```НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра обчислювальної техніки

Реєстраційний номер №\_\_\_\_ На правах рукопису

УДК \_\_\_\_\_\_\_\_\_\_\_

«ЗАТВЕРДЖУЮ»

Зав. кафедрою, д. т. н., професор

\_\_\_\_\_\_\_\_\_\_\_\_ (Луцький Г. М.)

(підпис, дата)

**АТЕСТАЦІЙНА БАКАЛАВРСЬКА РОБОТА**

За спеціальністю 6.050102

«Комп’ютерна інженерія»

|  |  |
| --- | --- |
| Виконавець:  Тищенко Денис Олександрович  ФІОТ, гр. ІО-71  № залікової книжки ІО-7120  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис, дата) | Науковий керівник:  Невдащенко Максим Валентинович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис, дата) |

**Київ 2011 р.**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра обчислювальної техніки

|  |  |
| --- | --- |
| Узгоджено  Науковий керівник  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис, дата) | Захищена «\_\_\_» \_\_\_\_\_\_\_\_\_ 2011 г.  З оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**АТЕСТАЦІЙНА БАКАЛАВРСЬКА РОБОТА**

на тему:

**«**Модуль роботи з базою даних для автоматизованої системи складання розкладу**»**

За спеціальністю 6. 050102

«Комп’ютерна інженерія»

|  |  |
| --- | --- |
|  | Виконавець роботи:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис, дата) |

**Київ 2011 р.**

ЗМІСТ

[ВСТУП 3](#_Toc295686339)

[РОЗДІЛ 1 3](#_Toc295686340)

[ТЕОРЕТИЧНІ ВІДОМОСТІ 3](#_Toc295686341)

[1.1.Опис проекту 3](#_Toc295686342)

[1.2.Шаблон MVC 3](#_Toc295686343)

[1.3.Технології клієнт серверної взаємодії в Java 3](#_Toc295686346)

[1.4.Об’єктно реляційне відображення. 3](#_Toc295686353)

[1.5.Бази даних 3](#_Toc295686358)

[ВИСНОВОК ПО РОЗДІЛУ 3](#_Toc295686366)

[РОЗДІЛ 2 3](#_Toc295686367)

[РОЗРОБКА МОДУЛЯ 3](#_Toc295686368)

[2.1 Загальні відомості 3](#_Toc295686369)

[2.2 Опис структури бази даних 3](#_Toc295686370)

[2.3 Розробка програмного забезпечення 3](#_Toc295686374)

[ВИСНОВОК ПО РОЗДІЛУ 3](#_Toc295686381)

[РОЗДІЛ 3 3](#_Toc295686382)

[РЕАЛІЗАЦІЯ МОДУЛЯ 3](#_Toc295686383)

[3.1. Загальні відомості 3](#_Toc295686384)

[3.2. Налаштування бази даних 3](#_Toc295686385)

[3.3. Реалізація програми 3](#_Toc295686388)

[ВИСНОВОК ПО РОЗДІЛУ 3](#_Toc295686391)

[ВИСНОВОК ПО РОБОТІ 3](#_Toc295686392)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 3](#_Toc295686393)

[ДОДАТОК А. ФАЙЛ КОНФІГУРАЦІЇ](#_Toc295686394)

[ДОДАТОК Б. ЛІСТИНГ ПРОГРАМИ](#_Toc295686395)

# ВСТУП

Останні 15 років розвиток Глобальної Мережі відбувався дуже стрімко. З середини 90-х кількість користувачів збільшувалася з кожним роком експоненціально. Популярність цієї служби зростає і до цього дня. Вже практично неможливо порахувати, яку кількість унікальних сторінок розміщено у Всесвітній павутині, однак сумарна цифра обчислюється мільярдами. Інтернет надає такі послуги, як обмін поштою, повідомленнями, файлами, організація аудіо-та відеоконференцій. Для багатьох мільйонів людей Мережа стала основним джерелом інформації та послуг.

Не дивно, що при такому положенні справ кількість різних Інтернет-служб і проектів стала зростати з колосальною швидкістю. Будь-яка хоч скільки-то велика справа (і навіть окремий захід), стосується воно бізнесу, благодійності або навчальної діяльності, має своє відображення в Інтернеті.

Ідея створити Інтернет проект для того, щоб дозволити переглядати і редагувати студентський розклад, є цілком логічною і обгрунтованою у сучасному світі. Такий проект дозволив би публікувати і дивитися в режимі он-лайн розклад предметів, з величезною швидкістю керувати й аналізувати дані, надаючи кожному бажаючому інформацію яка його цікавить.

Звичайно, для реалізації такого проекту потрібно ретельне опрацювання кожної деталі з тією метою, щоб забезпечити надійність, розширюваність, високу швидкість роботи системи. Велика увага повинна бути приділена як для інтерфейсу (адже система розрахована на масове використання), так і функціональному апарату, який повинен справлятися з великою кількістю різних запитів, так і питанню збереження даних, які є найціннішим ресурсом для розкладу. Саме останньому питанню і присвячена дана бакалаврська робота.

# РОЗДІЛ 1

# ТЕОРЕТИЧНІ ВІДОМОСТІ

# Опис проекту

Проект, що розробляється являє собою автоматизовану систему для перегляду і редагування розкладу.

Ціль даного проекту заклечається:

1. Забезпечення одночасної роботи багатьох користувачів.
2. Надійне зберігання великих обсягів даних.
3. Швидкий доступ до даних, малий час реакції.
4. Надання користувачеві можливості перегляду розкладу.
5. Надання можливості редактору редагувати розклад.

Оскільки система повинна бути масовою, логічніше за все її реалізувати у вигляді веб-проекту. З одного боку, схеми і принципи розробки веб-проектів добре вивчені і описані, а, з іншого боку, це дозволить позбутися від необхідності в розробці власної клієнтської програми. Даний проект можна розділити на 3 практично незалежні частини:

1. Слой зберігання даних
2. Слой логіки системи
3. Слой інтерфейсу користувача.

Більш докладно цей підхід, названий шаблоном MVC, розглянутий в наступному розділі. Дана робота представляє собою розробку 1-го шару - зберігання даних.

Варто зауважити, що база даних є вхідним параметром, тобто в даній роботі не розглядається питання проектування бази даних для складання розкладу, але розробляється питання взаємодії з базою через програмний інтерфейс.

# Шаблон MVC

## Загальні відомості

Патерн Модель-представлення-контроллер або по-англійськи Model-view-controller використовується досить довго. Ще в 1979 році його описав Трігве Реенскауг у своїй роботі «Розробка додатків на Smalltalk-80: як використовувати Модель-представлення-контролер ». З тих пір патерн зарекомендував себе як дуже вдала архітектура програмного забезпечення. Отже, «MVC - це архітектура програмного забезпечення, в якій модель даних програми, призначений для користувача інтерфейс і керуюча логіка розділені на три окремі компоненти, так, що модифікація одного з компонентів надає мінімальний вплив на інші компоненти.

Шаблон MVC дозволяє розділити дані, подання та обробку дій користувача на три окремі компоненти:

1. Модель (Model). Модель надає дані (зазвичай для View), а також реагує на запити (зазвичай від контролера), змінюючи свій стан.
2. Представлення (View). Відповідає за відображення інформації (призначений для користувача інтерфейс).
3. Поведінка (Controller). Інтерпретує дані, введені користувачем, та інформує модель і уявлення про необхідність відповідної реакції.

Важливо відзначити, що як уявлення, так і поведінку залежать від моделі. Однак модель не залежить ні від уявлення, ні від поведінки. Це одне з ключових переваг такого поділу. Воно дозволяє будувати модель незалежно від візуального представлення, а також створювати кілька різних уявлень для однієї моделі »[1].

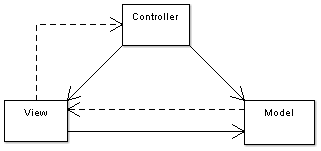


Рис. 1. - Схема MVC

«Основна проблема, яку дозволяє вирішити патерн MVC, полягає в тому, як відокремити один від одного управління програмою (контролер), її зовнішній вигляд (вид), а також внутрішню обробку і механізм ухвалення рішень (модель). При цьому потрібно забезпечити їх представлення у вигляді трьох визначених, відокремлюваних один від одного, компонентів. Розглянемо переваги даного підходу.

По-перше, поділ тягне за собою можливість заміни. Реалізувавши три окремих компонента, можна забезпечити їх несуперечливу взаємодію один з одним. Крім того, при необхідності будь-який компонент можна замінити. Наприклад, якщо б знадобилося, щоб Web-додаток працював з кишеньковими комп'ютерами, а не з Web-браузерами, можна замінити компонент «вид» на інший компонент, за допомогою якого можна було б представляти вміст у форматі кишенькового комп'ютера, а не Web-браузера. Якщо виникне необхідність забезпечити голосове введення даних, а не змушувати користувачів клацати мишкою, можна замінити компонент «контролер» на інший компонент, заснований на засобах голосового введення.

По-друге, пошук важковловимих помилок в коді, а також підтримка програми після її передачі замовнику істотно спрощуються, якщо забезпечено раціональне поділ її логіки. Набагато краще мати безліч невеликих компонентів, які взаємодіють один з одним, ніж невеликий набір великих, громіздких файлів. При такому розділенні компонентів складна архітектура програмної системи стає набагато більш простішою. І нарешті, для однієї моделі можна реалізувати декілька контролерів та компонентів представлення. Повернемося до попереднього прикладу з кишеньковими комп'ютерами при використанні архітектури MVC можна реалізувати окремі контролери та компоненти представлення для кишенькових комп'ютерів, мобільних телефонів і навіть застарілих браузерів. Всі ці компоненти можуть взаємодіяти з однією і тією ж моделлю.

Як підсумок можна сказати, що використання парадигми MVC надає в розпорядження програмісту потужну структуру об'єктів-компонентів, функції яких строго розмежовані, що гарантує надійність і розширюваність системи, що розробляється »[2].

## Реалізація MVC на різних мовах програмування

Запропоноване програмістами Smalltalk-80 рішення виявилося настільки ефективним, що по закінченні вже майже 30 років з моменту своєї появи, шаблон проектування MVC до цих пір є стандартом настільних і Інтернет-програм. У цьому легко переконатися - досить розглянути, наскільки MVC представлений в популярних платформах програмування. Найбільша кількість реалізацій має PHP, але і для Java, Perl, Python, Ruby є свої варіанти. До появи версії MVC від Microsoft, для платформи. NET так само існували свої варіанти: Maverick.NET і MonoRail »[3].

#### PHP

Концепція мови PHP не передбачає ніяких спеціальних засобів для побудови додатків по MVC моделі. Поділ на відображення, контролер і модель є досить умовним і повністю лягає на плечі розробника. Однак, є велика кількість проектів і бібліотек, що надають широкий набір можливостей для реалізації одного з компонентів шаблону MVC. Крім того, існує маса фреймворків для платформи PHP, велика частина яких надає засоби для створення програм на основі MVC шаблону. Серед найбільш поширених фреймоврків можна відзначити наступні:

* Zend Framework;
* CakePHP;
* php.MVC;
* FUSE;
* Symfony.

Однак, існує і маса інших рішень для PHP, що значно спрощують створення MVC додатків.

#### Java

Важливою особливістю Java 2 Enterprise Edition (J2EE) є той факт, що в цій технології визначений шаблон моделі - Beans. Представленням є Java Server Page або, як альтернатива, код сервлета, який генерує подання. Контролером можна вважати сервлет.

Java Swing, на відміну від інших платформ, не тільки надає інтерфейс розробки на основі шаблону MVC, але і сам реалізований на його основі. Представленням є клас - спадкоємець класу JComponent. Внаслідок організації подієвої моделі Java на інтерфейсах, контролер є набір анонімних класів обробки відповідних подій. Як і інші платформи, Swing надає розробку моделі програмісту [3].

#### Ruby

Найбільш популярним і потужним фреймворком, що спрощує розробку MVC додатків, є Ruby on Rails.  
«Ruby on Rails - об'єктно-орієнтований програмний каркас для створення веб-додатків, написаний на мові програмування Ruby. Ruby on Rails надає архітектурний зразок Model-View-Controller (модель-подання-контролер) для веб-додатків, а також забезпечує їх інтеграцію з веб-сервером і сервером бази даних »[4]. Надає однорідне середовище для розробки динамічних AJAX-інтерфейсів, з обробкою запитів і видачі даних в контролерах, відображення предметної області в базі даних. «Ruby on Rails є відкритим програмним забезпеченням та розповсюджується під ліцензією MIT.

Ruby on Rails визначає наступні принципи розробки додатків:

* Програми не повинні визначати власну архітектуру, оскільки вони використовують готовий каркас модель-представлення-контроллер.
* Мова Ruby дозволяє використовувати легко читаєму нотацію для визначення семантики програм (таких як відносини між таблицями в базі даних).
* Ruby on Rails надає механізми повторного використання, що дозволяють мінімізувати дублювання коду в програмах (принцип Don't Repeat Yourself).
* За замовчуванням використовуються угоди по конфігурації, типові для більшості додатків (принцип Convention over configuration). Явна специфікація конфігурації потрібна тільки в нестандартних випадках »[4].

#### .NET

«ASP.NET визначає представлення і контролер, а розробка моделі є завданням розробника. Представлення є набором файлів ASPX і ASCX. Клас подання успадковується від класу контролера на відміну від оригінальної реалізації MVC в Smalltalk, в якій подання і контролер розділені і пов'язані перехресними посиланнями. Контролером є реалізація класів Page і Control, а також обробники подій, описані розробником »[5]. Windows Forms (WinForms), як і ASP.NET, задає тільки представлення і контролер. Класи, успадковані від класів Form або Control, є поданням. На відміну від ASP.NET, яке використовує спадкування подання від контролера, в WinForms дані компоненти компілюються в один клас. Контролером є виконувані на рівні операційної системи генерація і транспорт подій, а також прийом і перенаправлення подій класами Form і Control у відповідні обробники, визначені розробником.

# Технології клієнт серверної взаємодії в Java

## Сервлети

### Загальні відомості

Сервлет - це Java програма, яка, згідно специфікації J2EE, функціонує в web контейнері, динамічно генерує HTML сторінку, XML документ або інший матеріал у відповідь на отриманий від клієнта запит. Оскільки сервлети пишуться на мові Java, сервлет не має прив'язки до будь-якої конкретної платформи або web серверу. Іх можна використовувати на будь-якій платформі, де вдалося запустити віртуальну Java-машину, і на будь-якому web-сервері, що підтримує виконання сервлетів. [6]

У шаблоні MVC сервлети займають місце контролерів. У якості відображення виступає JSP. Даними є POJO - прості java об'єкти. Взаємодія сервлета з клієнтом будується за стандартною схемою запит-відповідь. Графічно взаємодія клієнта - сервера проілюстровано на рисунку 1.



*Рис.1.2.* Взаємодія клієнта з сервером через HTTP протокол.

При цьому сам сервлет безпосередньо з клієнтом не зв'язується, а в ролі посередника, що підтримує зв'язок з віддаленим клієнтом, виступає web-контейнер. Web-контейнер - програма, що працює на web-сервері і виконує певні функції.

Прикладом web-контрейнера для сервлетів є Apache Tomcat. Tomcat (у старих версіях - Catalina) - програма-контейнер сервлетів, написана на мові Java і реалізує специфікацію сервлетів і специфікацію JavaServer Pages (JSP), які є стандартами для розробки веб-програм на мові Java

### Життєвий цикл сервлета

У кожного сервлета один і той же життєвий цикл:

1. Ініціалізація сервлета. Сервер завантажує і ініціалізує сервлет.
2. Виконання сервлета. Сервлет ні кого не обслуговує або обслуговує одного і більше клієнтів.
3. Видалення сервлета. Сервер видаляє сервлет.



*Рис.1.3. Життєвий цикл сервлета*

Розглянемо далі більш детально кожен з етапів життєвого циклу: ініціалізація класу HttpServlet ініціалізує сервлет і ідентифікує ініціалізацію. Щоб проініціалізувати сервлет певним чином, необхідно перевизначити метод init згідно тієї версії JSDK, яку використовуємо: метод init (ServletConfig) для версії 2.0 і метод init () для версії 2.1. При перевизначенні методу init слід дотримуватися наступних правил [7]:

1. У разі виникнення помилки ініціалізації, яка призводить сервлет в стан непрацездатності, слід викликати виключення UnavailableException. Як приклад виникнення такого типу помилок - неможливість встановлення необхідного мережевого з'єднання або з'єднання з базою даних.

2. Не слід викликати метод System.exit.

3. Тільки для версії 2.0: слід зберігати параметр ServletConfig так, щоб метод getServletConfig міг повернути цю величину. Найпростіший спосіб реалізувати це - реалізувати так, щоб новий метод init викликав super.init.

Коли сервлет знаходитися на етапі виконання він може обслуговувати клієнтські запити або у разі їх відсутності нічого не обслуговувати. HTTP сервлет підтримує запити клієнта через метод service. Метод service підтримує стандартні HTTP запити клієнта, відправляючи кожен запит відповідному розробленому для обробки даного запиту методу. Щоб управляти HTTP запитами з сервлета, потрібно наслідувати клас HttpServlet і перевизначити методи сервлета, які управляють HTTP запитами, підтримувані сервлетом. Як приклад можна розглянути два методи взаємодії користувача і сервлета:

• doGet

• doPost

Приклад: форма сторінки html може послати на сервер get або post запит, після того як користувач натисне кнопку submit. Сервер направить запит на обробку в контейнер сервлетів, а той у свою чергу - визначеному сервлету. Управління запитами GET тягне за собою перевизначення методу doGet. Для відповіді, метод doGet використовує об'єкт Writer, отриманий з об'єкта HttpServletResponse, щоб повернути клієнту текстову інформацію. Управління запитами POST тягне за собою перевизначення методу doPost. Для відповіді, приклад методу doPost використовує об'єкт Writer, отриманий з об'єкта HttpServletResponse, щоб повернути клієнту текстову інформацію. При видаленні сервлета контейнером сервлетов повинен бути викликаний відповідний метод. Метод destroy дозволяє класу HttpServlet знищити сервлет.

Щоб знищити будь-які пов'язані з сервлетом ресурси, слід перевизначити метод destroy. Метод destroy повинен скасувати будь-який ініціалізація процес і синхронізувати сталий статус з поточним статусом в пам'яті. Сервер викликає метод destroy після того, як завершаться всі виклики сервісу, або пройде певний проміжок часу встановлюється сервером, що швидше відбудеться. Якщо сервлет виробляє яку-небудь "довгограючу" операцію (операція, час виконання якої набагато більше інших операцій), метод service може продовжувати виконуватися в той момент, коли сервер викличе метод destroy. Розробник відповідальний за те, щоб бути впевненим що, ці процеси повинні бути завершені

## Java Server Faces

### Загальні відомості

JavaServer Faces це фреймворк для веб-програм, написаний на Java. Фреймворк є серверною компонентною інфраструктурою, що полегшує розробкуWEB інтерфейсів на Java. JSF близька до традиційних технологій розробки інтерфейсів, таких як AWT, SWT і Swing. Одним з її основних переваг є те, що програміст звільняється від чорнової роботи, яка лягає на плечі розробників самої інфраструктури JSF. Через це сама JSF виявляється складніше багатьох інших Web-технологій, але її внутрішня структура прихована від очей розробників програм. Створювати Web-програми на основі JSF як правило легше, ніж за допомогою інших каркасів: вони виходять швидше, а також простіше в сенсі структури і конфігурації. [8]

При тому що JSF надає набір стандартних графічних компонент для створення інтерфейсів, JSF орієнтована на створення Web-програм і має такі переваги: [9]

• Чіткий поділ бізнес-логіки та інтерфейсу

• Управління збереженням на рівні компонент

• Проста робота з подіями на стороні сервера

• Використання простих і знайомих концепцій, таких як графічний інтерфейс або шари в Web-програмі.

• Доступність кількох реалізацій від різних компаній-розробників

• Широка підтримка з боку інтегрованих засобів розробки (IDE)

Як правило, програма, створена на основі JSF, складається з наступних частин:

• Об'єктів JavaBean, керуючих станом і поведінкою програми

• Компонентів GUI, з можливістю збереження стану

• Класів, що реалізують подієву модель, наприклад, слухачів (listeners), аналогічних тим, що використовуються при традиційній розробці графічних інтерфейсів

• Сторінок, які виступають в ролі уявлень в парадигмі Модель-Представлення-Контролер (Model-View-Controller - MVC). Подібні сторінки можуть звертатися до кореневих вузлів подання (view roots) через дерево компонентів JSF.

### JSF та MVC

JSF проектувалася з урахуванням досвіду накопиченого протягом декількох років еволюції програмних засобів для розробки Web-програм на Java. Все починалося c JSP - технології, що володіє чималими достоїнствами, але, на жаль, дозволяє дуже легко змішувати код на Java з HTML-вмістом Web-сторінок.

Отже потрібно було розробити такий інструмент, який би міг дозволити розробляти проект, не змішуючи логіку і інтерфейс в одному місці, то є такий інструмент, який би підтримував шаблон MVC, описаний в розділі 1.2.

Спочатку для цієї мети була розроблена «модель 1», що заохочує перенесення Java-коду в компоненти JavaBeans, до яких потім можна було звертатися на сторінці з допомогою тега <jsp:useBean>. Цей підхід непогано працював у випадку простих Web-програм. [9] Для більш складних програм розроблена «модель 2». Відповідно до цієї моделі контролером є сервлети (або Actions) а відображення сторінок реалізується за допомогою JSP-сторінок. Приклад реалізації даної моделі - Apache Struts. Головним недоліком такої моделі була подієва модель - вона носила примітивний характер. Оскільки протокол за яким здійснюється взаємодія клієнта і сервера не передбачає збереження подій (http протокол), вся робота по збереженню стану між кількома запитами користувача лягати повністю на розробника програми. [9]

JSF надає розробникам компонентну модель і значно кращу підтримку MVC, ніж реалізація «модель 2». Незважаючи на те, що JSF спроектована на основі протоколу, який не підтримує збереження станів, вона значно ближче до реальної парадигми MVC, ніж архітектура «модель 2». Крім того, JSF допомагає створювати більш гнучкі, краще орієнтовані на обробку подій графічні інтерфейси, ніж «модель 2». На відміну від простої схеми "запит - обробка", характерної для «модель 2», JSF пропонує повноцінний набір подій, таких як: вибір пункту меню, натискання на кнопку, розкриття вузла в дереві. Гнучко настроюється модель подій в JSF дозволяє додатку менше залежати від деталей HTTP і, як наслідок, спрощує процес розробки. JSF покращує архітектуру додатків у порівнянні з «модель 2», допомагаючи звільнити класи контролера і JSP-сторінки від коду бізнес логіки та інтерфейсу. Класи, що реалізують контролер виходять легковагими і не прив'язаними до JSF взагалі, що дозволяє їх легко тестувати

# Об’єктно реляційне відображення.

## Загальні відомості.

«В об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Як приклад можна розглянути адресну книгу. У термінах об'єктно-орієнтованого програмування вони будуть представлятися об'єктами класу «Людина», які будуть містити наступний список полів: ім'я, список (або масив) телефонів і список адрес. Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відносин між ними. Ці об'єкти називають «постійними» (англ. persistent). Історично існує кілька підходів до вирішення цього завдання. Вирішення проблеми зберігання даних існує - це реляційні системи управління базами даних. Використання реляційної бази даних для зберігання об'єктно-орієнтованих даних приводить до семантичного провалу, тому що розходження в принципах управління даними в об'єктно-орієнтованому та реляційному підходах досить великі ». Розглянемо, які проблеми можуть виникнути при вирішенні зберігати об'єкти в реляційну базу даних:

* проблема гранулярності - рівня деталізації;
* проблема підтипів: успадкування, поліморфізм;
* проблема ідентифікації;
* проблеми, пов'язані з установкою зв'язків між об'єктами.

Все вищесказане змушує програмістів писати програмне забезпечення, яке повинне уміти як обробляти дані в об'єктно-орієнтованому вигляді, так і вміти зберегати ці дані в реляційній формі. Ця постійна необхідність в перетворенні між двома різними формами даних не тільки сильно знижує продуктивність, але і створює труднощі для програмістів, тому що обидві форми даних накладають обмеження один на одного. Крім того, від розробника потрібна висока кваліфікація, так як він повинен вміти професійно працювати з двома парадигмами. Розроблено безліч пакетів, що усувають необхідність у перетворенні об'єктів для зберігання в реляційних базах даних. Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення автоматично. Маючи список таблиць в базі даних і об'єктів у програмі, вони автоматично перетворять запити з одного виду в інший. У результаті запиту деякого об'єкта буде сформований і виконаний необхідний SQL-запит, а результати будуть автоматично перетворені в об'єкти. Основними особливостями ORM є:

* декларативне відображення між об'єктною моделлю і схемою бази даних: класи об'єктної моделі, їх атрибути і відносини відображаються в таблиці і колонки БД за допомогою автоматично створюваних SQL-запитів;
* надання API управління об'єктами в БД: створення, видалення, читання і зміна;
* мова запитів, що використовується для ефективного пошуку персистентних об'єктів, що задовольняють умовам запиту;
* підтримка транзакцій;
* раннє і пізнє завантаження даних;
* кешування;
* незв'язані об'єкти: дозволяє передавати персистентні об'єкти поза рамками сесії;

Отже, схема моделі MVC при використанні ORM-інструментів набуває наступний вигляд:



Рис. 1.4. - Схема MVC при використанні засобів об’єктно орієнтованого відображення.

З точки зору програміста система повинна виглядати як постійне сховище об'єктів. Розробник може створювати об'єкти і працювати з ними як зазвичай, а вони автоматично будуть зберігатися в реляційній базі даних. На практиці все не так просто і очевидно. Всі системи ORM зазвичай проявляють себе в тому чи іншому вигляді, зменшуючи в деякому роді можливість ігнорування бази даних. Більш того, шар транзакцій може бути повільним і неефективним (особливо в термінах згенерованого SQL). Все це може призвести до того, що програми будуть працювати повільніше і використовувати більше пам'яті, ніж програми, написані «вручну». Але ORM позбавляє програміста від написання великої кількості коду, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість розробки. Крім того, більшість сучасних реалізацій ORM дозволяють програмістові при необхідності самому жорстко задати код SQL-запитів, який буде використовуватися при тих чи інших діях (збереження в базу даних, завантаження, пошук і т. д.) з постійним об'єктом.

## JPA

Java Persistence API (скорочено JPA) надає інтерфейс для забезпечення персистентності POJO об'єктів при ORM [11]. Спочатку, JPA був спроектований розробниками EJB як частина специфікації EJB3.0 JSR220, однак використання JPA не обмежується тільки EJB компонентами. Дана технологія може бути використана безпосередньо веб-програмами, прикладними програмами і навіть поза рамками платформи J2EE. Специфікація описує наступні три напрями:

* API (пакет javax.persistance);
* Мова запитів;
* дані для об’єктно-реляційного відображення.

Слід зазначити, що JPA - це тільки лише специфікація, стандарт. Конкретні реалізації можна знайти в таких продуктах, як openJPA, Hibernate та інші. Офіційним продуктом від Sun, реалізують специфікацію JPA, є сервер GlassFish, в якому це досягається шляхом інтеграції рішення TopLink (докладніше про TopLink див. нижче).

## Огляд різних ORM фреймворків

#### OpenJPA

OpenJPA є реалізацією специфікації JPA з відкритим кодом, розробленої групою Apache і розповсюджується під ліцензією Apache 2.0 Licence.

OpenJPA може бути використаний як окремий модуль для забезпечення збереження POJO об'єктів, так і бути вбудованим в будь-який сумісний програмний контейнер і безліч різних фреймворків. Зокрема, на офіційному сайті проекту представлені інструкції з інтеграції з такими серверами, як:

* GlassFish;
* Java System Application Server;
* WebSphere Application Server;
* JOnAS Application Server;
* BEA Weblogic Server;

По-замовчуванню використовується в сервері BEA Weblogic Server.

#### Hibernate

Hibernate - бібліотека для мови програмування Java, призначена для вирішення завдань об'єктно-реляційного проектування. Вона являє собою вільне програмне забезпечення з відкритим кодом, що розповсюджується на умовах GNU Lesser General Public License. Дана бібліотека надає легкий у використанні фреймворк для відображення об'єктно-орієнтованої моделі даних у традиційні реляційні бази даних.

Hibernate забезпечує прозору підтримку збереження даних (persistence) для «POJO» (тобто для стандартних Java-об'єктів). Hibernate використовується як в standalone Java-програмах, так і в програмах на платформі Java EE, що використовують сервлети або EJB.

Hibernate був розроблений командою програмістів на чолі з Гевіном Кінгом (Gavin King), який заснував Hibernate Project, для того щоб подолати кілька недоліків, що існують в EJB 2.x компонентах управління даними. Hibernate тісно пов'язаний з JBoss Group як результат найму фірмою JBoss кількох провідних розробників Hibernate. Зовсім недавно Hibernate був залучений у розвиток персистентних інтегрованих середовищ в Java Community Process (JCP), та багато концепцій Hibernate були введені в специфікацію Java Persistence APIs (JPA). Найостанніша версія Hibernate розроблена у відповідності зі специфікацією JPA.

Hibernate є одним з найбільш популярних ORM-інструментів для Java. По-замовчуванню використовується в сервері JBoss, є широка підтримка у надзвичайно потужному і поширеному фреймворку Spring.

#### Cayenne

Apache Cayenne є ORM-фреймворком з відкритим кодом, поширюваним під ліцензією Apache License. Проект виник в 2001р. і придбав певну популярність в наші дні, однак, за межами країн СНД. Cayenne має широкі можливості у сфері забезпечення збережності об'єктів, зокрема, надає засоби для:

* відображення Java-об'єкта у різні схеми БД;
* управління транзакціями;
* генерації SQL коду;
* віддаленої персистентності об'єктів за допомогою Web служб;
* збереження об'єктів в XML сховища, що дозволяє їх з легкістю використовувати не Java клієнтами: наприклад, AJAX-сумісними браузерами;
* зворотного проектування.

Крім того, значною перевагою Cayenne перед іншими ORM інструментами є наявність графічного засобу для налаштування даного продукту. Розробники Cayenne спочатку поставили перед собою мету позбавити програміста від вивчення чергового формату анотацій і XML-документа.

Підтримка специфікації EJB3.0/JPA знаходиться в стадії розробки і в останній на даний момент версії реалізована лише частково.

#### TopLink

TopLink є ORM-фреймворком для розробок на Java. Існує два варіанти цього продукту:

* TopLink Essentials – проект з відкритим кодом;
* Oracle's TopLink – пропрієтарний продукт.

TopLink Essentials є більш обмеженим у можливостях у порівнянні з пропрієтарної версією. Наприклад, TopLink Essentials не надає засобів по синхронізації кеша в разі кластерної програми, індвідуальної політики кешування та ін. Проект TopLink спочатку розроблявся групою The Object People в SmallTalk в 1990-х. У 1996 почала вводитися підтримка мови Java, а в 2002р. проект був викуплений корпорацією Oracle, де він і продовжує розроблятися як член сімейства продуктів Oracle Fusion Middleware. [15] У 2006р. Oracle відкрила частину вихідних кодів і ресурсів розробників TopLink для відкритого проекту Glassfish від Sun Microsystems. Новий продукт отримав назву TopLink Essentials і був реалізацією EJB3.0/JPA специфікації. У 2007р. Oracle відкрила частину вихідних кодів і ресурсів розробників TopLink для створення відкритого проекту EclipseLink від Eclipse Foundation. У березні 2008р. Eclipse Foundation оголосила про те, що Sun Microsystems вибрав проект EclipseLink в якості основи для складання специфікації JPA 2.0 [13].

## Вибір ORM реалізації

З приходом в Java стандарту EJB3/JPA2.0 більшість розробників ORM інструментів почали забезпечувати відповідність своїх продуктів даному стандарту. Відмінності в плані можливостей забезпечення персистентності об'єктів почали стиратися. Найбільш важливим фактором при виборі ORM інструменту стають особливості реалізації: деякі продукти повільно працюють з великими обсягами даних, інші не здатні забезпечувати швидкі відповіді на великій кількості малих вибірок.

Для роботи проекту потрібно, щоб шар моделі був здатний працювати з великою кількістю необ'ємним вибірок. Тому вибір ліг на Hibernate. Цей інструмент показує відмінні результати саме при невеликих обсягах даних. Hibernate вкрай поширений в Інтернеті - величезна кількість проектів створюється з використанням цього продукту. Тому Hibernate має вельми широку підтримку в Інтернеті, а значить, знайти відповіді на всі питання не складе труднощів. Крім того, організація шару моделі у вигляді зв'язки Hibernate + PostgreSQL є поширеним в Мережі рішенням.

# Бази даних

## Загальні відомості

База Даних (БД) - структурований організований набір даних, що описують характеристики будь-яких фізичних або віртуальних систем. (Пойменована сукупність структурованих даних предметної області). За моделлю представлення даних БД класифікуються:

* Картотеки;
* ієрархічні;
* мережеві;
* реляційні;
* багатовимірні;
* об'єктно-орієнтовані.

На рівні фізичної моделі електронна БД являє собою файл або їх набір у форматі TXT, CSV, Excel, DBF, XML або в спеціалізованому форматі конкретною системою управління базами даних (більш докладно СУБД розглянуті нижче). Також в СУБД в поняття фізичної моделі включають спеціалізовані віртуальні поняття, існуючі в її рамках - таблиця, табличний простір, сегмент, куб, кластер і т. д.

В даний час найбільшого поширення набули реляційні бази даних. Картотеками користувалися до появи електронних баз даних. Мережеві і ієрархічні бази даних вважаються застарілими, об'єктно-орієнтовані поки ніяк не стандартизовані і не набули широкого поширення. Деяке відродження отримали ієрархічні бази даних у зв'язку з появою і поширенням XML. В даний час найбільшого поширення набули реляційні бази даних.

## СУБД

### Загальні відомості

Система управління базами даних - комплекс програмних і лінгвістичних засобів загального або спеціального призначення, який реалізує підтримку створення баз даних, централізованого управління та організації доступу до них різних користувачів в умовах прийнятої технології обробки даних.

СУБД характеризується використовуваною моделлю, засобами адміністрування та розробки прикладних процесів.

СУБД забезпечує:

* опис і стиснення даних;
* маніпулювання даними;
* фізичне розміщення та сортування записів;
* захист від збоїв, підтримку цілісності даних та їх відновлення;
* роботу з транзакціями і файлами;
* безпека даних.

СУБД визначає модель представлення даних.

### Реляційні СУБД

Реляційна СУБД - СУБД, керуюча реляційними базами даних. Поняття реляційний (англ. relation - відношення) пов'язане з розробками відомого англійського фахівця в області систем баз даних Едгара Кодда (Edgar Codd).

Ці моделі характеризуються простотою структури даних, зручним для користувача табличним поданням і можливістю використання формального апарата алгебри відносин і реляційного числення для обробки даних. Реляційна модель орієнтована на організацію даних у вигляді двовимірних таблиць. Кожна реляційна таблиця являє собою двовимірний масив і має наступні властивості:

* кожен елемент таблиці - один елемент даних;
* всі осередки в стовпці таблиці однорідні, тобто всі елементи в стовпці мають однаковий тип (числовий, символьний і т. д.);
* кожен стовпець має унікальне ім'я;
* однакові рядки в таблиці відсутні;
* порядок проходження рядків і стовпців може бути довільним.

Базовыми понятиями реляционных СУБД являются:

Базовими поняттями в реляційних СУБД є :

* атрибут;
* відношення;
* [кортеж](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D1%82%D0%B5%D0%B6).

До числа найбільш поширених реляційних СУБД відносяться:MySQL, Microsoft SQL Server, Oracle, Sybase, и другие.

### ООСУБД

Об'єктно-орієнтована СУБД - реалізує об'єктно-орієнтований підхід. Ця система управління обробляє дані як абстрактні об'єкти, наділені властивостями, у вигляді неструктурованих даних, і використовують методи взаємодії з іншими об'єктами навколишнього світу. Об'єктно-орієнтована база даних - база даних, в якій дані оформлені у вигляді моделей об'єктів, що включають прикладні програми, які управляються зовнішніми подіями. Результатом поєднання можливостей (особливостей) баз даних і можливостей об'єктно-орієнтованих мов програмування є Об'єктно-орієнтовані системи управління базами даних (ООСУБД). ООСУБД дозволяє працювати з об'єктами баз даних так само, як з об'єктами в програмуванні на ООЯП. ООСУБД розширює мови програмування, прозоро вводячи довготривалі дані, управління паралелізмом, відновлення даних, асоційовані запити й інші можливості. Деякі об'єктно-орієнтовані бази даних розроблені для щільної взаємодії з такими об'єктно-орієнтованими мовами програмування як Python, Java, C #, Visual Basic. NET, C + +, Objective-C і Smalltalk; інші мають свої власні мови програмування. ООСУБД використовують точно таку ж модель, що й об'єктно-орієнтовані мови програмування. Об'єктно-орієнтовані бази даних звичайно рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, які мають складну структуру. У маніфесті ООБД (Atkinson et al., 1989) пропонуються обов'язкові характеристики, яким повинна відповідати будь-яка ООБД. Їх вибір заснований на 2 критеріях: система повинна бути об'єктно-орієнтованою і представляти собою БД.

Три класи характеристик:

* Обов’язкові;
* необов’язкові;
* відкриті  — дозволяють користувачу вибирати властивості.

Приклад об’єктно орієнтовної СУБД: [IBM Lotus Notes/Domino](http://ru.wikipedia.org/wiki/IBM_Lotus_Notes/Domino), Jasmine, ObjectStore, [Caché](http://ru.wikipedia.org/wiki/Cach%C3%A9_%28%D0%A1%D0%A3%D0%91%D0%94%29), [СООБЗ Cerebrum](http://www.shuklin.com/ai/ht/ru/cerebrum/), [db4objects](http://www.db4o.com).

### Об’єктно реляційні СУБД

Об'єктно-реляційна СУБД (ОРСУБД) - реляційна СУБД (РСУБД), що підтримує деякі технології, які реалізують об'єктно-орієнтований підхід. Різниця між об'єктно-реляційними і об'єктними СУБД: перші являють собою надбудову над реляційної схемою, другі ж спочатку об'єктно-орієнтовані. Головна особливість і відмінність об'єктно-реляційних, як і об'єктних, СУБД від реляційних полягає в тому, що О (Р) СУБД інтегровані з об'єктно-орієнтованою (OO) мовою програмування, внутрішньою або зовнішьою як C + +, Java. Характерні властивості OРСУБД - 1) комплексні дані, 2) успадкування типу, і 3) об'єктна поведінка.

Комплексні дані можуть бути реалізовані через постійно-збережені об'єкти (persistent objects). Створення комплексних даних у більшості існуючих ОРСУБД базується на попередньому визначенні схеми через визначений користувачем тип (UDT - user-defined type). Використовуються також вбудовані конструктори складових типів, наприклад масив (ARRAY). Ієрархія структурних комплексних даних пропонує додаткову властивість, успадкування типу. Тобто структурний тип може мати підтипи, які використовують всі його атрибути і містять додаткові атрибути, специфіковані в підтипі. Об'єктна поведінка закладається через опис програмних об'єктів. Такі об'єкти повинні бути збережені і перенесені для обробки в базі даних, тому вони називаються зазвичай як постійні (чи довготривалі) об'єкти. Усередині бази даних всі відносини з постійним програмним об'єктом є відносини з його об'єктним ідентифікатором (OID).

Об'єктно-реляційними СУБД є, наприклад, широко відомі Oracle Database, Microsoft SQL Server 2005, PostgreSQL.

### Вибір СУБД

При виборі моделі СУБД (ООБД, РБД або ОРБД) доцільно керуватися наступними критеріями:

1. наявність установлених стандартів;
2. природність рішення задачі зберігання;
3. відсутність залежності від конкретного виробника;

Нижче представлений порівняльний аналіз за попередніми критеріями у вигляді таблиці:

*Порівняння моделі СУБД*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ООБД | РБД | ОРБД |
| Наявність установлених стандартів | Ні | Так | Так |
| Природність рішення задачі зберігання | Так | Ні | Так |
| Відсутність залежності від конкретного виробника | Так | Так | Ні |

Як видно з таблиці, у кожної моделі є свій недолік. Розглянемо, що собою представляє і які наслідки тягне кожен з них:

Відсутність установлених стандартів є найбільш серйозною проблемою, тому як з року в рік можуть змінюватися специфікації моделі, не враховуючи, при цьому, питання зворотної сумісності. Ця проблема може привести до того, що реалізація програми буде регулярно втрачати свою актуальність і вимагати доопрацювання.

Природність зберігання даних означає той факт, що інформація (в даному випадку - об'єкти) зберігаються в базі даних в їх природному вигляді і не вимагають ніяких додаткових трансформацій. Реляційні БД не можуть надати такої можливості. У минулому розділі «Об'єктно-реляційне відображення» проблема зберігання об'єктних структур у реляційних БД розглянута детально.

Прихильність до конкретного виробника призводить до того, що розробка програми безпосередньо залежить від політики і переваг конкретного виробника, від якості реалізації його програмного продукту. При цьому перестати використовуватися його ПЗ і підключити інше означає те, що потрібна серйозна переробка проекту. Крім того, продукти класу ОРСУБД (такі як Oracle, MS SQL) є платними і досить потужними, громіздкими системами, що не зовсім підходить під потреби проекту.

Однак, таке рішення як PostgreSQL, хоч і є об'єктно реляційної базою, як сказано в розділі 1.5.2.4, але дозволяє розробити базу, не залежну від конкретного виробника, до того ж це рішення не є громіздкою системою і що дуже важливо - є безкоштовним рішенням .

З вищесказаного випливає, що доцільно вибрати або РСУБД або ОРСУБД, яка дозволить не залежати від конкретного виробника. Оскільки в додатку, що розробляється цілком для зберігання даних використовується об'єктно-реляційне відображення, і така реалізація ОРСУБД як PostgreSQL істотно знижує головні недоліки ОРСУБД, будучи при цьому безкоштовним рішенням, вибір ліг саме на неї.

# ВИСНОВОК ПО РОЗДІЛУ

У даному розділі була розглянута проблема забезпечення персистентності даних, зокрема - об'єктів Java. На даний момент існує безліч різних рішень, таких як:

* серіалізація об'єктів в потік (збереження у файл, передача по мережі та ін.);
* збереження об'єктів у реляційній БД за допомогою безпосереднього використання JDBC;
* використання об'єктно-реляційних відображень (ORM) в різних реалізаціях у відповідності зі специфікаціями, не прийнятими як стандарт;
* використання ООБД;
* використання технології J2EE EJB2;
* використання продуктів, що реалізують специфікацію EJB3.0/JPA.

У кожного методу є свої переваги і недоліки. Як підсумок, розглянемо таблицю, яка об'єднує воєдино особливості кожного з рішення:

Порівняння різних рішень по збереженню об’єктів

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Підтримка: | Механизм решения: | | | | | |
| Серіалізація | JDBC | ORM | ООБД | EJB2 | JPA |
| Java об’єкты | + | - | + | + | + | + |
| Принципи ООП | + | - | + | + | - | + |
| Атомарність транзакцій | - | + | + | + | + | + |
| Багатопоточність | - | + | + | + | + | + |
| Великй набір даних | - | + | + | + | + | + |
| Використання існуючої схеми сховища | - | + | + | - | + | + |
| Різні види сховищ | - | - | - | - | + | - |
| Запити | - | + | + | + | + | + |
| Чіткі стандарти та сумісність | + | - | - | - | + | + |
| Простота використання | + | + | + | + | - | + |

Легко помітити, що для використання у великих проектах ,які швидко розвиваються, підходять тільки такі рішення, як ORM, EJB2.0 і JPA.

Однак, головним недоліком не-JPA ORM продуктів є те, що проект, що розробляється стає прив'язаним до конкретного виробника. Перехід на іншу реалізацію ORM буде однозначно пов'язаний з масою труднощів, потягне за собою необхідність в серйозному перегляді написаного програмного коду. У свою чергу, використання продуктів, що реалізують EJB2.0 дозволить уникнути подібних труднощів. Але дана технологія є дуже громіздкою і складною, що часто робить її прийнятною для використання лише в досить великих і масштабних проектах з серйозним бюджетом.

Очевидно, що JPA є якраз тією технологією, яка, не вимагаючи величезних витрат на підготовку розробників, дозволить швидко і зручно забезпечити можливість зберігати об'єкти та гнучко оперувати ними. При цьому розробка не буде жорстко прив'язана до конкретного виробника, що дозволить у будь-який час перейти до використання найбільш підходящої для реалізації стандарту

# РОЗДІЛ 2

# РОЗРОБКА МОДУЛЯ

# Загальні відомості

У попередньому розділі були розглянуті загальні положення щодо мети створення проекту і завдань, які на нього покладають. У цьому розділі буде сформульовано опис структури даної бази даних, включаючи описи всіх сутностей.

У пункті «Вибір СУБД» був проведений аналіз різних моделей баз даних, і було обгрунтовано рішення про вибір об'єктно - реляційної моделі, як про найбільш вдале рішення для розробляємого проекту.

Проектування моделі можна почати з різних сторін:

1. Описати модель за допомогою об'єктів Java (як саме це зробити буде розглянуто нижче), потім на основі об'єктної моделі створити об'єктно-реляційну структуру.
2. Описати об'єктно-реляційну структуру, потім на її основі створити об'єктну модель. Даний підхід в Hibernate носить назву зворотного проектування.

Оскільки база даних, як було описано в розділі 1.1, є вхідним значенням, в даній роботі буде створюватися об'єктна модель на основі об'єктно-реляційної структури. На сьогоднішній день існують засоби, які дозволяють автоматизовано побудувати доменну модель на основі моделі, описаної в термінах бази даних.

Однак у навчальних цілях і для більш глибокого ознайомлення зі специфікацією JPA в проекті, що розробляється реалізація об'єктної моделі буде здійснюватися вручну.

# Опис структури бази даних

## Опис сутностей

##### **Навчальний заклад**

Дана сутність використовується для визначення приналежності приміщення, предмета, потоку і викладача до навчального закладу, який буває двох типів - кафедра і факультет. Додаткова інформація про навчальний заклад:

1. Назва.
2. Скорочена назва
3. Тип навчального закладу
4. Тип учебного заведения.
5. Ім’я, прізвище керівника навчального закладу.
6. Номер корпусу
7. Навчальний заклад в межах котрого існує даний навчальний заклад.

##### **Приміщення**

Сутність, яка дозволяє описати аудиторію, в якій проводяться заняття. Має таку інформацію:

1. Номер приміщення
2. Вид приміщення
3. Кількість місць в приміщенні
4. Наявність інвентарю в приміщенні
5. Навчальний заклад , котрому належить приміщення

##### **Предмет**

Дана сутність описує предмет в розкладі:

1. Назва.
2. Аббревіатура.
3. Потік, у котрого проводится даний предмет.

##### **Потік**

У рамках потоку може існувати кілька груп. Таким чином потік об'єднує кілька груп і забезпечує приналежність групи до певної кафедри:

1. Назва потоку
2. Рік створення потоку
3. Вид потоку1.
4. Вид потоку2.
5. Навчальний заклад , котрому належить приміщення

##### **Група**

Є інформаційною одиницею, що характеризує у кого буде проводитися навчальне заняття.

1. Назва.
2. Номер.
3. Потік.

##### **Викладач**

Сутність, характеризує того, хто веде навчальне заняття, і приписаний до навчального закладу:

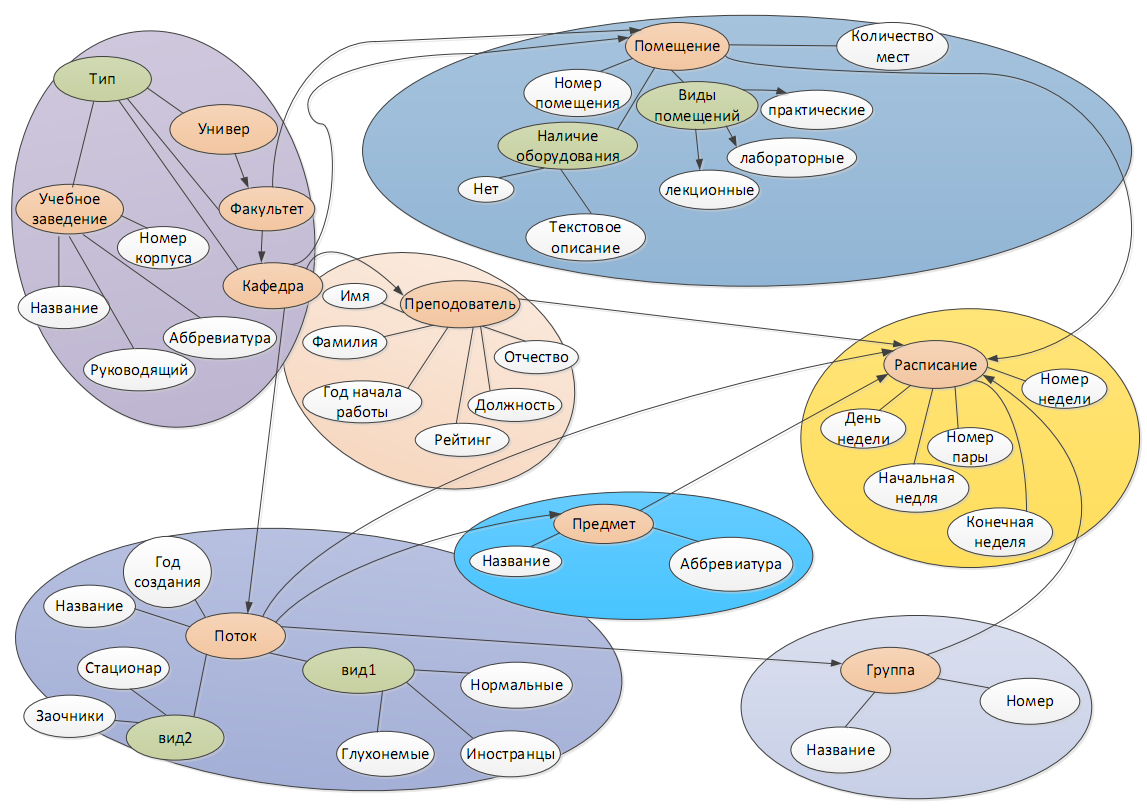
1. Ім’я.
2. Прізвище.
3. По-батькові.
4. Навчальний заклад.
5. Посада.
6. Рейтинг.
7. Рік початку роботи.

##### **Розклад**

Сутність, що дозволяє чітко ідентифікувати у кого проходить навчальне заняття, де проходить навчальне заняття, хто веде навчальне заняття, який предмет у навчального заняття:

1. Номер тижня.
2. День тижня.
3. Номер пари.
4. Група.
5. Приміщення.
6. Предмет.
7. Викладач.
8. Початковий тиждень.
9. Кінцевий тиждень.

## Схема використовуваних сутностей



*Рис 2.1.схема сутностей,використовуваних в базі даних розкладу.*

На рисунку 2.1 наведена загальна схема сутностей,що відображує візуальний опис сутностей, який було приведено у розділі 2.2.1. Також на рисунку чітко показані зв'язки між сутностями бази даних розкладу.

## Схема бази даних



*Рис 2.2. Схема бази даних.*

Зі схеми даної бази видно, що в ній реалізуються переважно відносини один до багатьох, тобто база має деревоподібну структуру.

Слід звернути увагу на таблицю Навчальний заклад. У ній міститься поле ID\_PTR, що є покажчиком на запис з цієї ж таблиці. Таким чином, досягається поєднання понять факультет і кафедра в рамках однієї таблиці. Дані поняття можна розмежувати в таблиці по полю тип, і визначити приналежність кафедри до факультету по полю ID\_PTR, що зберігають ідентифікатор на запис відповідного факультету.

Кожна таблиця в базі, як видно на малюнку, має власне поле ідентифікатора. Наявність такого поля в кожній таблиці диктується використанням специфікації JPA, про яку йтиме мова в розділі 2.3.4.

Варто звернути увагу, що такі поля як тип, в таблиці навчальний заклад, посаду в таблиці викладач, вид1 і вид2 в таблиці потік будуть заповнюватися з програми тільки через відповідні типи , для того що б уникнути заповнення таблиці некоректними даними.

# Розробка програмного забезпечення

## Терміни та визначення

*Профіль підключення* (або просто профіль) - набір параметрів, необхідних для підключення до визначеної бази даних. У число параметрів входить адреса і тип СУБД, дані для підключення, параметри об'єктно-реляційного відображення та ін. Одному типу підключення відповідає один профіль. [13]

*Фабрика менеджерів сутностей* - об'єкт з бібліотеки Hibernate, який призначений для управління менеджерами сутностей.

*Менеджер сутностей* - об'єкт з бібліотеки Hibernate, призначений для управління сутностями, являє собою сеанс роботи з базою даних. У рамках одного менеджера можна відкривати і закривати транзакції, зберігати, отримувати та видаляти об'єкти.

*Сутність* - простий java об'єкт (POJO), що представляє собою рядок з таблиці в базі даних.

Сервер програм - сервер, призначений для виконання прикладних процесів. Сервер програм:

* взаємодіє з клієнтами, одержуючи завдання;
* взаємодіє з базами даних, вибираючи дані, необхідні для обробки;
* надає різні програмні інтерфейси (створюючи, таким чином, особливе середовище виконання) для прикладних процесів.

Веб-контейнером будемо називати сервер програм, спеціально спроектований для обслуговування клієнтів Web-мережі (тобто, в першу чергу, за протоколом HTTP) і який надає програмні інтерфейси, необхідні для виконання в рамках контейнера веб-програм.

## Загальні відомості

Розроблюване програмне забезпечення призначене для роботи з сутностями, через менеджери сутностей. Метою даного програмного продукту є надання можливості для рівня логіки вибору даних з бази і їх занурення в базу даних.

Під роботою з сутностями через менеджери мається на увазі:

1. Автоматичне занурення списку даних, прийнятих від рівня логіки, в базу даних.
2. Вибірка даних з бази і надання їх рівню логіки.
3. Автоматичне створення менеджерів сутностей.

## Особливості модуля

Важливою особливістю розроблюваного модуля є те, що він призначений для використання у веб-програмі. А це означає, що модуль повинен:

1. Інтегруватися у веб контейнер: не дивлячись на те, що модуль розробляється як частина веб-програми, його функції та поведінка має бути максимально наближена до підключеного модуля веб-контейнера. Це виражатиметься в тому, що модуль повинен розширювати програмний інтерфейс сервера і працювати не в циклі життя клієнтського запиту, а в контексті всієї веб-програми.
2. Легко помітити, що з попередньої вимоги випливає нова: модуль повинен працювати в багато поточному режимі.

Більш докладно питання реалізації цих вимог будуть розглянуті нижче.

## Розробка сутностей

### Опис сутностей в програмі

Специфікація JPA 2.0 чітко описує яким чином можна писати класи сутностей. Слід зауважити що екземпляри даних класів будуть представляти собою рядок таблиці з бази даних.

Для того, що б встановити відповідність між таблицею в базі і класом в програмі існує два підходи:

1. Використовувати xml «мапінг» файли.
2. Використовувати анотації.

Оскільки від вибору підходу залежить тільки зручність при проектуванні об'єктної моделі, у даному проекті був обраний другий підхід. Він дозволяє відразу при написанні вихідного коду класу задавати відповідність між полями класу і таблицею в базі прямо у файлі класу. Далі слід навести деякі правила з специфікації JPA, для того що б показати чим класи сутностей відрізняються від звичайних класів у java:

1. На початку класу сутності, перед його модифікатором доступу повинна бути анотація @Entity.
2. Кожен клас сутність повинен мати поле унікального ідентифікатора, з анотацією @Id.
3. Клас сутність повинен мати конструктор без аргументів з модифікатором доступу public.
4. Клас сутність повинен бути top level класом, і не бути типу enum.
5. Клас сутність не повинен мати модифікатор final.
6. Також не повинні бути final поля і методи класу.

Специфікація JPA визначає доступ до стану об'єкта суті, через

1. Через поля класа
2. Через методи доступу до полей класу.

При виборі першого підходу анотації пишуться над полями класу. При виборі другого підходу, анотації вказуються тільки над методами доступу до полів класу. Важливо не переплутати і вибрати або перший, або другий підхід. В іншому випадку, якщо будуть використовуватися частково обидва підходи, специфікація правильність роботи не гарантує.

Згідно зі специфікацією, якщо ім'я класу відрізняється від імені таблиці, то потрібно скористатися анотацією @Table(name = "").[14]

Теж саме стосується і полів класу. Якщо поле класу має різне назву з відповідним полем у таблиці, то слід використовувати анотації @Column(name = "").

Слід приділити увагу використанню типів в класах сутності. Дозволені специфікацією JPA типи - це всі примітивні java типи, обгортки над примітивними типами, тип String, типи описані користувачем, типи що реалізують інтерфейс Serializable, типи сутностей, колекцій сутностей і обрізаних класів.

### Опис зв’язків між сутностями

При розробці сутностей постає питання про опис зв'язків між сутностями. Оскільки таблиці в базі даних мають відносини з іншими таблицями, що було вже наведено в розділі 2.2.3, то і в об'єктному поданні сутності повинні мати такі ж відносини.

Специфікація JPA дозволяє визначити відносини один до одного, один до багатьох / багато до одного і багато до багатьох. У заданої бази даних використовуються тільки відносини багато до одного і один до багатьох. В такому випадку використовуються анотації @ ManyToOne і @ OneToMany. Остання анотація використовується у випадку двобічного зв'язку:

@ManyToOne – використовує клас сутність, відповідна таблиця якого містить в собі зовнішній ключ на іншу таблицю. У другої таблиці, згідно специфікації JPA теж є клас сутність. Посилання на екземпляр цього класу суті,має зберігатися в полях першого класу сутності. При чому анотація @ ManyToOne пишеться саме над цим полем, або методом доступу, в залежності від обраного способу доступу до об'єктів, про який говорилося вище.

Слід зазначити, що наведені правила із специфікації є лише короткими витягами з неї, а саме ті, які застосовувалися в даній роботі. [13]

# ВИСНОВОК ПО РОЗДІЛУ

У другому розділі була проведена розробка проекту з урахуванням даних та висновків, отриманих у попередньому розділі. Другий розділ складався з двох етапів:

1. опис бази даних;
2. проектування програми.

Питання опису моделі можна почати з двох сторін:

1. реляційної (проектування схеми бази даних);
2. об’єктної (проектування об’єктів-сутностей).

Можливості бібліотеки Hibernate також дозволяють вибрати розробнику той варіант рішення, який йому більше підходить. У даній роботі в навчальних цілях використовувалися опис моделі із застосуванням двох підходів.

Відповідно до завдання на роботу модуль призначений для забезпечення персистентності даних. Було наведено опис бази даних і розробка сутностей - об'єктного уявлення, згідно специфікації JPA 2.0. В описі бази даних були описані всі використовувані суті, і згодом всі сутності з взаємозв'язками між собою були представлені графічно. На основі сутностей бази даних, була представлена схема бази даних. При розробці об'єктної моделі були наведені правила згідно з якими можна описувати суті, і взаємозв'язки між ними.

Проектування програми полягала у формуванні мети створення програми, детального опису кола завдань, які можна буде вирішувати з допомогою розроблюваного програмного модуля.

# РОЗДІЛ 3

# РЕАЛІЗАЦІЯ МОДУЛЯ

## Загальні відомості

Задачу установки, налаштування і введення модуля в експлуатацію можна розділити на декілька логічних етапів:

1. Налаштування бази даних: установка і настройка СУБД, створення схеми БД.
2. Реалізація програмного модуля
3. Інтеграція програмного модуля в прикладну або веб програму.
4. Створення та налаштування об'єктів Java - сутностей системи.

## Налаштування бази даних

Список сутностей системи збігається зі списком таблиць в структурі бази даних. Повний список (включаючи і логічні, і фізичні назви) представлений нижче:

1. Навчальний заклад (institution).
2. Приміщення (Class).
3. Викладач (Lecturer).
4. Предмет (Subject).
5. Потік (Flow).
6. Група (Group).
7. Розклад (Schedule).

Окремо сутності були розглянуті у розділі 2.2.1 та графічно наведені на малюнку 2.1. Схема ж самої бази даних була наведена в розділі 2.2.2.

### Идентифікатор записів

У кожної таблиці є обов'язковий стовпець id - це поле ідентифікатора, яке служить для однозначного визначення запису в межах однієї таблиці.  
Визначимо список вимог, яким має задовольняти дане поле:

1. Повинно бути унікальним в рамках існування таблиці: для того, щоб уникнути колізій, не один ідентифікатор не повинен повторюватися. Навіть якщо на момент створення запису якийсь ідентифікатор не існує, але існував раніше, він повинен бути пропущений.
2. Не може бути порожнім: ідентифікатор повинен бути конкретним виразом, ідентифікатор не може бути NULL.
3. Повинна бути можливість генерувати значення поля автоматично: розробник не повинен явно піклуватися про те, щоб задовольнити попередні 2 вимоги.
4. Повинно бути первинним ключем: для швидкого пошуку.

В якості типу поля ідентифікатора виберемо числове: це самий логічний і зрозумілий метод, якщо не потрібно вирішувати якісь більш-менш специфічні завдання.

Опишемо в термінах SQL поле идентифікатора:

id INT NOT NULL auto\_increment PRIMARY KEY.

### Установка та налаштування PostgreSQL

Є два наступних типу дистрибутивів сервера PostgreSQL для Windows: [18]

1. Бінарний дистрибутив, до складу якого входить програма установки; вона встановлює все, що потрібно, так що можна відразу ж запускати сервер.
2. Дистрибутив вихідного коду, в якому міститься весь код і файли підтримки для створення виконуваних файлів.

При виконанні даної роботи для встановлення був узятий бінарний дистрибутив.На комп'ютері, на якому передбачається встановити СУБД, повинно бути наступне:

1. 32-розрядна операційна система Windows, така як 9x, Me, NT, 2000 або XP. Під управлінням сімейства NT (NT, Windows 2000 і XP) сервер PostgreSQL можна запускати як службу.
2. Підтримка протоколу TCP/IP
3. Досить місця на жорсткому диску для розпакування, установки і створення баз даних відповідно до вимог проекту.

При установці сервера програма інсталяції пропонує зробити мінімальні, але необхідні настройки сервера бази даних. А саме: [17]

1. Вибрати обліковий запис, з якого буде запускатися сервер бази даних. У разі якщо обліковий запис не створена заздалегідь програма установки її створює.
2. Вибрати порт на який сервер баз даних буде приймати з'єднання.
3. Вибір суперюзера - ім'я та пароль облікового запису, від імені якого буде вестися робота з базою даних. Дане поняття специфічне для PostgreSQL.
4. Вибір кодування, в якій база даних буде зберігати дані

Согласно вышеуказанным требованиям, были выбраны такие настройки :

Порт 5432, ім'я суперюзера - postgres, пароль суперюзера - 1234, кодування бази даних - WIN1251, або її синонім CP1251. Дані налаштування також вказуються в конфігураційних файлах Hibernate, про що буде сказано нижче.

## Реалізація програми

### Шаблон Singleton

Паттерн Singleton гарантує, що у класу є тільки один екземпляр, та надає до нього глобальну точку доступу. На даний момент існують декілька варіантів реалізації зі своїми недоліками і перевагами [19]. Розглянемо перший і найпростіший спосіб:

public final class Singleton {

private static Singleton \_instance = null;

private Singleton() {}

public static Singleton getInstance() {

if (\_instance == null)

\_instance = new Singleton();

return \_instance;

}

}

У цього рішення є єдиний недолік - воно не працює в багатопотоковому середовищі і тому не підходить в більшості випадків. Рішення підходить винятково для однопоточних програм.

Розглянемо другий варіант:

public final class Singleton {

private static Singleton \_instance = new Singleton();

private Singleton() {}

public static Singleton getInstance() {

return \_instance;

}

}

Проблема багатопоточності при даному підході вирішена, але губляться дві важливі речі:

1. Лінива ініціалізація (Об'єкт instance буде створений classloader-му під час ініціалізації класу)
2. Відсутня можливість обробки виняткових ситуацій (exceptions) під час виклику конструктора.

Рішення підходить для багатопотокових додатків, за умови відсутності небезпеки виникнення виняткових ситуацій в конструкторі та відсутність необхідності ледачою ініціалізації.

Розглянемо третій варіант (рішення Біла Пью):

public class Singleton {

private Singleton() {}

private static class SingletonHolder {

private static final Singleton INSTANCE = new Singleton();

}

public static Singleton getInstance() {

return SingletonHolder.INSTANCE;

}

}

У даному випадку повністю вирішена проблема ледачою ініціалізації - об'єкт ініціалізується під час першого виклику методу getInstance (). Але залишилася проблема з обробкою виняткових ситуацій в конструкторі. Так що, якщо конструктор класу не викликає побоювань створення виняткових ситуацій, то дані метод є вдалим рішенням.

Розглянемо четвертий варіант:

public final class Singleton {

private static Singleton \_instance = null;

private Singleton() {}

public static synchronized Singleton getInstance() {

if (\_instance == null)

\_instance = new Singleton();

return \_instance;

}

}

У цього варіанта є тільки один недолік. Синхронізація корисна лише один раз, при першому зверненні до getInstance (), після цього кожен раз, при зверненні до цього методу, синхронізація просто забирає час. З іншого боку, якщо виклик getInstance () не відбувається досить часто, то цей метод має перевагу перед іншими: він простий, зрозумілий, ліниво ініціалізується, дає можливість обробляти виняткові ситуації в конструкторі. До того ж, синхронізація в Java працює досить швидко. Тепер розглянемо варіант синхронізованого рішення, в якому буде зроблена спроба вирішити проблему, що виникла в попередньому варіанті. Найбільш поширений спосіб - «Double-Checked Locking». [19] Отже, п'ятий варіант, в своєму оригінальному варіанті:

public final class Singleton {

private static Singleton \_instance = null;

private Singleton() {}

public static Singleton getInstance() {

if (\_instance == null) {

synchronized (Signleton.class) {

if (\_instance == null) {

\_instance = new Singleton();

}

}

return \_instance;

}

}

Однак, зважаючи на цілий ряд складнощів, які виникають при використанні цього методу, в JAVA 5 вирішили використовувати модифікатор volatile [18]. На даний момент рішення виглядає так:

public final class Singleton {

private static volatile Singleton \_instance = null;

private Singleton() {}

public static Singleton getInstance() {

if (\_instance == null) {

synchronized (Signleton.class) {

if (\_instance == null) {

\_instance = new Singleton();

}

}

return \_instance;

}

}

Розглянемо, який метод є найбільш підходящим для розробляється схеми:

1. Екземпляр класу фабрики менеджерів сутностей, повинен бути тільки один.
2. Створення даного екземпляру відбувається в багатопотоковому середовищі, тобто при безлічі потоків повинен бути створений тільки один екземпляр даного класу.

Вищевказані вимоги, вказують на те, що краще всього використовувати останній варіант шаблону Singleton.

### Интеграція модуля

Для того, щоб ввести модуль в експлуатацію, потрібно підготувати сервер і іншу частину системи для роботи з ним. Підготовка включає в себе наступні етапи :

1. Завантаження бібліотеки Hibernate і всі її залежності; установка бібліотеки (для веб-програми установка полягає в розміщенні набору бібліотек у спеціальну директорію проекту / lib) [21].
2. Завантаження JDBC драйверів для тих СУБД, з якими передбачається робота бібліотеки Hibernate. Аналогічно до попереднього пункту, драйвера потрібно помістити в / lib. У даній роботі використовується драйвер для роботи з PostgreSQL.
3. Приміщення модуля і всіх класів-сутностей, включаючи їх файли відображення, в CLASSPATH проекту
4. Розміщення файлу конфігурації профілю в папці META-INF проекту.

Після успішного виконання вище описаних кроків модуль готовий до використання.

# ВИСНОВОК ПО РОЗДІЛУ

У третьому розділі була проведена реалізація модуля. Даний розділ складався з двох етапів, а саме:

1. Установка та налаштування використовуємої бази даних.
2. Реалізації програмного модуля.

У першому розділі був проведений аналіз існуючих реалізацій баз даних. На основі цього аналізу та вимог до модуля, що реалізовувався була вибрана об’єктно реляційна база даних PostgreSQL.

В третьому розділі детально описано як установити даний програмний продукт, починаючи від скачування необхідних даних до запуску бази в експлуатацію. В даному проекті був обраний бінарний дистрибутив, який установлювався на 32 розрядну операційну систему Windows XP. Також були зазначені своєрідні налаштування бази для розробляє мого проекту.

При реалізації програмного модуля найбільша увага приділялася тому щоб даний модуль міг працювати в багато потоковому режимі. В розділі 3.3.1 детально описано як багато потоковий режим впливає на розробку даного модуля.

# ВИСНОВОК ПО РОБОТІ

У даній бакалаврській роботі було розглянуто питання розробки модуля персистентності для автоматизованої системи складання розкладу. Проектування і реалізація модуля почалася з аналізу предметної області та огляду існуючих у сучасному світі рішень.

Одним з найбільш популярних в наші дні шаблонів проектування програми є шаблон MVC - модель-відображення-контроллер, - який передбачає розбиття програмного модуля на 3 відповідні частини. Кожний логічний блок виконує свою роль і, в ідеальному випадку, є незалежним від іншого блоку. Взаємодія здійснюється через спеціально розроблені програмні інтерфейси, а самі частини модуля є легкозамінними.

Розробка модуля персистентності відбувалася з урахуванням вищеописаного шаблону, в якому розроблений модуль виконує роль моделі в рамках даного проекту для автоматизованого складання розкладу. В якості технології для реалізації поставленого завдання була обрана мова програмування Java з огляду на те, що Java є досить зручним і потужним інструментом для вирішення подібного кола завдань. Ця мова програмування користується великою популярністю для розробки он-лайн систем та інтернет проектів, а значить, не варто упускати з уваги той факт, що навички, набуті в ході написання даної роботи, будуть корисними в реальному житті і професійному зростанні.

У роботі були розглянуті різні підходи для вирішення проблеми забезпечення персистентності даних об'єктно-орієнтованих мов програмування. У число аналізованих варіантів увійшли використання об'єктно-орієнтованих СУБД, об'єктно-реляційних СУБД, серіалізациі, Enterprise Java Beans, а також бібліотек об'єктно-реляційного відображення (ORM).

Проте, у цій роботі не була використана технологія EJB, яка була спеціально розроблена для реалізації шару моделі в проектах, написаних на Java. Рішення про використання ORM було прийнято тому, що EJB є занадто важкою і надлишкової технологією для реалізації розроблюваного проекту.

Після були розглянуті різні бібліотеки, що забезпечує об'єктно-реляційне відображення: openJPA, Cayenne, Hibernate. Унаслідок того, що набір можливостей та принципів роботи почав регламентуватися загальним стандартом - JPA, перераховані вище інструменти багато в чому схожі. Головними факторами при виборі стали особливості роботи (на малих і великих вибірках), а також активність розвитку і підтримки проекту. Тому вибір ліг на Hibernate, який, хоча і має ряд своїх серйозних недоліків, є вдалим рішенням для розробленого модуля.

У третьому розділі було наведено опис процесу реалізації проекту: установка СУБД PostgreSQL, реалізація програмного модуля відповідно до вимог, сформульованими у другому розділі. Особливу увагу було приділено питанню забезпечення багатопоточності, без рішення якого модуль був би неприпустимим для експлуатації в реальному інтернет проекті.

Підсумками даної бакалаврської роботи є шар персистентності, готовий для введення в експлуатацію і складається з програмного модуля (який легко адаптувати для вирішення питань збереження об'єктів з використанням Hibernate і в інших проектах). Крім того, в ході написання даної роботи були придбані важливі навички і знання в предметній області, які будуть вкрай корисні у професійній діяльності та подальшому навчанні.

Список ВИКОРисТАНОЇ ЛІТЕРАТУРИ

1. Model-View-Controller – Вікіпедія: http://ru.wikipedia.org/wiki/MVC

1. [Модель, вид, контролер (MVC)](http://webphp.ru/model-vid-kontroller-mvc/" \o "Permanent Link to Модель, вид, контроллер (MVC)): http://webphp.ru/model-vid-kontroller-mvc/
2. Узагальнений Model-View-Controller: http://www.rsdn.ru/?article/patterns/generic-mvc.xml
3. Ruby on Rails — Вікіпедія: http://ru.wikipedia.org/wiki/Ruby\_on\_Rails
4. MVC Framework: великий вступ для початківців: <http://habrahabr.ru/blogs/net/49718/>
5. Сервлет — Вікіпедія: <http://ru.wikipedia.org/wiki/Сервлет>
6. Вступ в технологію Java Servlet: http://www.ibm.com/developerworks/ru/edu/j-intserv/section3.html
7. Введение в Java Server Faces http://www.ibm.com/developerworks/ru/edu/j-jsf1/index.html
8. Java SE Desktop Technologies - Java Beans: http://java.sun.com/javase/technologies/desktop/javabeans/index.jsp
9. ORM – Вікіпедія: http://ru.wikipedia.org/wiki/ORM
10. PostgreSQL Performance blog: http://www.ebrueggeman.com/blog/?tag=postgresql-performance
11. Персистентність Java-об’ектів: Часть 2: http://citcity.ru/18902/
12. Java Persistence API: http://java.sun.com/javaee/technologies/persistence.jsp
13. TopLink – Wikipedia: http://en.wikipedia.org/wiki/TopLink
14. Руководство по PostgreSQL: <http://postgresql.ru.net/manual/tutorial.html>
15. PostgreSQL: Руководство адміністратора: http://postgresql.ru.net/manual/admin.html
16. PostgreSQL: установка, налаштування , опис: http://postgresql.ru.net/docs/win\_inst.html
17. Реалізація Singleton в JAVA: http://habrahabr.ru/blogs/complete\_code/27108/
18. The "Double-Checked Locking is Broken" Declaration: http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html
19. Теорія и практика Java: : http://www.ibm.com/developerworks/ru/library/j-jtp03304/index.html
20. Сайт проекту Hibernate: https://www.hibernate.org/