами. */

Считается, что любая спроектированная система, должна удовлетворять принципам низкой связности и высокого зацепления модулей. Соответствие данным шаблонам позволяет легко модифицировать и сопровождать программный код а также повышает степень его повторного использования.

Мера связности модулей определяется количеством информации которой располагает один модуль о природе другого. В свою очередь, мера зацепления модуля определяется степенью сфокусированности его обязанностей.

Таким образом, примером хорошего дизайна системы может служить набор утилит, в котором, каждая утилита (если ее рассматривать как модуль) выполняет лишь минимальные обязанности (высокое зацепление) и почти ничего не знает о природе других утилит (низкая связность), в связи с чем может быть легко заменена на аналог в некотором варианте использования.

2. Призначення та взаємозв'язок Web-сервера та серверу додатків.

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров (делает запросы (POST, GET)), и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.

Клиент, которым обычно является веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URL-адресами. Ресурсы — это HTML-страницы, изображения, файлы,

медиа-потоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Веб-серверы могут иметь различные дополнительные функции, например:

- Автоматизация работы веб страниц;
- ведение журнала обращений пользователей к ресурсам;
- автентификация и авторизация пользователей;
- поддержка динамически генерируемых страниц.

Сервер приложений (англ. [application server](#)) — это программная платформа ([software framework](#)), предназначенная для эффективного исполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения.

Сервер приложений - это, грубо говоря, программа, позволяющая централизованно на стороне сервера хранить и выполнять приложения, управляя при этом распределением нагрузки, обеспечивая работу с БД и т.д. Если сервер приложений работает с клиентами по протоколу HTTP, то он одновременно является и веб-сервером.

Обычно этот термин относится к Java-серверам приложений. В этом случае сервер приложений ведет себя как расширенная виртуальная машина для запуска приложений, прозрачно управляя соединениями с базой данных с одной стороны и соединениями с веб-клиентом с другой.

Преимущества серверов приложений

Целостность данных и кода

Выделяя бизнес логику на отдельный сервер, или на небольшое количество серверов, можно гарантировать обновления и улучшения приложений для всех пользователей. Отсутствует риск, что старая версия приложения получит доступ к данным или сможет их изменить старым несовместимым образом.

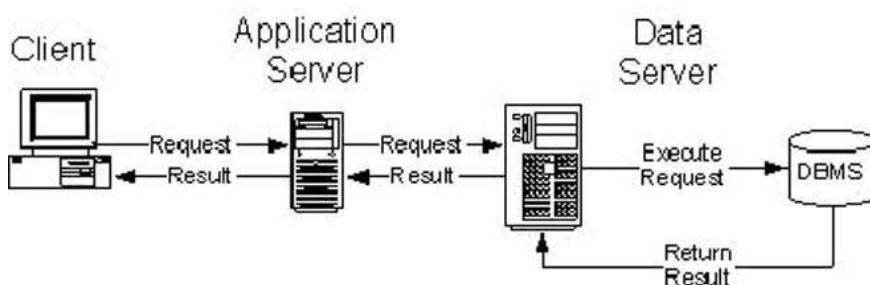
Централизованная настройка и управление

Безопасность

Сервер приложений действует как центральная точка, используя которую, поставщики сервисов могут управлять доступом к данным и частям самих приложений, что считается преимуществом защиты.

Поддержка транзакций

[Транзакция](#) представляет собой единицу активности, во время которой большое число изменений ресурсов (в одном или различных источниках) может быть выполнено атомарно (как неделимая единица работы).



3. Призначення суперглобальних масивів PHP.

В PHP версии 4 и выше введено понятие "суперглобальных" массивов. Эти массивы содержат всю информацию о состоянии сервера и среды выполнения скрипта. Массивы доступны в любом месте скрипта без дополнительных объявлений, т.е. не надо использовать ключевое слово **global**.

Всего массивов девять. Имена всех массивов записываются заглавными буквами, а начинается имя всегда с "\$_" (кроме массива \$GLOBALS).

\$GLOBALS	Массив содержит ссылки на все переменные, объявленные в данном скрипте. Это ассоциативный массив, в котором имена переменных являются ключами.
\$_SERVER	Массив содержит все данные о настройках среды выполнения скрипта и параметры сервера.
\$_GET	Список переменных, переданных скрипту методом GET, т.е. через параметры URL-запроса.
\$_POST	Список переменных, переданных скрипту методом POST.
\$_COOKIE	Массив содержит все cookies, которые сервер установил на стороне пользователя.
\$_FILES	Содержит список файлов, загруженных на сервер из формы. Более детально мы рассмотрим этот массив в уроке, посвящённом загрузке файлов на сервер.
\$_ENV	Содержит переменные окружения, установленные для всех скриптов на сервере.
\$_REQUEST	Этот массив объединяет массивы \$_GET, \$_POST и \$_COOKIE. очень часто бывает удобен при обработке пользовательских запросов, но применять его для защищённой обработки данных не стоит.
\$_SESSION	Массив содержит все переменные сессии текущего пользователя.

Преимущества:

1. Переменная никогда не сможет быть изменена внешним пользователем; если сценарий записал туда некое значение, то оно там и останется!
2. Массивы \$_GET и \$_POST помогут вам отделить данные, пришедшие от формы методом POST (в теле запроса) от данных, добавленных кем-либо, пытающимся имитировать передачу данных методом GET, в конец адресной строки. .
3. Суперглобальные массивы позволят вам не только изолировать значения переменных, но и разнести их по категориям, в зависимости от способа доставки на сервер. А кое-кому значительно усложнят взлом вашего сервера.

Таким образом, назначение суперглобальных массивов в php заключается в том, чтобы обеспечивать безопасность принимаемых данных. Все переменные, которые имеются, содержатся в суперглобальных массивах описанных выше. Т.о. если мы будем использовать несуществующие переменные , то будет выведено сообщение об ошибке.

1. Шаблони High Cohesion та Low Coupling. Їх місце у багаторівневій архітектурі.

GRASP = General Responsibility Assignment Software Patterns = Общие шаблоны распределения обязанностей в программных системах. Это шаблоны, используемые в объектно-ориентированном проектировании для решения общих задач по назначению обязанностей классам и объектам объектно-ориентированном. Таким образом, GRASP-шаблоны — это хорошо документированные, стандартизованные и проверенные временем принципы объектно-ориентированного анализа, а не попытка принести что-то принципиально новое.

Low Coupling — это принцип, который позволяет распределить обязанности между объектами таким образом, чтобы степень связанности между системами оставалась низкой.

Степень связанности (coupling) — это мера, определяющая, насколько жестко один элемент связан с другими элементами, либо каким количеством данных о других элементах он обладает.

/* Связывания объектов бывает: - Объект TypeX содержит атрибут (переменную-член), который ссылается на экземпляр объекта TypeY или сам объект TypeY.

-Объект TypeX вызывает службы объекта TypeY.

-Объект TypeX содержит метод, который каким-либо образом ссылается на экземпляр объекта TypeY или сам объект TypeX (обычно это подразумевает использование TypeY в качестве типа параметра, локальной переменной или возвращаемого значения).

-Объект TypeX является прямым или непрямым подклассом объекта TypeY.

-Объект TypeY является интерфейсом, а TypeX реализует этот интерфейс. */

Шаблон Low Coupling поддерживает независимость классов, что, в свою очередь, повышает возможности повторного использования и обеспечивает более высокую эффективность приложения. Отношение наследования повышает степень связанности классов.

Элемент с низкой степенью связанности (или слабым связыванием) зависит от не очень большого числа других элементов. Изменения компонентов мало сказываются на других объектах. Принципы работы и функции компонентов можно понять, не изучая другие объекты. Таким образом, мы имеем

- Малое число зависимостей между классами (подсистемами).
- Слабая зависимость одного класса (подсистемы) от изменений в другом классе (подсистеме).
- Высокую степень повторного использования подсистем.

High Cohesion — это принцип, который задаёт свойство сильного зацепления *внутри* подсистемы. Классы (подсистемы) таким образом получаются сфокусированными, управляемыми и понятными.

Зацепление (cohesion) (или более точно, функциональное зацепление) — это мера связанности и сфокусированности обязанностей класса.

Считается, что объект (подсистема) обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет огромных объемов работы. Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к следующим проблемам: трудность понимания, сложность при повторном использовании, сложность поддержки, ненадежность, постоянная подверженность изменениям.

/* Степени зацепления:
-Очень слабое зацепление: Только один класс отвечает за выполнение множества операций в самых различных функциональных областях.
-Слабое зацепление. Класс несет единоличную ответственность за выполнение сложной задачи из одной функциональной области.
-Среднее зацепление. Класс имеет несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой.
-Сильное зацепление. Класс имеет среднее количество обязанностей из одной функциональной области и для выполнения своих задач взаимодействует с другими классами. */

Считается, что любая спроектированная система, должна удовлетворять принципам низкой связности и высокого зацепления модулей. Соответствие данным шаблонам позволяет легко модифицировать и сопровождать программный код а также повышает степень его повторного использования.

Мера связности модулей определяется количеством информации которой располагает один модуль о природе другого. В свою очередь, мера зацепления модуля определяется степенью сфокусированности его обязанностей.

Таким образом, примером хорошего дизайна системы может служить набор утилит, в котором, каждая утилита (если ее рассматривать как модуль) выполняет лишь минимальные обязанности (высокое зацепление) и почти ничего не знает о природе других утилит (низкая связность), в связи с чем может быть легко заменена на аналог в некотором варианте использования.

2. Призначення та взаємозв'язок Web-сервера та серверу додатків.

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров (делает запросы (POST, GET)), и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.

Клиент, которым обычно является веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URL-адресами. Ресурсы — это HTML-страницы, изображения, файлы, медиа-потоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Веб-серверы могут иметь различные дополнительные функции, например:

Автоматизация работы веб страниц;
ведение журнала обращений пользователей к ресурсам;
автентификация и авторизация пользователей;
поддержка динамически генерируемых страниц.

Сервер приложений (англ. [application server](#)) — это программная платформа ([software framework](#)), предназначенная для эффективного исполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения.

Сервер приложений - это, грубо говоря, программа, позволяющая централизованно на стороне сервера хранить и выполнять приложения, управляя при этом распределением нагрузки, обеспечивая работу с БД и т.д. Если сервер приложений работает с клиентами по протоколу HTTP, то он одновременно является и веб-сервером.

Обычно этот термин относится к Java-серверам приложений. В этом случае сервер приложений ведет себя как расширенная виртуальная машина для запуска приложений, прозрачно управляя соединениями с базой данных с одной стороны и соединениями с веб-клиентом с другой.

Преимущества серверов приложений

Целостность данных и кода

Выделяя бизнес логику на отдельный сервер, или на небольшое количество серверов, можно гарантировать обновления и улучшения приложений для всех пользователей. Отсутствует риск, что старая версия приложения получит доступ к данным или сможет их изменить старым несовместимым образом.

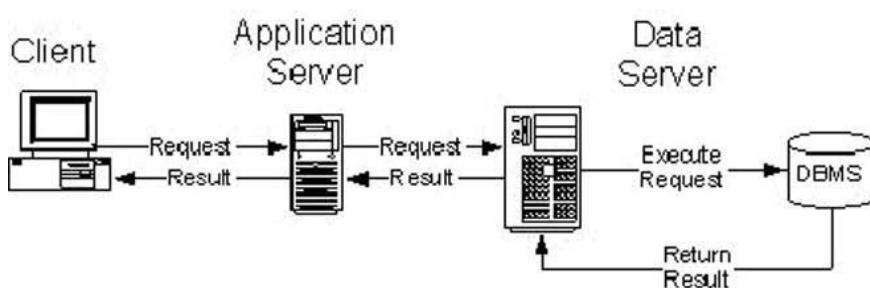
Централизованная настройка и управление

Безопасность

Сервер приложений действует как центральная точка, используя которую, поставщики сервисов могут управлять доступом к данным и частям самих приложений, что считается преимуществом защиты.

Поддержка транзакций

[Транзакция](#) представляет собой единицу активности, во время которой большое число изменений ресурсов (в одном или различных источниках) может быть выполнено атомарно (как неделимая единица работы).



3. Призначення суперглобальних масивів PHP.

В PHP версии 4 и выше введено понятие "суперглобальных" массивов. Эти массивы содержат всю информацию о состоянии сервера и среды выполнения скрипта. Массивы доступны в любом месте скрипта без дополнительных объявлений, т.е. не надо использовать ключевое слово **global**.

Всего массивов девять. Имена всех массивов записываются заглавными буквами, а начинается имя всегда с `$_` (кроме массива `$GLOBALS`).

<code>\$GLOBALS</code>	Массив содержит ссылки на все переменные, объявленные в данном скрипте. Это ассоциативный массив, в котором имена переменных являются ключами.
<code>\$_SERVER</code>	Массив содержит все данные о настройках среды выполнения скрипта и параметры сервера.
<code>\$_GET</code>	Список переменных, переданных скрипту методом GET, т.е. через параметры URL-запроса.
<code>\$_POST</code>	Список переменных, переданных скрипту методом POST.
<code>\$_COOKIE</code>	Массив содержит все cookies, которые сервер установил на стороне пользователя.
<code>\$_FILES</code>	Содержит список файлов, загруженных на сервер из формы. Более детально мы рассмотрим этот массив в уроке, посвящённом загрузке файлов на

	сервер.
\$_ENV	Содержит переменные окружения, установленные для всех скриптов на сервере.
\$_REQUEST	Этот массив объединяет массивы \$GET, \$POST и \$COOKIE. очень часто бывает удобен при обработке пользовательских запросов, но применять его для защищённой обработки данных не стоит.
\$_SESSION	Массив содержит все переменные сессии текущего пользователя.

Преимущества:

4. Переменная никогда не сможет быть изменена внешним пользователем; если сценарий записал туда некое значение, то оно там и останется!
5. Массивы \$_GET и \$_POST помогут вам отделить данные, пришедшие от формы методом POST (в теле запроса) от данных, добавленных кем-либо, пытающимся имитировать передачу данных методом GET, в конец адресной строки. .
6. Суперглобальные массивы позволят вам не только изолировать значения переменных, но и разнести их по категориям, в зависимости от способа доставки на сервер. А кое-кому значительно усложнят взлом вашего сервера.

Таким образом, назначение суперглобальных массивов в php заключается в том, чтобы обеспечивать безопасность принимаемых данных. Все переменные, которые имеются, содержатся в суперглобальных массивах описанных выше. Т.о. если мы будем использовать несуществующие переменные , то будет выведено сообщение об ошибке.

Билет 9

1. Призначення та методологія об'єктного аналізу в межах дослідження предметної області.
2. Протокол НТТР. Призначення. Переваги та недоліки.
3. Призначення Web-сервера, його місце в архітектурі Web-додатку.

1. Призначення та методологія об'єктного аналізу в межах дослідження предметної області.

Загальні положення

Важливі визначення теорії об'єктно-орієнтованого аналізу були дані в класичній книзі Г. Буча «Об'єктно-орієнтований аналіз та проектування».

Парадигма об'єктно-орієнтованого програмування випливає з об'єктно-орієнтованого сприйняття світу, що складається з великої кількості об'єктів. Вони є порівняно незалежними, але постійно взаємодіють між собою. Кожний об'єкт має певні властивості та вміє виконувати деякі функції. Можна вважати, що об'єктна модель є конкретизацією абстрактнішої фреймової моделі. Об'єктом називається абстракція, що характеризується станом, поведінкою та ідентифікованістю; сукупності схожих об'єктів утворюють клас; терміни «екземпляр (примірник) класу» та «об'єкт» рівноправні.

Стан об'єкта характеризується переліком (як правило, статичним) усіх властивостей об'єкта і поточними (як правило, динамічними) значеннями кожної з цих властивостей.

До цього можна додати, що «статичний перелік властивостей» є характеристикою всього класу, а «поточні динамічні значення» — характеристикою окремого об'єкта — екземпляру класу. Опис окремого екземпляру на основі загального опису класу можна отримати, якщо визначити конкретні значення властивостей. Поведінка визначається тим, як об'єкт функціонує та реагує на зовнішні події; поведінку прийнято характеризувати в термінах зміни станів об'єкта та передачі повідомлень між об'єктами; поточний стан об'єкта є сумарним результатом його поведінки.

Ідентифікованість — це така властивість об'єкта, яка відрізняє його від усіх інших об'єктів.

Важливим є те, що об'єкти слід розглядати як абстракції певних сутностей, тобто об'єкт описує властивості даної сутності, що є найважливішими з певної точки зору.

При цьому для екземплярів класу спільними є всі характеристики, а не деякі. Точніше, спільним є перелік характеристик, а не конкретні значення; екземпляри одного класу можна розрізняти між собою саме за рахунок того, що характеристики різних екземплярів класу мають різні значення.

Призначення ОOA

OOA - методологія аналізу суті реального світу на основі понять класу і об'єкту, складових словник наочної області, для розуміння і пояснення того, як вони (суть) взаємодіють між собою. Моделі ОOA надалі перетворюються в об'єктно-орієнтований проект.

2. Протокол HTTP. Призначення. Переваги та недоліки.

HTTP

Протокол HTTP (Hypertext Transfer Protocol, протокол передачі гіпертекстів) — це протокол прикладного рівня для розподілених гіпертекстових інформаційних систем. Крім передачі гіпертекстів він може застосовуватися й в інших областях, таких, як сервери імен і розподілені системи керування об'єктами, але для наших цілей важливо те, що на цьому протоколі з моменту своєї появи в 1990 р. і донині базується World Wide Web. Приведений нижче короткий його опис заснований на специфікації HTTP/1.1 (RFC 2616 , червень 1999 р.).

HTTP є протоколом типу запит/відгук. Клієнт посилає серверу запит, що складається з типу запиту, URI і версії протоколу, за яких випливає повідомлення, що містить модифікатори запиту, клієнтську інформацію і, можливо, тіло запиту. Сервер відповідає рядком стану, що містить версію протоколу і код стану, за якою слідує повідомлення, що містить серверну інформацію, метаінформацію і, можливо, тіло повідомлення.

У дійсності, ситуація звичайно є більш складною через участь у процесі обміну інформацією посередників між клієнтом і сервером. Існують три таких посередники: проксі-сервер, шлюз і тунель:

Проксі-сервер (або сервер повноважень) приймає запит клієнта, включаючи повний URI, переписує все повідомлення або частину його і передає виправлений запит серверу, зазначеному в URI.

Шлюз розташовується над сервером і при необхідності транслює запити до протоколу більш низького рівня, підтримуваний сервером.

Тунель не змінює переданих повідомлень, а використовується при передачі даних через посередника типу брандмауера (firewall).

Переваги

Простота – протокол настільки простий у реалізації, що дозволяє з легкістю створювати не тільки клієнтські додатки, але і примітивні сервери.

Розширеність - ви можете легко розширювати можливості протоколу завдяки упровадженню своїх власних заголовків, зберігаючи сумісність з іншими клієнтами і серверами. Вони будуть ігнорувати невідомі їм заголовки, але при цьому ви можете одержати необхідний вам функціонал при рішенні специфічної задачі.

Поширеність - при виборі протоколу HTTP для рішення конкретних задач немаловажним фактором є його поширеність. Як наслідок, це численність різної документації по протоколу на багатьох мовах світу, вставка зручних у використанні засобів розробки в популярні IDE, підтримка протоколу як клієнта багатьма програмами і великий вибір серед хостингових компаній із серверами HTTP.

Недоліки і проблеми

Великий розмір повідомлень - використання текстового формату в протоколі породжує відповідний недолік: великий розмір повідомлень у порівнянні з передачею двійкових даних. Через це зростає навантаження на устаткування при формуванні, обробці і передачі повідомлень. Для розв'язання даної проблеми до протоколу вбудовані засоби для забезпечення кешування на стороні клієнта, а також засобу компресії переданого контенту. Нормативними документами по протоколі передбачена наявність проксі-серверів, що дозволяють клієнту одержати документ із найбільш близького до нього сервера. Також до протоколу було впроваджене дельта-кодування, щоб клієнту передавався не весь документ, а тільки його змінена частина.

Відсутність «навігації» - хоча протокол розробляється як засіб роботи з ресурсами сервера, у нього відсутні в явному виді засоби навігації серед цих ресурсів. Наприклад, клієнт не може явно запросити список доступних файлів, як у протоколі FTP. Передбачалося, що кінцевий користувач уже знає URI необхідного йому документа, закачавши який, він буде робити навігацію завдяки гіперпосиланням. Це цілком нормальні і зручно для людини, але важко, коли стоять задачі автоматичної обробки й аналізу всіх ресурсів сервера без участі людини. Рішення цієї проблеми лежить цілком на плечах розробників додатків, що використовують даний протокол.

Нема підтримки розподіленості - протокол HTTP розробляється для рішення типових побутових задач де сам по собі час обробки запиту повинен забирати незначний час або взагалі не прийматися в розрахунок. Але в промисловому використанні із застосуванням розподілених обчислень при високих навантаженнях на сервер протокол HTTP виявляється безпомічний. У 1998 році W3C запропонував альтернативний протокол HTTP-NG (англ. HTTP Next Generation) для повної заміни застарілого з акцентуванням уваги саме на цій області. Ідею його необхідності підтримали великі фахівці з розподілених обчислень, але даний протокол дотепер перебуває в стадії розробки.

Структура протоколу

Кожне HTTP-повідомлення складається з трьох частин, що передаються в зазначеному порядку:

- Стартовий рядок (англ. Starting line) — визначає тип повідомлення;
- Заголовки (англ. Headers) — характеризують тіло повідомлення, параметри передачі та інші дані;
- Тіло повідомлення (англ. Message Body) — безпосередньо даного повідомлення. Обов'язково повинно відокремлювати від заголовків порожнім рядком.

Заголовки і тіло повідомлення можуть бути відсутні, але стартовий рядок є обов'язковим елементом, тому що вказує на тип запиту/відповіді. Виключенням є версія 0.9 протоколу, у якої повідомлення запиту містить тільки стартовий рядок, а повідомлення відповіді тільки тіло повідомлення.

3. Призначення Web-сервера, його місце в архітектурі Web-додатку.

Веб-сéрвер (англ. Web Server) — це сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними. Веб-сервер — основа Всесвітньої павутини.

Веб-сервером називають як програмне забезпечення, що виконує функції веб-сервера, так і комп'ютер, на якому це програмне забезпечення працює.

Клієнти дістаються веб-сервера за URL-адресою потрібної їм веб-сторінки або іншого ресурсу.

Веб-застосунок — розподілений застосунок, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер може бути реалізацією так званих тонких клієнтів. Браузер здатний відображати веб-сторінки і, зазвичай, входить до складу операційної системи, а його оновлення і супровід здійснює постачальник операційної системи. Логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у відображені інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-застосунки є міжплатформовими сервісами. Унаслідок цієї універсальноті і відносної простоти розробки веб-застосунки стали широко популярними в кінці 1990-х — початку 2000-х років.

Істотною перевагою побудови веб-застосунків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб писати різні версії для Microsoft Windows, Mac OS X, GNU/Linux та інших операційних систем, застосунок створюється один раз для довільно обраної платформи і на ній розгортається. Проте різна реалізація HTML, CSS, DOM та інших специфікацій в браузерах може викликати проблеми при розробці веб-застосунків і подальшої підтримки. Крім того, можливість користувача настроювати багато параметрів браузера (наприклад, розмір шрифту, кольори, відключення підтримки сценаріїв) може перешкоджати коректній роботі застосунку.

Інший (менш універсальний) підхід полягає у використанні Adobe Flash або Java-аплетів для повної або часткової реалізації призначеного для користувача інтерфейсу. Оскільки більшість браузерів підтримують ці технології (зазвичай, за допомогою плагінів), Flash- або Java-застосунки можуть легко виконуватись. Оскільки вони надають програмістові більший контроль над інтерфейсом, то здатні обходити багато несумісностей у конфігураціях браузерів, хоча несумісність між Java або Flash реалізаціями клієнта може спричиняти різні ускладнення. У зв'язку з архітектурною схожістю з традиційними клієнт-серверними застосунками, певним чином «створюючи» клієнтами, існують суперечки щодо коректності зарахування подібних систем до веб-застосунків; альтернативний термін «Насичений інтернет-застосунок» (англ. Rich Internet Application).

Архітектура веб-застосунків

Веб-застосунок отримує запит від клієнта і виконує обчислення, після цього формує веб-сторінку і відправляє її клієнтові мережею з використанням протоколу HTTP. Саме веб-застосунок може бути клієнтом інших служб, наприклад, бази даних або стороннього веб-застосунку, розташованого на іншому сервері. Яскравим прикладом веб-застосунку є система управління вмістом статей Вікіпедії: безліч її учасників можуть брати участь у створенні мережової енциклопедії, використовуючи для цього браузери своїх операційних систем (Microsoft Windows, GNU/Linux або будь-якої іншої операційної системи) без завантаження додаткових виконуваних модулів для роботи з базою даних статей.

Для більшої інтерактивності і продуктивності розроблений новий підхід до розробки веб-застосунків, названий AJAX, і який нині є стандартним де-факто. При використанні Ajax сторінки веб-застосунку здатні відправляти веб-запити до сервера у фоновому режимі, і не перезавантажуються цілком, а лише довантажують необхідні дані з сервера, що значно пришвидшує роботу і робить її зручнішою.

Для створення веб-застосунків використовуються різноманітні серверні технології та мови програмування

реализации которых можно существенно повысить уровень защищенности Web-приложения и предотвратить (или как минимум существенно усложнить) возможность перехвата сеансов.

Реализация механизмов отслеживания сеансов и сохранения состояний

Перехват сеанса возможен в тот момент, когда Web-приложение оказывается полностью зависимым от обработки информации о сеансах и состояниях в клиентской части. В некоторых случаях подмену идентификаторов можно выполнить, даже несмотря на реализацию механизма отслеживания сеансов в серверной части. Если для исследования идентификаторов сеансов выполнялось несколько предварительных атак, то последующий перехват сеанса может оказаться гораздо проще. Поэтому реализации адекватных методов отслеживания сеансов и сохранения состояний необходимо уделять самое пристальное внимание.

Основываясь на практическом опыте, можно сформулировать ряд правил реализации механизмов поддержки сеансов и отслеживания состояний. Эти правила ни в коей мере не являются нечерньюющими или универсальными. Скорее, их нужно рассматривать как рекомендации, которыми следует руководствоваться при разработке механизмов отслеживания сеансов и состояний.

Идентификаторы сеансов должны быть уникальными

В каждом Web-приложении между браузером и Web-сервером должен устанавливаться логический сеанс взаимодействия. Именно для этого между браузером и сервером должна курсировать строка или последовательность данных (идентификатор сеанса и его имя). Даже если один и тот же пользователь регистрируется в системе повторно, для него должен генерироваться новый идентификатор сеанса и в идеале соответствующее имя.

Идентификаторы сеансов должны быть сложными

Наиболее простой способ выявления "слабых" идентификаторов заключается в быстром создании множества сеансов или многократном их создании/разрушении в течение короткого периода времени. Идентификаторы сеансов, которые последовательно увеличиваются, используют временные метки или основанные на каком-либо другом шаблоне, должны вызывать большое беспокойство. Такие идентификаторы сеансов плохо защищены от последовательности предварительных атак, на основе которых взломщик сможет перехватить сеанс, успешно сгенерировав идентификаторы сеансов, совпадающие с идентификаторами текущих пользователей приложения.

Случайные идентификаторы сеансов можно сгенерировать на основе случайного числа, текущего времени и какого-либо специального числа. На основе этих данных можно сгенерировать хэш-код, который будет использоваться в качестве идентификатора сеанса. Вероятность совпадения хэш-кодов довольно низка, так что с их помощью успех предварительных атак можно свести практически к нулю.

Идентификаторы сеансов должны быть независимыми

Идентификаторы сеансов не должны основываться на именах пользователей, паролях или состояниях приложения. В серверной части можно поддерживать справочную таблицу, которая будет использоваться для связывания идентификаторов сеансов, регистрационных данных пользователя и состояния приложения.

Обеспечение безопасности при использовании механизма поддержки сеансов

Как видно из материала данной главы, поддержка **сеансов** — это чрезвычайно мощный механизм, позволяющий отслеживать сеансы работы с Web-приложением отдельных пользователей и отделять выполняемые ими действия друг от друга. Вместе с тем в контексте одного сеанса зачастую передаются конфиденциальные данные (имена, пароли, номера кредитных карточек и т.д.). Так что в процессе разработки очень важно позаботиться также и об обеспечении безопасности. Ниже приведены некоторые общие рекомендации, касающиеся **подсистемы поддержки сеансов**, при

Creator (Создатель)

См. также: Абстрактная фабрика (шаблон проектирования)

Шаблон Creator решает, кто должен создавать объект. Фактически, это применение шаблона Information Expert к проблеме создания объектов. Более конкретно, нужно назначить классу В обязанность создавать экземпляры класса А, если выполняется как можно больше из следующих условий:

Класс В содержит или агрегирует объекты А.

Класс В записывает экземпляры объектов А.

Класс В активно использует объекты А

Класс В обладает данными инициализации для объектов А.

Альтернативой создателю является шаблон проектирования Фабрика. В этом случае создание объектов концентрируется в отдельном классе.

3.

```
/* hello.cgi.c - Hello, World CGI */
```

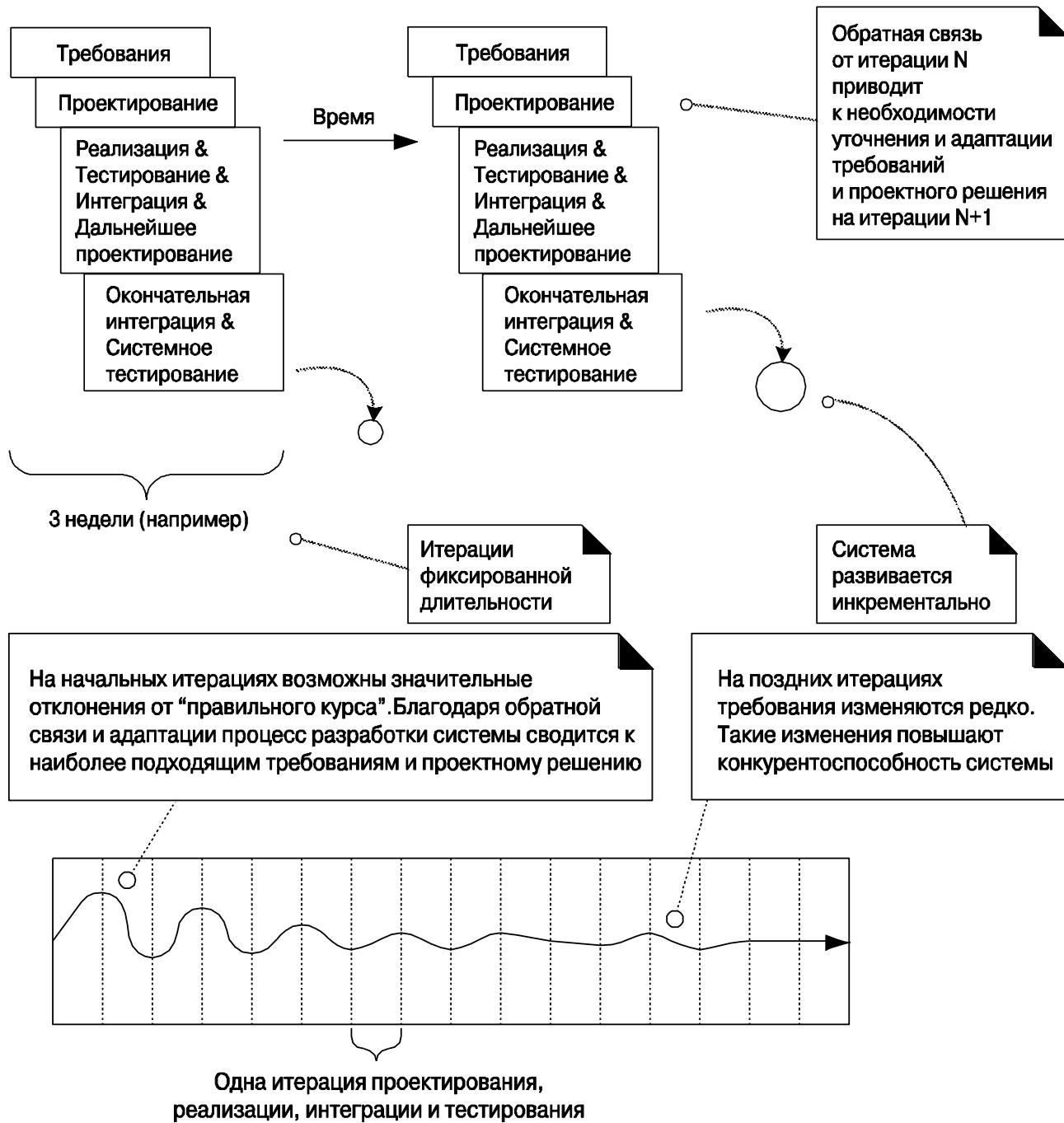
```
#include <stdio.h>
```

```
int main() {
printf("Content-Type: text/html\r\n\r\n");
printf("<html> <head>\n");
printf("<title>Hello, World!</title>\n");
printf("</head>\n");
printf("<body>\n");
printf("<h1>Hello, World!</h1>\n");
printf("</body> </html>\n");
}
```

Билет 12

1. Разработка выполняется в виде нескольких краткосрочных мини-проектов фиксированной длительности (например, по 4 недели), называемых итерациями (iteration)
Каждая итерация включает свои собственные этапы:

анализа требований, проектирования, реализации и завершается тестированием, интеграцией и созданием работающей системы



- ❖ Своевременное осознание возможных технических рисков, осмысление требований, задач проекта и удобства использования системы
- ❖ Быстрый и заметный прогресс
- ❖ Ранняя обратная связь, возможность учета пожеланий пользователей и адаптации системы. В результате система более точно удовлетворяет реальные требования руководителей и потребителей

- ◆ Управляемая сложность. Команда разработчиков не перегружена лишней работой в рамках этапа анализа и проектирования и не “парализована” слишком сложными и долгосрочными задачами
 - ◆ Полученный при реализации каждой итерации опыт можно методически использовать для улучшения самого процесса разработки
2. Базується на взаємодії клієнту безпосередньо з розподіленими об'єктами на сервері (при цьому протокол HTTP може використовуватись лише як транспорт)
Дозволяє використовувати всі переваги об'єктно-орієнтованого підходу
- скорочення часу розробки (автономна розробка)
 - повторне використання програмних компонентів

3.

```
package hall;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
}
}
```

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 13

1. Діаграма прецедентів. Призначення. Їх взаємозв'язок з діаграмами послідовностей.

Прецедент – це опис послідовності подій використання системи, опис будь-якого варіанту використання системи

В діаграммі прецедентів ілюструється набір прецедентів системи і виконавців, а також взаємозв'язки між ними

Призначення : представити контекстну діаграмму, дозволяючи швидко визначити зовнішніх виконавців і ключові методи

Діаграмма послідовностей – це схема, яка для певного сценарію прецедента показує генеруємі зовнішніми виконавцями події, їх порядок, а також події, що генеруються всередині самої системи

2. Основні технології побудови AJAX-додатків

AJAX (Asynchronous Javascript and XML – « а с и н х р о н н ы й JavaScript и XML») - підхід до побудови інтерактивних веб-додатків, що полягає в

«Фоновому» обміні даними браузера з веб-сервером. В результаті, при оновленні

даних, веб-сторінка не перезавантажується повністю, і веб-додатки стають більш швидкими і зручними.

AJAX являє собою не одну, а групу технологій і базується на принципах

використання DHTML для динамічної зміни змісту сторінки і

використання XMLHttpRequest для звернення до сервера

Основні переваги технології:

- Можливість заощадити трафік, завантажуючи тільки необхідну частину веб-сторінки;
- Можливість знизити навантаження на сервер;

- Створення більш інтерактивних і швидких додатків , максимально бістро реагуючих на клік пользователь .
- До недоліків можна віднести:
- Необхідність підтримки браузером Java Script :
- Проблеми з індексацією сторінок пошуковими системами ;
- Неможливість додавання сторінок в закладки ;
- Складності при вдосконаленні та доопрацюванні проекту .
- Найбільш загальні випадки застосування AJAX :
 - Перевірка правильності заповнення форми з залученням можливостей сервера;
 - Удосконалення існуючого інтерфейсу , додавання підказок і т.д.
- Створення простих чатів
- Створення додатків, що вимагають оновлення інформації в режимі реального часу

3 Проілюструвати відмінності сервету від сторінок JSP.

Сервлет є Java-інтерфейсом, реалізація якого розширює функціональні можливості сервера. Сервлет взаємодіє з клієнтами за допомогою принципу запит-відповідь.

Хоча сервлети можуть обслуговувати будь-які запити, вони зазвичай використовуються для розширення веб-серверів. Для таких додатків технологія Java Servlet визначає HTTP-специфічні сервлет класи.

JSP (Java Server Pages) — технологія, що дозволяє веб-розробникам динамічно генерувати [HTML](#), [XML](#) та інші веб-сторінки.

JSP — одна із високопродуктивних технологій, оскільки весь код сторінки транслізується в java-код сервлету за допомогою компілятора JSP

сторінок (напр. [Jasper](#)), а потім компілюється в [байт-код](#) віртуальної машини [java \(JVM\)](#).

JSP (Java Server Pages) - це розширення сервлетів - ще більше нагадує PHP з синтаксису. JSP доцільніше використовувати коли більша частина документа складається з "статичної" HTML-розмітки. Якщо ж велика частина документа динамічно змінюється - доцільніше використовувати сервлет. За великим рахунком, JSP перетвориться в сервлет під час роботи і все те, що актуально для сервлета - буде актуально і для JSP. Різниця в відмінному синтаксисі.

С точки зрения web-программирования, framework-система (CMF-система) это платформа позволяющая решать задачи, которые постоянно возникают при создании Интернет-приложений [8].

В современных фреймворках заложен базовый функционал и набор архитектурных стандартов, которые система накладывает на web-приложения. Это снимает с разработчиков необходимость придумывать все с нуля и позволяет более эффективно использовать код повторно; CMF-системы обеспечивают инверсию управления (от англ. Inversion of Control, IoC) Инверсия управления переносит ответственность за выполнение действий с кода приложения на фреймворк.

Рассмотрим наиболее известные framework-системы.

ASP.NET — технология создания веб-приложений и веб-сервисов от компании Майкрософт. Она является составной частью платформы Microsoft.NET и развитием более старой технологии Microsoft ASP. Корпорация Майкрософт выпустила несколько расширений для ASP.NET:

- ASP.NET AJAX
- ASP.NET MVC Framework

ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует шаблон проектирования Model-View-Controller (MVC). В апреле 2009 года, исходный код ASP.NET MVC был опубликован под лицензией Microsoft Public License (MS-PL) Проект ASP.NET MVC — это часть платформы ASP.NET. Разработка приложений ASP.NET MVC можно рассматривать как альтернативу разработке страниц web-форм ASP.NET, но не как замену модели web-форм [12] .

Zend Framework (ZF)—является объектно-ориентированным фреймворком с открытым исходным кодом (лицензия New BSD), использует PHP5. Основывается на идеях MVC. Разрабатывается компанией Zend, являющейся разработчиком самого PHP. Помимо MVC-компонентов Zend Framework содержит множество библиотек, необходимых для построения приложения. Также есть компоненты для интеграции со сторонними сервисами.

CakePHP—это программный каркас для создания веб-приложений, написанный на языке PHP и построенный на принципах открытого ПО — лицензия MIT. Многие идеи и принципы CakePHP были заимствованы от Ruby

on Rails. CakePHP отличается от других своих собратьев (Symfony, PHPonTrax) тем, что он полностью совместим как с PHP4 так и с PHP5.

Структура XML-документа

XML-документ состоит из деклараций, элементов, комментариев, специальных символов и директив. Все эти составляющие документа описаны в данной главе.

8.1.3.1. Элементы и атрибуты

XML — это *теговый язык* разметки документов. Иными словами, любой документ на языке XML представляет собой набор **элементов**, причем начало и конец каждого элемента обозначается специальными пометками, называемыми *тегами*.

Элемент состоит из трех частей: начального тега, содержимого и конечного тега. Тег — это текст, заключенный в угловые скобки "<" и ">". Конечный тег имеет тоже имя, что начальный тег, но начинается с косой черты "/". Пример XML-элемента:

```
<author>Сергей Довлатов</author>
```

Имена элементов зависят от регистра, т. е. **<author>**, **<Author>** и **<AUTHOR>** — это имена различных элементов. Наличие закрывающего тега всегда обязательно. Если тег является *пустым*, т. е. не имеет содержимого и закрывающего тега, то он имеет специальную форму:

```
<элемент/>
```

Любой элемент может иметь *атрибуты*, содержащие дополнительную информацию об элементе. Атрибуты всегда включаются в начальный тег элемента и имеют вид:

```
имя_атрибута="значение_атрибута"
```

Атрибут обязан иметь значение, которое всегда должно быть заключено в одинарные или двойные кавычки. Имена атрибутов также зависят от регистра. Пример элемента, имеющего атрибут:

```
<author country="USA">Сергей Довлатов</author>
```

Элементы должны либо следовать друг за другом, либо быть вложены один в другой:

```
<books>
  <book isbn="5887821192">
    <title>Часть речи</title>
    <author>Бродский, Иосиф</author>
    <present/>
  </book>
  <book isbn="0345374827">
    <title>Марш одиноких</title>
```

```
<author>Довлатов, Сергей</author>
<present/>
</book>
</books>
```

Здесь элемент `books` (книги) содержит два вложенных элемента `book` (книга), которые, в свою очередь, имеют атрибут `isbn` и содержат три последовательных элемента: `title` (название), `author` (автор) и `present` (есть в наличии), причем последний пуст, т. к. в данном случае соответствует логическому флагжу.

Из приведенного описания видно, что синтаксис XML напоминает синтаксис HTML (что естественно, т. к. оба они являются диалектами одного языка SGML), но требования к оформлению правильных XML-документов выше. Еще одним очень важным отличием XML от HTML является то, что содержимое элементов, т. е. все, что содержится между начальным и конечным тегами, считается данными. Это означает, что XML не игнорирует символы пробела и разрыва строк, как это делает HTML.

8.1.3.2. Пролог и директивы

Любой XML-документ состоит из *пролога и корневого элемента*, например:

```
<?xml version="1.0"?>
<books>
  <book isbn="0345374827">
    <title>Марш одиноких</title>
    <author>Довлатов, Сергей</author>
    <present/>
  </book>
</books>
```

В этом примере пролог сводится к единственной *директиве* (первая строка документа), указывающей версию XML. За ней следует XML-элемент с уникальным именем, который содержит в себе все остальные элементы и называется корневым. Директива (processing instruction) — это выражение, заключенное в специальные теги "<?" и "?>", которое содержит указания программе, обрабатывающей XML-документ.

Стандарт XML резервирует только одну директиву `<?xml version="1.0"?>`, указывающую на версию языка XML, которой соответствует данный документ (второй версии XML пока нет). В действительности, эта директива несколько богаче и в самом общем виде выглядит так:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

Здесь атрибут `encoding` задает кодировку символов документа. По умолчанию считается, что XML-документы должны создаваться в формате [UTF-8](#) или [UTF-16](#). Если же используется какая-либо другая кодировка символов, то ее название согласно [Таблицы П7.1](#) должно быть указано в данном атрибуте, как показано в примере. Атрибут `standalone` говорит о том, содержит ли данный документ [внешние разделы](#). Значение `yes` означает, что таких разделов нет, значение `no` — что они есть.

В общем случае, пролог может содержать также [декларации типа документа](#).

8.1.3.3. Комментарии

XML-документы могут содержать *комментарии*, которые игнорируются приложением, обрабатывающим документ. Комментарии строятся по тем же правилам, что и в HTML:

- начинайте комментарий с символов "<!--",
- завершайте комментарий символами "-->",
- не используйте внутри комментария символов "--".

Пример комментариев:

```
<!-- это комментарий -->
<!-- а вот еще комментарий,
занимающий более одной строки -->
```

8.1.3.4. Имена и данные

Все *имена* элементов, атрибутов и разделов должны начинаться с буквы Unicode и состоять из букв, цифр, символов точки (.), подчеркивания (_) и дефиса (-). Единственное ограничение состоит в том, что они не должны начинаться с комбинации букв xml в любом регистре; подобные имена зарезервированы для будущих расширений языка. Существенно, что стандарт допускает использование в именах не только английских букв, но и любых других, хотя существующие XML-процессоры часто ограничены теми системами кодировок, которые в них заложены создателями. Поэтому мы в своих примерах пишем имена по-английски.

Массив (тип [array](#)) может быть создан языковой конструкцией [array\(\)](#). language construct. В качестве параметров она принимает любое количество разделенных запятыми пар `key => value` (ключ => значение).

```
array(
    key    => value,
    key2   => value2,
    key3   => value3,
    ...
)
```

Существует два типа массивов, различающиеся по способу идентификации элементов.

1. В массивах первого типа элемент определяется индексом в последовательности. Такие массивы называются [простыми массивами](#).
2. Массивы второго типа имеют ассоциативную природу, и для обращения к элементам используются ключи, логически связанные со значениями. Такие массивы называют [ассоциативными массивами](#).

Важной особенностью PHP является то, что PHP, в отличие от других языков, позволяет создавать массивы любой сложности непосредственно в теле программы (скрипта).

Массивы могут быть как [одномерными](#), так и [многомерными](#).

Простые массивы и списки в PHP

При обращении к элементам [простых индексируемых массивов](#) используется целочисленный индекс, определяющий позицию заданного элемента.

Пример #1 Простой массив

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// Начиная с PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

Массивы, индексами которых являются числа, начинающиеся с нуля - это **списки**:

Доступ к элементам простых массивов (списков) осуществляется следующим образом:

```
<?php
// Простой способ инициализации массива
$names[0] = "Апельсин";
$names[1] = "Банан";
$names[2] = "Груша";
$names[3] = "Помидор";
// Здесь: names - имя массива, а 0, 1, 2, 3 - индексы массива

// Выводим элементы массивов в браузер:
echo $names[0]; // Вывод элемента массива names с индексом 0
echo "<br>";
echo $names[3]; // Вывод элемента массива names с индексом 3
// Выводит:
// Апельсин
// Помидор
?>
```

С технической точки зрения разницы между простыми массивами и списками нет.

Простые массивы можно создавать, не указывая индекс нового элемента массива, это за вас сделает PHP.

[Простые многомерные массивы:](#)

Пример простого многомерного массива:

```
<?php
// Многомерный простой массив:
$arr[0][0] = "Овощи";
$arr[0][1] = "Фрукты";
$arr[1][0] = "Абрикос";
$arr[1][1] = "Апельсин";
$arr[1][2] = "Банан";
$arr[2][0] = "Огурец";
$arr[2][1] = "Помидор";
$arr[2][2] = "Тыква";

// Выводим элементы массива:
echo "<h3>".$arr[0][0].":</h3>";
for ($q=0; $q<=2; $q++) {
echo $arr[2][$q]."<br>";
}
echo "<h3>".$arr[0][1].":</h3>";
for ($w=0; $w<=2; $w++) {
echo $arr[1][$w]."<br>";
}
?>
```

Ассоциативные массивы в PHP

В PHP индексом массива может быть не только число, но и строка. Причем на такую строку не накладываются никакие ограничения: она может содержать пробелы, длина такой строки может быть любой.

Ассоциативные массивы особенно удобны в ситуациях, когда элементы массива удобнее связывать со словами, а не с числами.

Итак, массивы, индексами которых являются строки, называются ассоциативными массивами.

Одномерные ассоциативные массивы:

Одномерные ассоциативные массивы содержат только один ключ (элемент), соответствующий конкретному индексу ассоциативного массива. Приведем пример:

```
<?php
// Ассоциативный массив
$names["Иванов"] = "Иван";
$names["Сидоров"] = "Николай";
$names["Петров"] = "Петр";
// В данном примере: фамилии - ключи ассоциативного массива
// , а имена - элементы массива names
?>
```

Доступ к элементам одномерных ассоциативных массивов осуществляется так же, как и к элементам обычновенных массивов, и называется доступом по ключу:

```
echo $names["Иванов"];
```

Многомерные ассоциативные массивы:

Многомерные ассоциативные массивы могут содержать несколько ключей, соответствующих конкретному индексу ассоциативного массива. Рассмотрим пример многомерного ассоциативного массива:

```
<?php
// Многомерный массив
$A["Ivanov"] = array("name"=>"Иванов И.И.", "age"=>"25", "email"=>"ivanov@mail.ru");
$A["Petrov"] = array("name"=>"Петров П.П.", "age"=>"34", "email"=>"petrov@mail.ru");
$A["Sidorov"] = array("name"=>"Сидоров С.С.", "age"=>"47", "email"=>"sidorov@mail.ru");
?>
```

Многомерные массивы похожи на записи в языке Pascal или структуры в языке С.

Доступ к элементам многомерного ассоциативного массива осуществляется следующим образом:

```
echo $A["Ivanov"]["name"]; // Выводит Иванов И.И.
echo $A["Petrov"]["email"]; // Выводит petrov@mail.ru
```

Ассоциативные многомерные массивы можно создавать и классическим способом, хотя это не так удобно:

```
<?php
// Многомерный ассоциативный массив
$A["Ivanov"]["name"]="Иванов И.И.";
$A["Ivanov"]["age"]="25";
$A["Ivanov"]["email"]="ivanov@mail.ru";

$A["Petrov"]["name"]="Петров П.П.";
$A["Petrov"]["age"]="34";
$A["Petrov"]["email"]="petrov@mail.ru";

$A["Sidorov"]["name"]="Сидоров С.С.";
$A["Sidorov"]["age"]="47";
$A["Sidorov"]["email"]="sidorov@mail.ru";

// Получаем доступ к ключам многомерного ассоциативного массива
echo $A["Ivanov"]["name"]."  
"; // Выводит Иванов И.И.
echo $A["Sidorov"]["age"]."  
"; // Выводит 47
echo $A["Petrov"]["email"]."  
"; // Выводит petrov@mail.ru
?>
```

Функции для работы с массивами

Рассмотрим некоторые часто используемые функции для работы с массивами.

Функция list()

Предположим, у нас есть массив, состоящий из трех элементов:

```
$names[0] = "Александр";
$names[1] = "Николай";
$names[2] = "Яков";
```

Допустим, в какой-то момент нам нужно передать значения всех трех элементов массива, соответственно трем переменным: **\$alex**, **\$nick**, **\$yakov**. Это можно сделать так:

```
$alex = $names[0];
$nick = $names[1];
$yakov = $names[2];
```

Если массив большой, то такой способ присвоения элементов массива переменным не очень удобен.

Есть более рациональный подход - использование функции [list\(\)](#):

```
list ($alex, $nick, $yakov) = $names;
```

Если нам нужны только "Николай" и "Яков", то мы можем сделать так:

```
list (, $nick, $yakov) = $names;
```

Функция array()

Функция [Array\(\)](#) используется специально для создания массивов. При этом она позволяет создавать пустые массивы. Вот методы использования функции [Array\(\)](#):

```
<?php
// Создает пустой массив:
$arr = array();
// Создает список с тремя элементами. Индексы начинаются с нуля:
$arr2 = array("Иванов", "Петров", "Сидоров");
// Создает ассоциативный массив с тремя элементами:
$arr3 = array("Иванов"=>"Иван", "Петров"=>"Петр", "Сидоров"=>"Сидор");
// Создает многомерный ассоциативный массив:
$arr4 = array("name"=>"Иванов", "age"=>"24", "email"=>"ivanov@mail.ru");
$arr4 = array("name"=>"Петров", "age"=>"34", "email"=>"petrov@mail.ru");
$arr4 = array("name"=>"Сидоров", "age"=>"47", "email"=>"sidorov@mail.ru");
?>
```

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 17

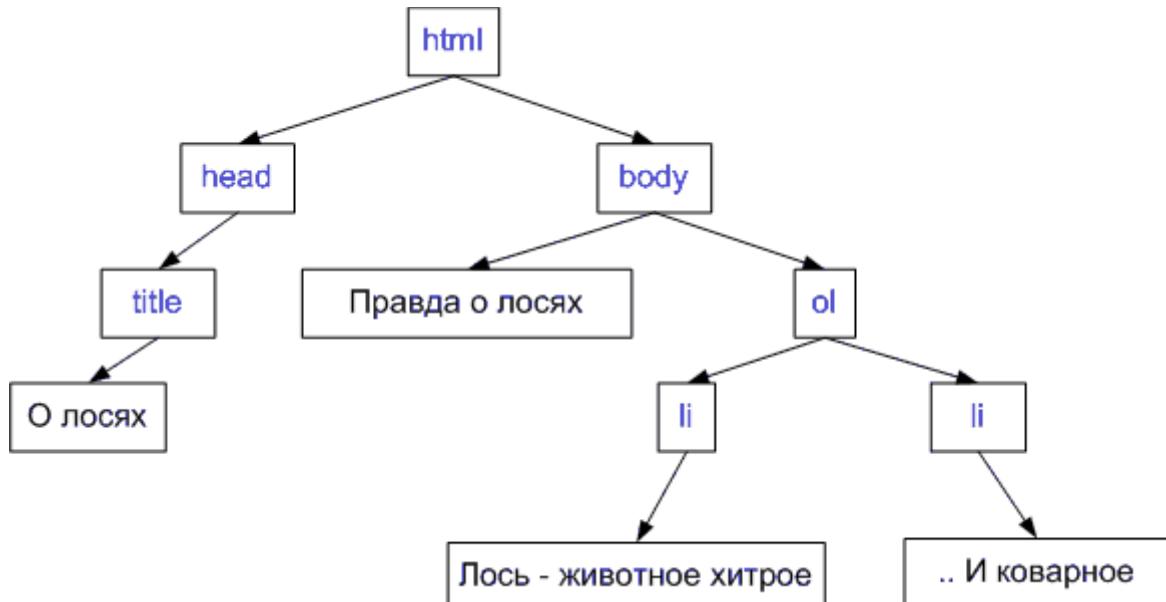
1. Види діяльності на початковій стадії розробки програмної системи.
 2. Модель DOM. Призначення, ступінь підтримки у сучасних програмних засобах.
 3. Проілюструвати взаємозв'язок Web-сторінки з моделлю DOM.
-
1. Етап аналізу – початкова стадія при розробці ПС. Етап полягає у дослідженні системних вимог і проблем, а не в пошуках шляхів їх вирішення. Наприклад, при розробці нової інформаційної системи для комп’ютерної бібліотеки, необхідно описати економічні процеси, пов’язані з її використанням.
На етапі аналізу вимоги замовника уточнюються, формалізуються і документуються. Фактично потрібно отримати відповідь на питання: "**Що повинна робити майбутня ПС?**". Список вимог до розроблюваної ПС повинен включати:
 - **функціональність** - опис виконуваних системою функцій;
 - **контекст** - склад людей та інших систем, що мають безпосередні стосунки з розроблюваною системою;
 - інтерфейси і розподіл функцій між людиною і системою;
 - сукупність умов, при яких передбачається експлуатація ПС (апаратні і програмні ресурси, зовнішні умови функціонування).Метою аналізу є перетворення загальних, неясних знань про вимоги до майбутньої системи в **точні (по можливості) визначення**. На цьому етапі **закладається архітектура** системи.
2. **Document Object Model, DOM** — специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами. Визначається ця специфікація консорціумом W3C. DOM – це фактично представлення документа у вигляді дерева тегів. Це дерево створюється за рахунок вкладеності тегів плюс текстові фрагменти сторінки, кожен з яких утворює окремий вузол. Слугує для представлення для надання XML/HTML -документа і його інтерфейсів таким зовнішнім об’єктам, як, наприклад, JavaScript. Крім того, кожен DOM-елемент є об’єктом і представляє властивості для маніпулювання своїм вмістом, для доступу до батьків і нащадків. Для маніпуляції з DOM використовується об’єкт document. Використовуючи document можна отримати необхідний елемент дерева і змінити його вміст. На даний момент модель DOM має 3 специфікації:
Рівень 1: базові можливості такі як отримання дерева вузлів, можливість додавати та змінювати данні.
Рівень 2: підтримка так званого простору імен XML <--filtered views--> і простору імен
Рівень 3: складається з 6 різних специфікацій, що являються додатковими розширеннями DOM.
Різноманітні браузерні «двигки» підтримують стандарти DOM на різних рівнях, що іноді призводить до труднощів HTML-верстки і низької захищеності сторінок.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>
```

```

        О лосях
    </title>
</head>
<body>
    Правда о лосях.
    <ol>
        <li>
            Лось - животное хитрое
        </li>
        <li>
            .. И коварное
        </li>
    </ol>
</body>
</html>

```



DOCTYPE также является DOM-узлом, и находится в дереве DOM слева от HTML.

Например, этот код получает первый элемент с тэгом `ol`, последовательно удаляет два элемента списка и затем добавляет их в обратном порядке:

```

...
<head>
    <title>О лосях</title>
<script>
function go() {
    var ol = document.getElementsByTagName('ol')[0]
    var hiter = ol.removeChild(ol.firstChild)
    var kovaren = ol.removeChild(ol.firstChild)
    ol.appendChild(kovaren)
    ol.appendChild(hiter)
}
window.onload = function() { document.body.onclick = go }
</script>

</head>
...

```

Аналогично за помощью DOM можно отображать любую информацию про объект, здешнюю модификации атрибутов объектов

Білет №18

Мова UML. Її зв'язок з сучасними підходами до розробки програмних систем.

UML (*Unified Modeling Language*) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної *UML-моделлю*. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

- керівникам проектів, які керують розподілом завдань і контролем за проектом
- проектувальникам інформаційних систем які розробляють технічні завдання для програмістів;
- бізнес-аналітикам, які досліджують реальну систему і здійснюють інженіринг і реінженіринг бізнесу компанії;
- програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Відмінності між різними архітектурними шаблонами.

Архітектурний шаблон — це типова модель, що описує архітектуру (структуру) майбутнього Web-додатку

Архітектурний шаблон визначає

- перелік програмних компонентів
- правила взаємодії між компонентами

«Тонкий» клієнт — це архітектурний шаблон, що надає обмежені можливості по керуванню конфігурацією клієнту (броузеру). Клієнт має лише стандартний броузер, що підтримує роботу з формами. Всі операції, пов'язані з бізнес-логікою, виконуються на сервері. Бізнес-логіка використовується лише при підготовці нової Web-сторінки для клієнта. Клієнт отримує HTML-сторінку, повністю сформовану на сервері. Найчастіше використовується в Інтернет-магазинах з найширшим колом споживачів.

«Тонкий» клієнт

Переваги:

- Можливість використання старих комп'ютерів (не навантажуються ресурси клієнтського комп'ютера)
- Можливе централізоване керування

Недоліки:

- Високі вимоги до серверного забезпечення

- Бідний інтерфейс користувача

«Товстий» клієнт — це архітектурний шаблон, що надає клієнту можливості використання сценаріїв та об'єктів. Основні мови сценаріїв: VBScript, JavaScript, ActionScript. Основні об'єкти: компоненти ActiveX, аплети Java. ActiveX компоненти фактично мають повний доступ до ресурсів клієнта, тому для забезпечення безпеки звичайно вони підписуються незалежними організаціями. Дозволяє розробляти складний інтерфейс користувача.

«Товстий» клієнт

Переваги:

- Дозволяє реалізовувати складний інтерфейс користувача

Недоліки:

- Специфічні вимоги до клієнтського забезпечення
- Складність адміністрування

Web-delivery - Базується на взаємодії клієнту безпосередньо з розподіленими об'єктами на сервери (при цьому протокол HTTP може використовуватись лише як транспорт).

Дозволяє використовувати всі переваги об'єктно-орієнтованого підходу. Використовуються такі протоколи як IOOP та DCOM для підтримки системи розподілених об'єктів.

Засоби взаємодії віддалених компонентів web-додатків.

Web-додаток — це розширення Web-сайту за рахунок використання бізнес-логіки

Web-додаток — це програмна система на базі архітектури “клієнт-сервер”, яка складається з броузера, Web-сервера, що взаємодіють через протокол HTTP та використовують сервер додатків

Web-служба — це серверний об'єкт, що реалізує деяку функціональність, з яким можуть взаємодіяти віддалені програми через протокол HTTP за допомогою XML передачі повідомлень

Дляожної Web-служби потрібна специфікація на **WSDL** (Web Service Description Language — мова опису Web-служб)

Web-служба «спілкується» через протокол **SOAP** (Simple Object Access Protocol — простий протокол доступу до об'єктів)

Переваги Web-служби:

- Незалежність від платформи
- Незалежність від мови
- Уніфікований інтерфейс взаємодії (використання стандартних протоколу HTTP та мови XML)

Білет №19

1. Призначення CASE-засобів та їх зв'язок з ООП Web-додатків.

CASE - Computer-Aided Software Engineering - система автоматизованої розробки програм.

CASE-засоби - програмні засоби, що підтримують процеси створення і супроводу програмних продуктів, включаючи аналіз і формулювання вимог, проектування продукту і баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне управління і управління проектом, а також інші процеси. CASE-засоби разом із системним програмним забезпеченням і технічними засобами утворюють повну середу розробки програмних систем.

Об'єктно-орієнтоване проектування (ООП) (object-oriented design, OOD) - методологія проектування, що сполучає в собі процес об'єктної декомпозиції і прийоми подання логічної і фізичної, а також статичної та динамічної моделей проектованої системи.

Об'єктно-орієнтований аналіз (OOA) (object-oriented analysis) - методологія, при якій вимоги до системи сприймаються з точки зору класів та об'єктів, прагматично виявленіх у предметній області.

Сучасні CASE-засоби охоплюють велику область підтримки численних технологій проектування інформаційних систем (ІС): від простих засобів аналізу і документування до повномасштабних засобів автоматизації, що покривають весь життєвий цикл програмного забезпечення (ПЗ).

Найбільш трудомісткими етапами розробки ІС є етапи аналізу і проектування, у процесі яких CASE-засоби забезпечують якість прийнятих технічних рішень і підготовку проектної документації. При цьому велику роль грають методи візуального представлення інформації. Це припускає побудову структурних чи інших діаграм у реальному масштабі часу, використання різноманітної кольорової палітри, а також перевірку синтаксичних правил. Графічні засоби моделювання предметної області дозволяють розробникам візуально вивчати існуючу ІС, перебудовувати її відповідно до поставлених цілей і наявних обмежень.

2. Технології Java для розробки Web-додатків. (не уверена, но вроде именно это имелось в виду)

Для отримання деяких інтерактивних можливостей веб-додатків у веб-програмуванні використовуються **аплети** - спеціальні Java-додатки, що завантажуються в браузер і працюють в ньому. Основне призначення аплетів - це розширення інтерактивних можливостей для користувача веб-інтерфейсів, у зв'язку з чим **аплети можуть бути використані пристворенні сайтів**.

Аплети підвищують надійність програмного коду сайту, оскільки практично виключають можливість проникнення шкідливих програм. Тому хоча на сьогоднішній день аплети і значно **поступаються за популярністю технології flash**, вони як і раніше можуть застосовуватися веб-програмістами.

Так, наприклад, **аплети дозволяють створювати інтерактивні зручні карти, анімацію, онлайн-ігри, різні веб-форми**. Для вбудовування в html-код сторінок аплетів застосовується спеціальний тег <applet>. У тілі html-сторінки (після відкриваючого тега <body>), даний тег розміщується в тому місці, де за задумом веб-програміста повинен розташовуватися відповідний аплет. **Синтаксис виклику аплету виглядає наступним чином :**

```
<applet code=імя_файла.class width=n height=m> ...</applet>
```

У даному прикладі «імя_файла.class» - це ім'я файлу, що завантажується, що містить аплет. Аплет повинен знаходитися в тій же папці, де і html-сторінка, що посилається на цей аплет.

Аплети можуть містити кілька додаткових необов'язкових параметрів, які можуть бути використані при необхідності. Так, параметр **alt** задає значення для альтернативного тексту у випадку, якщо аплети з яких-небудь причин не відображаються. Параметри **vspace** і **hspace** відповідають за відступи від решти контенту html-сторінки.

Аплети мають надійну систему захисту від впровадження шкідливих кодів і дозволяють звести ризик зараження вірусом до мінімуму. Для цього розробниками Java була створена так звана **«пісочниця» - механізм захисту, що включає три основних компоненти** :

- перевірка на рівні JVM;
- захист на рівні мови;
- інтерфейс JavaSecurity.

Сучасне **веб-програмування** при створенні сайтів часто передбачає використання вже готових аплетів, адже створено таких програм досить багато. Аплети мають кросплатформеність і кросбраузерність, швидко завантажуються. Але є у аплетів і деякі недоліки .

Так, для відображення аплетів браузер повинен підтримувати Java (Java-розширення не завжди за замовчуванням встановлені у браузері). Крім того, для роботи аплетів потрібна

віртуальна машина JVM, тому при першому запуску аплетів необхідно очікувати також запуску JVM.

Веб-програмування сайтів на сьогоднішній день широко використовує сучасні «просунуті» технології, а аплети - це скоріше приємне доповнення до можливостей створюваних сайтів. Тому **аплети присутні найчастіше на розважальних сайтах, інтернет-порталах**.

3. Підтримка математичних операцій у Javascript.

Читать здесь: <http://on-line-teaching.com/js/js.math.htm>

Білlet №2

Каскадний процес розробки. Переваги та недоліки

Основна ідея полягає в тому, що процес розробки ділиться на чітко певні фази, що виконуються строго послідовно.

Класична каскадна модель включає наступні області:

- Розробка вимог: збір бізнес-вимог замовника і їх перетворення в функціональні вимоги до програмного продукту.
- Аналіз та дизайн: розробка моделі предметної області, проектування схеми бази даних, об'єктної моделі, користувальницького інтерфейсу.
- Реалізація: створення продукту по специфікаціям, розробленим на попередньому етапі.
- Тестування: включає перевірку відповідності функціональності програмного продукту потребам користувачів, а також пошук дефектів в реалізації.
- Розгортання: навчання користувачів, іnstалляція системи, переклад в промислову експлуатацію.

У каскадній моделі кожна з процесних областей являє собою окрему фазу проекту.

Переваги:

- Справляється зі складнощами і добре працює для тих проектів, які досить зрозумілі, але все ж важко розв'язні;
- Доступна для розуміння;
- Проста і зручна в застосуванні, так як процес розробки виконується поетапно;
- Представляє собою шаблон, в який можна помістити методи для виконання аналізу, проектування, кодування, тестування і забезпечення;
- При правильному використанні моделі дефекти можна виявити на більш ранніх етапах, коли їх усунення ще не вимагає відносно великих витрат;

Недоліки:

- Процес вкрай неефективний при постійних змінах тренуваній. Кожна зміна змушує повернутися до фази визначення вимог і повторювати весь процес з початку.
- Складно управляти ризиками деяких типів (таких, як ризики, пов'язані з використанням нових технологій або ризики некоректного визначення вимог).

Подібні ризики можуть проявити себе тільки на етапі реалізації, коли число можливих шляхів виправлення ситуації набагато менше, ніж на початку проекту.

- Вельми обмежені можливості оцінки і коректування важливих атрибутів проекту - швидкості розробки, якості продукту, обґрунтованості прийнятих архітектурних рішень.
- виникає необхідність в жорсткому управлінні і контролі, оскільки в моделі не передбачена можливість модифікації вимог;
- всі вимоги повинні бути відомі на початку життєвого циклу
- весь програмний продукт розробляється за один раз. Немає можливості розбити систему на частини.

Архітектурний шаблон Thick Client.

Товстий клієнт — це архітектурний шаблон, що надає клієнту можливості використання сценаріїв на мовах JavaScript, VBScript, або ActionScript та таких об'єктів, як Java-апплети та компоненти ActiveX.

Забезпечує розширену функціональність клієнта, яка не залежить від веб-сервера. Зазвичай, при використанні цього архітектурного шаблону, сервер виконує тільки роль збереження даних, а всі операції по їх обробці виконуються машиною клієнта. Даний архітектурний шаблон дозволяє розробляти складний інтерфейс користувача.

Переваги:

- Має широкий функціонал та мультимедійні можливості
- Має режим багатокористувальської роботи
- Надає можливість роботи навіть при нестабільному зв'язку із сервером
- Висока швидкодія
- Низькі технічні вимоги до серверної машини

Недоліки:

- Великий розмір клієнтського дистрибутиву
- Прив'язаність до платформи, для якої розроблялася клієнтська частина
- Виникають проблеми з віддаленим доступом до даних
- Складність оновлення ПЗ
- Складність актуалізації даних
- Складність адміністрування системи
- Високі технічні вимоги до клієнтської машини

Навести приклад створення об'єкта JavaScript

JavaScript предоставляет разработчикам возможность создавать объекты и работать с ними. Для этого существуют следующие приёмы:

- Оператор new
- Литеральная нотация
- Конструкторы объектов
- Ассоциативные массивы

Используем оператор new

Это, наверное, самый легкий способ создания объекта. Вы просто создаете имя объекта и приравниваете его к новому объекту Javascript.

```
//Создаем наш объект
var MyObject = new Object();
//Переменные
MyObject.id = 5; //Число
MyObject.name = "Sample"; //Строка
//Функции
MyObject.getName = function()
{
    return this.name;
}
```

Минус данного способа заключается в том, что вы можете работать только с одним вновь созданным объектом.

```
//Используем наш объект
alert(MyObject.getName());
```

Литеральная нотация

Литеральная нотация является несколько непривычным способом определения новых объектов, но достаточно легким для понимания. Литеральная нотация работает с версии Javascript 1.3.

```
//Создаем наш объект с использованием литеральной нотации
MyObject = {
    id : 1,
    name : "Sample",
    boolval : true,
    getName : function()
    {
        return this.name;
    }
}
```

Как видите, это довольно просто.

```
Объект = {
идентификатор : значение,
...
}
```

И пример использования:

```
alert(MyObject.getName());
```

Конструкторы объектов

Конструкторы объектов - это мощное средство для создания объектов, которые можно использовать неоднократно. Конструктор объекта - это, по сути, обычная функция Javascript, которой так же можно передавать различные параметры.

```
function MyObject(id, name)
{
```

```
}
```

Только что мы написали конструктор. С помощью него мы и будем создавать наш объект.

```
var MyFirstObjectInstance = new MyObject(5,"Sample");
var MySecondObjectInstace = new MyObject(12,"Othe Sample");
```

Таким образом мы создали различные экземпляры объекта. Теперь мы можем работать отдельно с каждым экземпляром объекта `MyObject`, не боясь того, что, изменяя свойства одного экземпляра, мы затронем свойства другого экземпляра.

Как и в ООП, у `MyObject` могут быть методы и различные свойства. Свойствам можно присвоить значения по умолчанию, либо значения, переданные пользователем в конструкторе объекта.

```
function MyObject(id, name)
{
    //Значения переданные пользователем
    this._id = id;
    this._name = name;
    //Значение по умолчанию
    this.defaultValue = "MyDefaultValue";
}
```

Аналогичным образом мы можем создавать и функции.

```
function MyObject(id,name)
{
    this._id = id;
    this._name = name;
    this.defaultValue = "MyDefaultValue";

    //Получение текущего значения
    this.getDefaultValue = function()
    {
        return this.defaultValue;
    }

    //Установка нового значения
    this.setDefaultValue = function(newvalue)
    {
        this.defaultValue = newvalue;
    }

    //Произвольная функция
    this.sum = function(a, b)
```

```
{  
    return (a+b);  
}  
}
```

С помощью конструкторов можно также создавать [закрытые \(private\) поля](#) класса.

Білет 10

1. Шаблон Creator. Призначення.

Проблема:

Хто відповідальний за створення нових екземплярів деякого класу?

Вирішення: Призначити класу В обов'язок створювати екземпляри класу А, якщо виконується одна із наступних умов:

- Клас В агрегує об'єкти А.
- Клас В містить об'єкти А.
- Клас В записує екземпляри об'єктів А.
- Клас В активно використовує об'єкти А.
- Клас В має дані ініціалізації, які будуть передаватися об'єктам А при їх створенні.

Клас В – створювач (creator) об'єктів А.

Якщо виконується декілька із цих умов, то краще використовувати клас В, що агрегує чи містить клас А.

Переваги: Підтримується шаблон Low Coupling, що сприяє зниженню затрат на супровід і забезпечує можливість повторного використання створених компонентів в подальшому. Використання шаблону Creator не підвищує степені зв'язаності, оскільки створений клас, як правило, виявляється видимим для класа-створювача за допомогою існуючих асоціацій.

2. Призначення підсистеми підтримки сесій у мові програмування PHP.

Протокол HTTP не має вбудованого способу підтримки стану між двома транзакціями. Якщо користувач робить запит на одну сторінку, а потім на іншу, то за допомогою HTTP не можливо встановити, що обидва запити приходять від одного й того ж користувача. Ідея керування сесіями полягає в забезпеченні відслідковування користувача протягом одного сеансу роботи з веб-сайтом. Для керування сесіями в PHP використовується унікальний ідентифікатор сеансу, що представляє собою криптографічно випадкове число. Воно генерується в PHP та зберігається на стороні клієнта протягом всього сеансу. Ідентифікатор сеансу може або зберігатися як cookie-набір або передаватися у складі URL.

Ідентифікатор сеансу грає роль ключа, що забезпечує можливість реєстрації деяких спеціальних змінних в якості так званих змінних сеансу. Їх зміст зберігається на сервері.

Сеанси використовуються для забезпечення механізму авторизації користувачів на веб-сайті, для створення кошика для покупок в інтернет-магазинах.

Щоб розпочати сесію, потрібно викликати функцію `session_start`. Змінні сесії можна зберігати у суперглобальному масиві `$_SESSION` наступним чином: `$_SESSION['key'] = 'value'`. При завершенні сесії необхідно видалити всі сесійні змінні та викликати функцію `session_destroy`.

3. Навести приклад CGI-модуля на мові С.

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
void main(void) {
    int Num; time_t t; srand(time(&t));
    Num = rand()%10;
    printf("Content-type: text/html\n");
    printf("<html><body>");
    printf("<h1>Здравствуйте!</h1>");
    printf("Случайное число в диапазоне 0-9: %d",Num);
    printf("</body></html>");
}
```

Билет №5

1. Требования к системе и описание прецедентов

Функциональные требования к системе определяют что информационная система должна выполнять. При описании функциональных требований информационную систему удобнее всего рассматривать как черный ящик, который обладает требуемыми функциями.

В методе проектирования СОМЕТ функциональные требования задаются с помощью описания всех допустимых прецедентов. При этом под *прецедентом* (use case) понимается некая ситуация единичного или множественных взаимодействий, происходящая между проектируемой системой и пользователем, либо другой системой, внешней по отношению к проектируемой.

Актером, участвующим в прецеденте, называется объект, взаимодействующий с проектируемой системой. В качестве актера могут выступать как пользователи, так и внешние программные или аппаратные системы. Таким образом, понятие *прецедента* можно определить следующим образом: *прецедент* (use case), — это последовательность взаимодействий между одним или несколькими актерами и проектируемой системой. В модели прецедентов *актеры* — единственные внешние сущности, взаимодействующие с системой.

Существует несколько способов моделирования *актеров* [М.Фаулер, К.Скотт 1999]. Обычно, в роли актера выступает либо пользователь либо внешняя система. Но бывают ситуации, особенно характерные для систем реального времени, когда в качестве актера выступает какое-нибудь устройство, например, контроллер ввода/вывода или таймер. Главным актером (primary actor) называется актер, инициирующий прецедент. Остальные актеры называются второстепенными (secondary actor); они также могут участвовать в прецеденте, получая результаты и генерируя исходные данные.

Роль — определенный тип поведения пользователя предметной области, в то время, как *актер* — более широкое понятие, охватывающее *роль*. Таким образом, говоря о *ролях*, подразумевается поведение конкретного пользователя и при этом *роль* правомерно может быть названа *актером*; обратное, как следует из вышеприведенного описания, не всегда верно.

Более известное определение *актеру* дается в нотации UML [Г.Буч, Дж. Рамбо, А. Джекобсон 2000]. В нотации UML *актер* определяется как один или несколько внешних пользователей, взаимодействующих с системой. Такое определение *актера* в нашем понимании однозначно соответствует *ролям*; *актер* же, в нашем понимании, — несколько более широкое понятие.

Прецедент начинается с некоторого действия *актера* и представляет собой завершенную последовательность событий, начатую этим действием и описывающую один из вариантов взаимодействия актера с системой. Очевидно также, что сложность precedента напрямую зависит от количества действий актера. Выявлять precedент легче всего с рассмотрения актеров и инициируемых ими действий.

Результат precedента, который, как правило, получает *главный актер*, является результатом выполнения одной или нескольких функций проектируемой системы. Таким образом, цель выявления и документирования precedента — описание требуемых от проектируемой системы функций, т.е. документирование функциональных требований. "Прозрачная" документация результата precedента — суть каждой диаграммы precedентов, а совокупность описания всех precedентов является собой внешнюю функциональную спецификацию системы.

2) Базовая архитектура веб приложений

Архитектура Web-приложений (рис. 1) наиболее близка к архитектуре централизованных вычислений, т. е. существует множество распределенных «тонких» клиентов, предназначенных исключительно для отображения данных и серверы, которые хранят данные. «Тонкие» клиенты подключаются к серверам. Архитектура Web-приложений отличается от архитектуры централизованных вычислений тем, что Web-приложения работают на основе языка HTML и протокола HTTP [5].

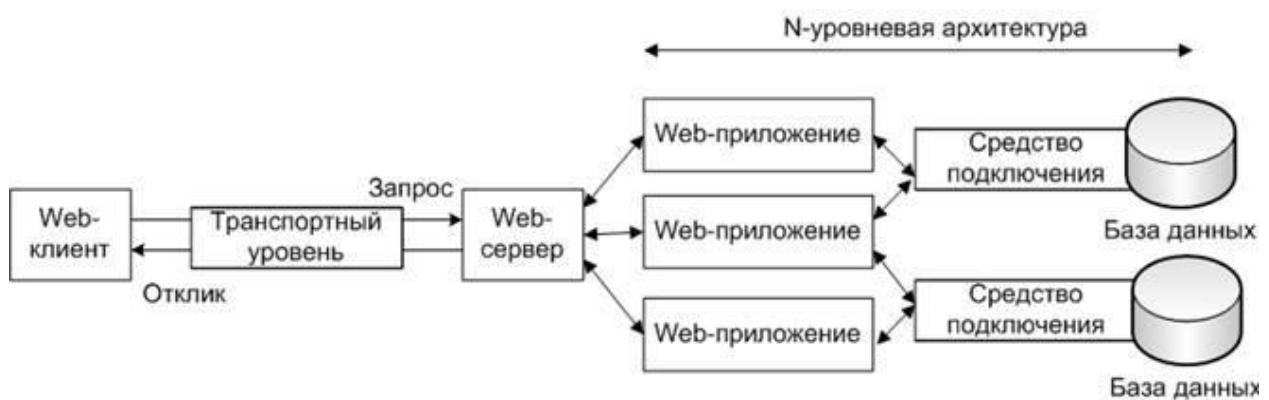


Рис. 1. Архитектура Web-приложения

Таким образом, любая информационная система, создаваемая как web-приложение, должна содержать следующие серверные компоненты:

- web-сервер;
- сервер баз данных;
- сервер приложений и (или) сервер обработки транзакций.

Взаимодействие web-сервера с базами данных может быть организовано двумя способами:

- через сервер транзакций;
- через API или CGI интерфейс web-сервера или сервера приложений.

3) Привести пример сервера веб приложений , объяснить назначение

Сервер приложений ([англ. application server](#)) — это программная платформа ([software framework](#)), предназначенная для эффективного исполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API ([Интерфейс прикладного программирования](#)), который определен самой платформой.

Для веб-приложений эти компоненты обычно работают на той же машине, где запущен веб-сервер. Их основная работа — обеспечивать создание динамических страниц. Однако современные серверы приложений нацелены гораздо больше не на то, чтобы генерировать веб-страницы, а на то, чтобы выполнять такие сервисы как [кластеризация](#), [отказоустойчивость](#) и [балансировка нагрузки](#), позволяя таким образом разработчикам сфокусироваться только на реализации [бизнес-логики](#).

Обычно этот термин относится к [Java](#)-серверам приложений. В этом случае сервер приложений ведет себя как расширенная виртуальная машина для запуска приложений, прозрачно управляя соединениями с базой данных с одной стороны и соединениями с веб-клиентом с другой.

Пример -

[Zope](#) ([англ. Zope Object Publishing Environment](#), среда публикации объектов Zope, произносится [зóуп]) — [объектно-ориентированный сервер приложений](#), написанный на языке программирования [Python](#). Zope разрабатывается на основе собственной [Open Source](#) лицензии [ZPL](#).

Zope (точнее, Zope2) обычно применяется в качестве [системы управления содержимым \(CMS\)](#). Для этих целей был создан [программный каркас CMF](#) ([англ. Content Management Framework](#)) — набор библиотек для создания систем публикаций под Zope. В свою очередь на основе [CMF](#) была создана система публикаций [Plone](#).

В конце 2005 года был выпущен Zope3. Это полностью переработанная версия Zope, которая разрабатывалась на протяжении нескольких лет, и при её разработке учитывался опыт использования Zope2. С выходом Zope3 разработчики получили мощный сервер приложений, с помощью которого стало возможным разрабатывать не только CMS, но и более сложные системы, в частности системы автоматизации бизнес-процессов и документооборота. В январе 2010 года Zope3 был переименован в [BlueBream](#)^[2].

БІЛЕТ №4

1. UML ([Unified Modeling Language](#)) — уніфікована мова моделювання, стандартизований засіб візуалізації структури та функціонування програми за допомогою діаграм UML. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду

можлива кодогенерація. Для всіх моделей застосовується одна і та ж система позначень, але інтерпретація діаграм виконується в різних ракурсах:

- концептуальний (діаграми описують поняття реального світу)
- ракурс специфікації (програмні абстракції і компоненти зі специфікаціями і інтерфейсами (без прив'язки до мови))
- ракурс реалізації (програмні абстракції з прив'язкою до мови)

Призначення UML - дати повне уявлення про архітектуру системи (статична уявлення) і описати принципи її роботи (динамічне представлення системи) шляхом побудови відповідних діаграм.

- **Концептуальна модель.** Модель використовується тільки для полегшення розуміння концепції системи. Такий тип використання моделей один з найважливіших, тому що так використовуються моделі, які виходять в результаті аналізу предметної області. Концептуальні моделі досить стабільні: якщо не змінюється предметна область, то немає потреби міняти і модель. Головне вимоги до моделі предметної області: концептуальна цілісність.
- **Модель проектування.** Модель проектування являє собою високорівневе (на рівні підсистем) і низькорівневе (на рівні класів, якщо мова йде про використання об'єктно-орієнтованого підходу) опис програмної системи. Модель проектування призначена для того, щоб, керуючись нею, розробити програмний код додатку. Як правило, архітектура (високорівневе описание) та код розробляються ітеративно, і розробникам в процесі розробки необхідно часто вносити зміни в модель проектування, щоб вона завжди залишалася актуальною. Головна вимога до моделі проектування: зрозумілість. Дійсно, розробники повинні повністю розуміти модель проектування, щоб вести розробку.
- **Модель реалізації.** Модель реалізації постійно модифікується під час всього часу розробки, вона повинна бути рівною мірою зрозуміла як розробнику, так і комп'ютеру. Таке призначення вимагає вказівки необхідних деталей реалізації в моделі, оскільки самостійно комп'ютер їх додати не зможе. Головна вимога до моделей реалізації: повнота.

2. Шаблон Creator вирішує, хто повинен створювати об'єкт. Більш конкретно, потрібно призначити класу В обов'язок створювати екземпляри класу А, якщо виконується якомога більше з таких умов:

- ї Клас В містить або агрегує об'єкти А.
- ї Клас В записує екземпляри об'єктів А.

ї Клас В активно використовує об'єкти А

ї Клас В володіє даними ініціалізації (has the initializing data), які передаватимуться об'єктам А при їх створенні (тобто при створенні об'єктів А клас В є експертом)

ї Клас В - утворювач об'єктів А.

Переваги шаблону Creator

- 1) Підтримується шаблон Low Coupling, що сприяє зниженню витрат на супровід і забезпечує можливість повторного використання створених компонентів надалі.
- 2) Застосування шаблону Creator не підвищує ступеня пов'язаності, оскільки створений (created) клас, як правило, виявляється видимим для класу-творця допомогою наявних асоціацій.

Основним призначенням шаблону Creator є виявлення об'єкта-творця, який при

виникненні будь-якої події має бути пов'язаний з усіма створеними ним об'єктами. При такому підході забезпечується низька ступінь зв'язаності.

3. Прецедент - це опис поведінки системи, як вона відповідає на зовнішні запити.

Діаграма прецедентів – [діаграма](#), на якій зображені відношення між **виконавцями** та **прецедентами** в системі. Ілюструє набір прецедентів системи та виконавців, а також взаємозв'язки між ними.

Призначення: подати контекстну діаграму, що дозволяє швидко визначити зовнішніх виконавців і основні методи використання системи

БІЛЕТ № 19

1. Призначення CASE-засобів та їх зв'язок з ОOA/П Web-додатків.

Термін CASE (Computer Aided Software Engineering) використовується у досить широкому змісті. Під терміном CASE-засіб розуміються програмні засоби, що підтримують процеси створення й супроводу ІС, включаючи аналіз і формулювання вимог, проектування прикладного ПЗ (додатків) і баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне керування й керування проектом, а також інші процеси. У функції CASE входять засоби аналізу, проектування й програмування. За допомогою CASE автоматизують процеси проектування інтерфейсів, документування й генерування структурованого коду бажаною мовою програмування.

Що ж до зв'язку з об'єкт-орієнтованим аналізом та проектуванням, то найбільш трудомісткими стадіями розроблення ПЗ є саме формування вимог(аналіз) і забезпечення правильності цих вимог(проектування), у процесі яких CASE-засоби забезпечують якість прийнятих технічних рішень і підготовку проектної документації. При цьому велику роль відіграють методи візуального подання інформації. Це передбачає побудову різноманітних графічних моделей, наскрізну перевірку синтаксичних правил тощо. Графічні засоби моделювання ПЗ дають змогу розробникам наочно вивчати функціонуючу ІС, перебудовувати її відповідно до поставлених цілей і обмежень.

2. Технології Java для розробки Web-додатків.

За допомогою технології Java можна додати своїй сторінці елементи інтерактивності, формувати, компонувати й повністю контролювати формат спливаючих вікон і будованих фреймів, організовувати такі активні елементи, як годинник, анімовані рядки та чат. Більшість web-камер, що передають на сайт живе зображення, також працюють на базі відповідних додатків Java.

Останні кілька років розробники докладали масу зусиль, щоб інтегрувати графіку і анімацію в свої аплети і додатки Java. Однак спочатку включені в Java графічні пакети AWT Java мали обмежені кошти для вирішення таких завдань. Тепер же, використовуючи інтерфейси прикладного програмування Java 2D і Java 3D, розробники можуть реалізовувати набагато більш складні графічні додатки, включаючи ігри, зберігачі екрану, екранні заставки та тривимірний графічний користувальницький інтерфейс.

Сервлет є Java-інтерфейсом, реалізація якого розширює функціональні можливості сервера. Сервлет взаємодіє з клієнтами за допомогою принципу запит-відповідь. **Java Servlet API** — стандартизований [API](#) для створення [динамічного контенту](#) до [веб-сервера](#), використовуючи платформу [Java](#). Сервлети — аналог технологій [PHP](#), [CGI](#) і [ASP.NET](#).

Java Server Pages (JSP) забезпечує поділ динамічної та статичної частин сторінки, результатом чого є можливість зміни дизайну сторінки, не зачіпаючи динамічний зміст. Ця властивість використовується при розробці та підтримці сторінок, так як дизайнераам немає необхідності знати, як працювати з динамічними даними.

3. Підтримка математичних операцій у Javascript.

JavaScript підтримує наступні арифметичні оператори:

+	Додавання	$2 + 8$ $7 + 3$	10 10
-	Віднімання	$2 - 8$ $7 - 3$	-6 4
*	Множення	$2 * 8$ $7 * 3$	16 21

/	Ділення	$8 / 2$ $7 / 3$	4 2.3333
%	Ділення за модулем	$8 \% 2$ $7 \% 3$	0 1
+	Однічний інкремент	$8++$ $7++$	9 8
--	Однічний декремент	$8--$ $7--$	7 6

Интересная особенность JavaScript - возможность выполнять арифметические операции над переменными различного типа. В этом случае интерпретатор самостоятельно выполняет приведение типов и выполняет указанную операцию. В процессе ведения типов используются следующие правила:

- Если один из операндов - строка, то все остальные операнды приводятся к строковому виду.

```
var1 = "Дядя"
var2 = "Ваня"

result = var1 + " " + var2
// result = "Дядя Ваня"

mixed = var2 + 100
// mixed = "Ваня100"
```

2. Все логические операнды приводятся к числовому виду, кроме случаев, когда все операнды в выражении логические. При этом true приводится к "1", a false - к "0". При сочетании логических операндов со строками - все операнды переводятся в текстовый вид.

```
var1 = true
var2 = true

result = var1 + var2
// result = 2

mixed = var2 + 100
// mixed = 101

var3 = "строка:"

str = var3 + var1
// str = "строка:true"
```

3. Если приведение типов выполнить не удалось - результатом выражения будет "NaN" (например, при попытке разделить строку на что-либо).

```
var1 = "Дядя"  
var2 = "Ваня"  
  
result = var1 / var2  
  
// result = "NaN"  
  
mixed = var2 * true  
  
// mixed = "NaN"
```

Объект Math содержит основные математические константы и стандартные математические функции. Например ln10(логарифм по основанию 10), pi(значение числа пи), abs(модуль),sqrt(корень),exp(експонента),pow(степень) и др.

Набор математических функций JavaScript позволяет решать довольно большой спектр задач, но злоупотреблять этим не стоит. Не забывайте, что код исполняется инетрпретатором, а вот о низкоуровневой оптимизации вычислений нет и речи, следовательно высокого быстродействия добиться будет очень сложно.

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 19

1. Призначення CASE-засобів та їх зв'язок з ОOA/П Web-додатків.

CASE ([англ. Computer-Aided Software Engineering](#)) — набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов.^[1] Также под CASE понимают совокупность методов и средств проектирования информационных систем с использованием CASE-инструментов.

CASE-средства можно классифицировать в соответствии с их целевым назначением по нескольким направлениям.

Поддерживающие жизненный цикл систем разработки CASE-средства подразделяются, например, на две категории:

- **CASE-средства «верхнего уровня»** (высокоуровневые) (upper CASE tools) поддерживают анализ и проектирование. В основном они используются при анализе и документировании требований пользователей. Они, прежде всего, подходят для визуализации, для создания различных схем и генерирования документации. Они поддерживают использование традиционных языков диаграмм (диаграммы сущность-связь, моделей данных, UML-схемы и т.д.).
- **CASE-средства «низкого уровня»** (низкоуровневые) (lower CASE tools) сосредотачиваются на тех реализациях, в которых из моделей может быть создан реальный программный продукт. Они поддерживают генерирование структуры базы данных, генерирование кода, проведение тестирования, управление версиями кода, управление конфигурацией, реверсивное проектирование и тому подобное.

С самого начала CASE-технологии развивались с целью преодоления ограничений при использовании структурной методологии проектирования (сложности понимания, высокой трудоемкости и стоимости использования, трудности внесения изменений в проектные спецификации и т.д.) за счет ее автоматизации и интеграции поддерживающих средств. Таким образом, CASE-

технологии не могут считаться самостоятельными, они только обеспечивают, как минимум, высокую эффективность их применения, а в некоторых случаях и принципиальную возможность применения соответствующей методологии. Большинство существующих CASE-систем ориентировано на автоматизацию проектирования программного обеспечения и основано на методологиях структурного (в основном) или объектно-ориентированного проектирования и программирования, использующих спецификации в виде диаграмм или текстов для описания системных требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств. В последнее время стали появляться CASE-системы, уделяющие основное внимание проблемам спецификации и моделирования технических средств.

Наибольшая потребность в использовании CASE-систем испытывается на начальных этапах разработки, а именно на этапах анализа и спецификации требований к ЭИС. Это объясняется тем, что цена ошибок, допущенных на начальных этапах, на несколько порядков превышает цену ошибок, выявленных на более поздних этапах разработки.

CASE - технология в рамках методологии включает в себя методы, с помощью которых на основе графической нотации строятся диаграммы, поддерживаемые инструментальной средой.

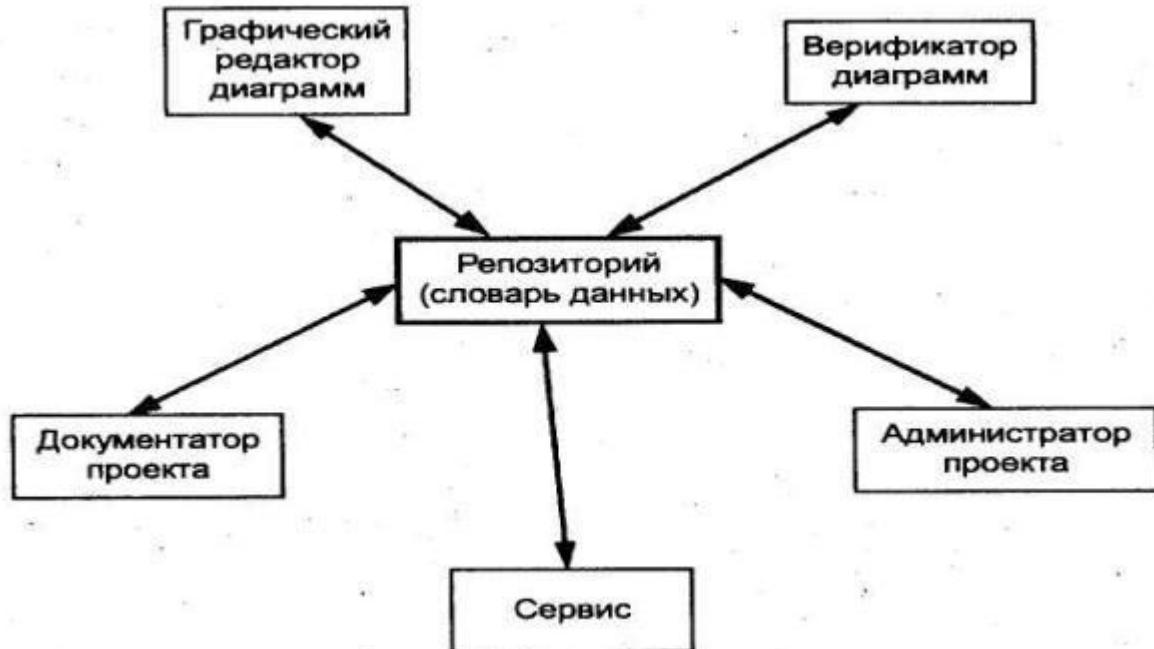
Методология определяет шаги и этапность реализации проекта, а также правила использования методов, с помощью которых разрабатывается проект.

Метод - это процедура или техника генерации описаний компонентов ЭИС (например, проектирование потоков и структур данных).

Нотация - отображение структуры системы, элементов данных, этапов обработки с помощью специальных графических символов диаграмм, а также описание проекта системы на формальных и естественных языках.

Инструментальные средства CASE - специальные программы, которые поддерживают одну или несколько методологий анализа и проектирования ИС.

Рассмотрим архитектуру CASE-средства, которая представлена нарис. 13.1.



2. Технології Java для розробки Web-додатків.

Технология Java Servlets

Технология Java Servlets (сервлеты) была разработана компанией Sun Microsystems, чтобы использовать преимущества платформы Java для решения проблем технологии CGI и API расширений сервера. Технология решает проблему производительности, выполняя все запросы как нити в одном процессе. Сервлеты также могут легко разделять ресурсы, и не зависят от платформы, поскольку выполняются внутри Java Virtual Machine (JVM).

Технология обладает широкими функциональными возможностями. Большое количество библиотек предоставляет самые разнообразные средства, необходимые в разработке. Модель безопасности Java делает возможным точное управление уровнем доступа, например позволяя доступ только к определенной части файловой системы. Обработка исключений Java делает сервлеты более надежным средством, чем расширения серверов на C/C++.

Любой сервлет является классом Java, и, поэтому, должен быть выполнен внутри Java VM так называемым сервлет - контейнером (servlet container, servlet engine). Сервлет - контейнер загружает класс сервлета при первом обращении к нему, либо сразу при запуске сервера при специальном указании. Далее сервлет остается загруженным для обработки запросов, пока он не выгружается явным образом, либо до остановки контейнера.

Технология является распространенной, и может быть использована со всеми популярными Web - серверами (Enterprise Server от Netscape, Microsoft Internet Information Server (IIS), Apache, Java Web Server от Sun).

Программный интерфейс позволяет сервлетам обрабатывать запросы на любом уровне, при необходимости используя любые низкоуровневые данные, такие как заголовки запросов, их тип, и т.д. Это дает большую гибкость при разработке нестандартных обработчиков, например при работе с двоичным или мультимедийным содержимым.

Поскольку сервлеты обрабатываются в одном процессе с помощью создания потоков внутри него, программный код сервлетов должен быть потоке - безопасным. Это накладывает определенную ответственность на программиста, но с помощью стандартных приемов, таких как отказ от использования полей в классах сервлетов, и хранение необходимых данных в контексте или внешнем хранилище такие свойства кода легко достигаются. При этом сервлеты приобретают такое неоценимое преимущество как масштабируемость.

Итак, сервлеты обеспечивают компонентный, платформе - независимый метод для построения web-приложений без ограничений производительности CGI программ. Они имеют широкий диапазон доступных прикладных API, позволяют использовать все преимущества Java, легко расширяются и масштабируются, поддерживаются всеми популярными Web - серверами. Все это делает их отличным средством разработки крупных Web - систем.

Технология Java Server Pages

Технология Java Server Pages (JSP) от компании Sun Microsystems явилась надстройкой над технологией Java Servlets, обеспечивающей более быструю и простую разработку web - приложений с помощью применения шаблонного подхода.

Для понимания архитектуры и преимуществ JSP необходимо знать технологию Java Servlets, поскольку они тесно связаны. Страницы Java Server Pages представляют из себя шаблоны страниц HTML, схожие с шаблонами PHP и ASP. Основным отличием от других подобных технологий является то, что код, находящийся внутри специальных тэгов не интерпретируется при обращении к странице, а предварительно компилируется в Java Servlet. Статические участки шаблона преобразуются в вызовы к функциям для их помещения в поток вывода. Код компилируется так, как если бы он находился внутри сервлета. Компиляция JSP страниц в сервлеты является трудоемкой, но проводится один раз - либо при первом обращении к странице, либо при запуске сервлет - контейнера.

Технология JSP удачно объединяет шаблонный подход к построению сайтов и все преимущества Java платформы. Благодаря этому технология получила широкое распространение как среди профессиональных коммерческих разработчиков, так и при создании открытых бесплатных проектов. Важным шагом к расширению шаблонного подхода стали так называемые библиотеки тэгов (tag libraries). Это гибкая возможность интегрировать стандартные, сторонние, или собственные программные компоненты в страницы. Простота создания и использования привели к большой популярности библиотек тэгов.

Благодаря работе на основе Java технология JSP не привязана к конкретной аппаратной или программной платформе. Таким образом JSP являются отличным решением для использования в гетерогенных средах.

Производительность технологии ограничена объективными особенностями архитектуры. Во-первых, страницы должны быть откомпилированы в сервлеты, что занимает значительное время. Во-вторых сервлеты выполняются в среде выполнения Java, т.е. в режиме интерпретации. Однако эти ограничения компенсируются дополнительными возможностями. Современные контейнеры поддерживают кластеризацию серверов, что перекладывает нагрузку на аппаратное обеспечение. Это является экономически оправданным и простым решением. Задача же компиляции в сервлеты является разовой и производится либо при первом обращении, либо при запуске сервлет - контейнера. Таким образом это не оказывается на общей производительности системы при рассмотрении за достаточный период времени.

Основными достоинствами JSP является простота разработки, характерная для шаблонного подхода, наличие большого количества сторонних библиотек, легкость их использования, мощные и разнообразные среды разработки. Благодаря всем этим факторам JSP является наиболее перспективной базовой технологией разработки при создании Web - сайтов. Однако при создании сложных Web - систем ограничения, накладываемые шаблонным подходом становятся серьезным препятствием к развитию.

3. Підтримка математичних операцій у Javascript.

Числовые операции

умножения (*);

деления (/);

сложения (+);

вычитания (-);

инкремент (++);

декремент (--);

Объекты Math и Number

Объект Math предоставляет математические функции: floor, round, max, min и т.д.

например, Math.cos(x)

Объект Number обладает следующими полезными свойствами

Property	Meaning
MAX_VALUE	Largest representable number
MIN_VALUE	Smallest representable number
NaN	Not a number
POSITIVE_INFINITY	Special value to represent infinity
NEGATIVE_INFINITY	Special value to represent negative infinity
PI	The value of π

(Это все что есть в презенташте)

JavaScript поддерживает следующие арифметические операторы:

+	(плюс)	Сложение	2 + 8 7 + 3	10 10
-	(минус)	Вычитание	2 - 8 7 - 3	-6 4
*	(звёздочка)	Умножение	2 * 8 7 * 3	16 21
/	(слэш)	Обычное деление	8 / 2 7 / 3	4 2.3333
%	(процент)	Деление по модулю	8 % 2 7 % 3	0 1

++ (два плюса)	Единичный инкремент	8++ 7++	9 8
-- (два минуса)	Единичный декремент	8-- 7--	7 6

Интересная особенность JavaScript - возможность выполнять арифметические операции над переменными различного типа. В этом случае интерпретатор самостоятельно выполняет приведение типов и выполняет указанную операцию. В процессе ведения типов используются следующие правила:

1. Если один из operandов - строка, то все остальные operandы приводятся к строковому виду.

```
var1 = "Дядя"
var2 = "Ваня"
result = var1 + " " + var2
// result = "Дядя Ваня"
mixed = var2 + 100
// mixed = "Ваня100"
```

2. Все логические operandы приводятся к числовому виду, кроме случаев, когда все operandы в выражении логические. При этом true приводится к "1", а false - к "0". При сочетании логических operandов со строками - все operandы переводятся в текстовый вид.

```
var1 = true
var2 = true
result = var1 + var2
// result = 2
mixed = var2 + 100
// mixed = 101
var3 = "строка:"
str = var3 + var1
// str = "строка:true"
```

3. Если приведение типов выполнить не удалось - результатом выражения будет "NaN" (например, при попытке разделить строку на что-либо).

```
var1 = "Дядя"
var2 = "Ваня"
result = var1 / var2
// result = "NaN"
mixed = var2 * true
// mixed = "NaN"
```

Однако на начальном этапе лучше воздержаться от приведения типов и фокусов с преобразованием результатов. Это избавит вас от значительного числа ошибок.

Объект Math

Объект Math содержит основные математические константы и стандартные математические функции. Наиболее часто используемые приведены в таблице:

Свойства	
LN10	Значение натурального логарифма числа 10
LN2	Значение натурального логарифма числа 2
PI	Значение числа Пи

Методы	
abs(число)	Возвращает абсолютное значение числа (т.е. число без учёта его знака)
ceil(число)	Откругляет число до ближайшего большего целого (округление "вверх")
exp(число)	Возвращает число "e" в степени "число"
floor(число)	Откругляет число до ближайшего меньшего целого (округление "вниз")
max(число1, число2)	Возвращает большее из двух чисел
min(число1, число2)	Возвращает меньшее из двух чисел
pow(число1, число2)	Возвращает "число1", возведённое в степень "число2"
random()	Возвращает случайное число в диапазоне от 0 до 1
round(число)	Округляет число в соответствии со стандартными правилами округления
sqrt(число)	Возвращает квадратный корень числа.

Из всех перечисленных функций имеет смысл дополнительно пояснить только ceil(), floor() и round(). Рассмотрим их отличия на примере:

```
num = 1.222
// ближайшее целое "вниз" - 1
// ближайшее целое "вверх" - 2
// арифметически откругляется до 1
```

```
alert(Math.ceil(num))
alert(Math.floor(num))
alert(Math.round(num))
```

// получим три сообщения: 2, 1, 1

```
num = 1.777
// ближайшее целое "вниз" - 1
// ближайшее целое "вверх" - 2
// арифметически откругляется до 2
```

```
alert(Math.ceil(num))
alert(Math.floor(num))
alert(Math.round(num))
```

// получим три сообщения: 2, 1, 2

Набор математических функций JavaScript позволяет решать довольно большой спектр задач, но злоупотреблять этим не стоит. Не забывайте, что код исполняется инетрпретатором, а вот о низкоуровневой оптимизации вычислений нет и речи, следовательно высокого быстродействия добиться будет очень сложно.

Чаще всего математика на сайтах используется для создания различных калькуляторов или расчёта положения элементов интерфейса. В примере к данному уроку мы рассмотрим скрипт для создания простого калькулятора.

Билет 20

1)

Артефакты

В начале дадим определение: артефакт (artifact) - это диаграмма, документ, модель, закон и т. д. - нечто, описывающее определенное понятие предметной области. Теперь давайте выясним какие производственные процессы выполняются при разработке программного обеспечения , и какие артефакты при этом разрабатываются. С каждой деятельностью в Рациональном Унифицированном Процессе связаны артефакты, которые либо подаются на вход, либо получаются на выходе. Артефакты используются как исходные данные для последующей деятельности, содержат справочные сведения о проекте или выступают в роли поставляемых по контракту составляющих.

Модели

Модели - это самый важный вид артефактов в Рациональном Унифицированном Процессе. Модель (model) - это упрощение реальности; она создается для лучшего понимания разрабатываемой системы. В Рациональном Унифицированном Процессе имеется девять моделей, которые совместно охватывают все важнейшие решения относительно визуализации, специфирования, конструирования и документирования программной системы:

- модель бизнес-процессов - формализует абстракцию организации;
- модель предметной области - формализует контекст системы;
- модель прецедентов - формализует функциональные требования к системе;
- аналитическая модель (необязательная) - формализует идею проекта;
- проектная модель - формализует словарь предметной области и области решения;
- модель процессов (необязательная) - формализует механизмы параллелизма и синхронизации в системе;
- модель развертывания - формализует топологию аппаратных средств, на которых выполняется система;
- модель реализации - описывает части, из которых собирается физическая система;
- модель тестирования - формализует способы проверки и приемки системы.

Вид - это одна из проекций модели. В Рациональном Унифицированном Процессе существует пять тесно связанных друг с другом видов системной архитектуры, о которые мы рассматривали на предыдущей лекции (см. лекцию 3): с точки зрения проектирования, процессов, развертывания, реализации и прецедентов.

Другие артефакты

Артефакты в Рациональном Унифицированном Процессе подразделяются на две группы: административные и технические. Технические артефакты, в свою очередь, делятся на четыре большие подгруппы:

- группа требований - описывает, что система должна делать;
- группа проектирования - описывает, как система должна быть построена;
- группа реализации - описывает сборку разработанных программных компонентов;
- группа развертывания - содержит все данные, необходимые для конфигурирования предоставленной системы.

В группу требований включается информация о том, что система должна делать. В составе этих артефактов могут быть модели прецедентов, нефункциональных требований, предметной области и иные формы выражения потребностей пользователя, в том числе макеты, прототипы интерфейсов, юридические ограничения и т. д.

Группа проектирования содержит информацию о том, как система должна быть построена с учетом ограничений по времени и бюджету, наличия унаследованных систем, повторного использования, требований к качеству и т. д.

Сюда относятся проектная модель, модель тестирования и иные формы выражения потребностей пользователя, в том числе прототипы и исполняемые архитектуры.

К группе реализации относится вся информация о программных элементах, из которых состоит система, в том числе исходный код на различных языках программирования, конфигурационные файлы, файлы данных, программные компоненты и т.д., а также информация о том, как собирать систему.

2)

Протокол HTTP и способы передачи данных на сервер

Internet построен по многоуровневому принципу, от физического уровня, связанного с физическими аспектами передачи двоичной информации, и до прикладного уровня, обеспечивающего интерфейс между пользователем и сетью.

HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) – это протокол прикладного уровня, разработанный для обмена гипертекстовой информацией в Internet. HTTP предоставляет набор методов для указания целей запроса, отправляемого серверу. Эти методы основаны на дисциплине ссылок, где для указания ресурса, к которому должен быть применен данный метод, используется универсальный идентификатор ресурсов (Universal Resource Identifier) в виде местонахождения ресурса (Universal Resource Locator, URL) или в виде его универсального имени (Universal Resource Name, URN).

Сообщения по сети при использовании протокола HTTP передаются в формате, схожем с форматом почтового сообщения Internet (RFC-822) или с форматом сообщений MIME (Multipurpose Internet Mail Exchange).

HTTP используется для коммуникаций между различными пользовательскими программами и программами-шлюзами, предоставляющими доступ к существующим Internet-протоколам, таким как SMTP (протокол электронной почты), NNTP (протокол передачи новостей), FTP (протокол передачи файлов), Gopher и WAIS. HTTP разработан для того, чтобы позволять таким шлюзам через промежуточные программы-серверы (proxy) передавать данные без потерь.

Протокол реализует *принцип запрос/ответ*. Запрашивающая программа – клиент инициирует взаимодействие с отвечающей программой – сервером и посыпает запрос, содержащий:

метод доступа;

адрес URI;

версию протокола;

сообщение (похожее по форме на MIME) с информацией о типе передаваемых данных, информацией о клиенте, пославшем запрос, и, возможно, с содержательной частью (телом) сообщения.

Ответ сервера содержит:

строку состояния, в которую входит версия протокола и код возврата (успех или ошибка); сообщение (в форме, похожей на MIME), в которое входит информация сервера, метаинформация (т.е. информация о содержании сообщения) и тело сообщения.

В протоколе не указывается, кто должен открывать и закрывать соединение между клиентом и сервером. На практике соединение, как правило, открывает клиент, а сервер после отправки ответа инициирует его разрыв.

Давайте рассмотрим более подробно, в какой форме отправляются запросы на сервер.

Форма запроса клиента

Клиент отсылает серверу запрос в одной из двух форм: в полной или сокращенной. Запрос в первой форме называется соответственно *полным запросом*, а во второй форме – *простым запросом*.

Простой запрос содержит метод доступа и адрес ресурса. Формально это можно записать так:

```
<Простой-Запрос> := <Метод> <символ пробел>
                           <Запрашиваемый-URI> <символ новой строки>
```

В качестве метода могут быть указаны *GET*, *POST*, *HEAD*, *PUT*, *DELETE* и другие. О наиболее распространенных из них мы поговорим немного позже. В качестве запрашиваемого URI чаще всего используется *URL*-адрес ресурса.

Пример простого запроса:

```
GET http://phpbook.info/
```

Здесь *GET* – это метод доступа, т.е. метод, который должен быть применен к запрашиваемому ресурсу, а <http://phpbook.info/> – это *URL*-адрес запрашиваемого ресурса. Полный запрос содержит строку состояния, несколько заголовков (заголовок запроса, общий заголовок или заголовок содержания) и, возможно, тело запроса. Формально общий вид *полного запроса* можно записать так:

```
<Полный запрос> := <Строка Состояния>
  (<Общий заголовок> | <Заголовок запроса> |
   <Заголовок содержания>)
   <символ новой строки>
   [<содержание запроса>]
```

Квадратные скобки здесь обозначают необязательные элементы заголовка, через вертикальную черту перечислены альтернативные варианты. Элемент *Строка состояния* содержит *метод запроса* и *URI* ресурса (как и *простой запрос*) и, кроме того, используемую версию протокола *HTTP*. Например, для вызова внешней программы можно задействовать следующую строку состояния:

```
POST http://phpbook.info/cgi-bin/test HTTP/1.0
```

В данном случае используется метод *POST* и протокол *HTTP* версии 1.0.

В обеих формах запроса важное место занимает *URI* запрашиваемого ресурса. Чаще всего *URI* используется в виде *URL*-адреса ресурса. При обращении к *серверу* можно применять как полную форму *URL*, так и упрощенную.

Полная форма содержит тип протокола доступа, адрес *сервера* ресурса и адрес ресурса на *сервере*([рисунок 4.2](#)).

В сокращенной форме опускают протокол и адрес *сервера*, указывая только местоположение ресурса от корня *сервера*. Полную форму используют, если возможна пересылка запроса другому *серверу*. Если же работа происходит только с одним *сервером*, то чаще применяют сокращенную форму.

Рис. 4.2. Полная форма URL

Далее мы рассмотрим наиболее распространенные *методы отправки запросов*.

Методы

Как уже говорилось, любой запрос *клиента* к *серверу* должен начинаться с указания метода. Метод сообщает о цели запроса *клиента*. Протокол *HTTP* поддерживает достаточно много методов, но реально используются только три: *POST*, *GET* и *HEAD*. Метод *GET* позволяет получить любые данные, идентифицированные с помощью *URI* в запросе ресурса. Если *URI* указывает на программу, то возвращается результат работы программы, а не ее текст (если, конечно, текст не есть результат ее работы). Дополнительная информация, необходимая для обработки запроса, встраивается в сам запрос (в строку статуса). При использовании метода *GET* в поле тела ресурса возвращается собственно затребованная информация (текст HTML-документа, например).

Существует разновидность метода *GET* – условный *GET*. Этот метод сообщает *серверу* о том, что на запрос нужно ответить, только если выполнено условие, содержащееся в поле *if-Modified-Since* заголовка запроса. Если говорить более точно, то тело ресурса передается в ответ на запрос, если этот ресурс изменился после даты, указанной в *if-Modified-Since*.

Метод *HEAD* аналогичен методу *GET*, только не возвращает тело ресурса и не имеет условного аналога. Метод *HEAD* используют для получения информации о ресурсе. Это может пригодиться, например, при решении задачи тестирования гипертекстовых ссылок. Метод *POST* разработан для передачи на *сервер* такой информации, как аннотации ресурсов, новостные и почтовые сообщения, данные для добавления в базу данных, т.е. для передачи

информации большого объема и достаточно важной. В отличие от методов *GET* и *HEAD*, в *POST* передается тело ресурса, которое и является информацией, получаемой из полей форм или других источников ввода.

3) **Сервлеты** - это альтернатива CGI-программирования. Сервлеты выполняются на Web-сервере и играют роль серверов промежуточного уровня.

- Чтение данных, передаваемых пользователем.
- Извлечение прочей (служебной) информации из HTTP-запроса.
- Генерация результатов (обращение к базе данных (JDBC), вызов методов RMI, COBRA).
- Форматирование результатов обработки.
- Установка параметров HTTP-ответа.
- Передача документа клиенту (браузеру).

Преимущества

- ◆ Эффективность (1 копия сервлета используется для обработки *N* запросов и *N* потоков).
- ◆ Переносимость (generic Java property).
- ◆ Богатые возможности (простота взаимодействия с Web-сервером).

```
import java.io.*;          // класс PrintWriter  
import javax.servlet.*;    // HttpServlet  
import javax.servlet.http.*; // HttpServletRequest  
                           // HttpServletResponse
```

```
public class ServletTemplate  
    extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
        PrintWriter out=response.getWriter();  
        out.println("Hello World");  
    }  
}
```

- ◆ Параметр **request** используется для чтения полей заголовка HTTP-запроса, и данных пользователя из формы.
- ◆ **response** используется для установки кода состояния HTTP-ответа и полей заголовка.
- ◆ Объект **out** используется для передачи данных браузеру.
- ◆ Класс, который реализует сервлет, должен быть подклассом класса **HttpServlet**. В нем переопределяются методы **doGet** и **doPost**.
- ◆ При вызове этих методов передается два параметра: экземпляры **HttpServletResponse** и **HttpServletRequest**. С помощью **HttpServletResponse** можно организовать передачу данных клиенту, т.е. создать объект **PrintWriter**. Генерация выходных данных осуществляется посредством методов **print()**/**println()**.
- ◆ Метод

```
public void init() throws ServletException  
{ }
```

вызывается при создании сервлета для чтения параметров сервера.

- ◆ Каждый раз, когда сервер принимает запрос, он порождает новый поток, в котором вызывается метод **service**. Этот метод проверяет типы запроса и вызывает соответствующий метод (**doGet**, **doPost**, **doPut**, **doDelete**).
- ◆ Можно переопределить сам метод **service()**.
- ◆ Метод **destroy()** используется для завершения работы сервлета.

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 21

4. Відмінності серверу додатків від Web-серверу.
5. Призначення моделі предметної області.
6. Масиви у мові PHP.

1) **Веб-сервер** — сервер, отвечающий за прием и обработку запросов (HTTP-запросов) от клиентов к веб-сайту. В качестве клиентов обычно выступают различные веб-браузеры. В ответ веб-сервер выдает клиентам HTTP-ответы, в большинстве случаев, вместе с HTML-страницей, которая может содержать: всевозможные файлы, какие-то изображения, медиа-поток или любые другие данные.

Также веб-сервер выполняет исполнение скриптов, например, таких как CGI, JSP, ASP и PHP, которые отвечают за организацию запросов к сетевым службам, базам данных, доступу к файлам, пересылке электронной почты и другим приложениям.

Распространенные веб-серверы: **Apache, nginx**

Сервер приложений — программная платформа, предназначенная для эффективного исполнения процедур (программ, механических операций, скриптов), которые поддерживают построение приложений. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API, который определен самой платформой.

Пример: JBoss Application Server.

Отличия:

Понятие веб-сервера относится скорее к способу передачи данных (конкретно, по протоколу HTTP), в то время как понятие сервера приложений относится к способу выполнения этих самых приложений (конкретно, удаленная обработка запросов клиентов при помощи каких-то программ, размещенных на сервере). Эти понятия нельзя ставить в один ряд. Они обозначают разные признаки программы. Какие-то программы отвечают только одну признаку, какие-то - нескольким сразу. Tomcat умеет выполнять приложения? Да, он является сервером приложений. Tomcat умеет отдавать данные по HTTP? Да, он является веб-сервером. От того, что Tomcat умеет отдавать данные и по другим протоколам, он не перестает быть и веб-сервером в тот числе.

Возьмите какую-нибудь БД, в которой на хранимых процедурах описана сложная логика. Тем же ораклом можно в ответ на SQL-запросы даже смски через email отправлять. Такую штуку можно назвать сервером приложений, но веб-сервером уже нет, потому что все это не работает с клиентом по HTTP протоколу.

Возьмите чистый апач, в котором не включены никакие модули для поддержки языков программирования. Он умеет отдавать только статичные файлы и картинки по протоколу HTTP. Это веб-сервер, но не сервер приложений. Включите модуль для поддержки PHP и разместите там программу на PHP, которая делает запросы к БД и динамически формирует страницы. Теперь апач стал и сервером приложений.

2) **Модель** - это система, исследование которой служит средством для получения информации о другой системе, это упрощённое представление реального устройства и/или протекающих в нём процессов, явлений.

Модель предметной области — это визуальное представление концептуальных классов или объектов реального мира в терминах предметной области. Моделирование предметной области — один из начальных этапов проектирования системы, необходимый для выявления, классификации и формализации сведений обо всех аспектах предметной области, определяющих свойства разрабатываемой системы.

3) Автомассивы

Отличительной особенностью PHP является то, что он сам может пронумеровывать элементы массива, и следовательно можно создавать массивы на ходу. Вот вам пример автомассива:

```
$a[] = 1;  
$a[] = 2;  
$a[] = 3;
```

Это аналогично такому варианту:

```
$a[0] = 1;  
$a[1] = 2;  
$a[2] = 3;
```

Ассоциативные массивы

Рассмотрим еще один способ присвоения индекса элементу массива. В данном случае мы будем использовать не цифры, как до этого, а буквы. Рассмотрим пример такого массива. Он представляет собой записную книжку, когда по фамилии можно найти имя:

```
$name["Aleynikov"] = "Alan";  
$name["Smirnov"] = "Vanya";  
$name["Nimiroff"] = "Vodka";  
  
// распечатаем чье-то имя  
  
echo $name["Aleynikov"];
```

В качестве результата выводит слово Alan. Задать такой тип массива можно также с помощью **array**:

```
$name = array("ivanov" => "ivan", "petrov" => "petya");
```

Многомерные массивы.

Помимо одномерных массивов в PHP есть также и многомерные массивы.

```
$name["petrov"] = array("name" => "dima", "age" => "18");  
  
$name["alanov"] = array("name" => "alan", "age" => "17");
```

Доступ к элементам осуществляется так:

```
echo $name["petrov"]["age"]; \\ напечатает 18.
```