МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
Тернопільський національний технічний університет
імені Івана Пулюя

Кафедра
комп'ютерних наук

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни
"Крос-платформне програмування"
для студентів денної та заочної форм навчання
напряму підготовки
6.050101 "Комп'ютерні науки"

Тернопіль, 2012

Методичні вказівки розроблені у відповідності з навчальним планом

напряму підготовки 6.050101 “Комп’ютерні науки ”

Укладачі: Козак Р.О., Михайлович Т.В.

Рецензент: *доцент ТНЕУ Касянчук М.М.*

Відповідальний за випуск: зав.каф. КН, проф., д.т.н. М.В.Приймак

Затверджено на засіданні кафедри комп’ютерних наук
Протокол № 2 від “ 4 ” 09 2012 р.

Схвалено та рекомендовано до друку методичною комісією факультету комп’ютерно-інформаційних систем і програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя.
Протокол № 2 від “27” 09 2012 р.

Вказівки складені з урахуванням матеріалів літературних джерел, названих у списку.

ЗМІСТ

ВСТУП

Натепер, в епоху активного використання комп'ютерів різноманітних архітектур, апаратного та програмного забезпечення, особливо гостро стоїть питання крос-платформного програмування, адже використання комп'ютерних технологій ставить перед розробниками програмних систем все нові і нові вимоги до забезпечення конфіденційності, цілісності, автентичності даних.

Дані методичні вказівки призначені для виконання лабораторних робіт у першому семестрі вивчення дисципліни "Крос-платформне програмування" для студентів спеціальності 6.050101 "Інформаційні управляючі системи та технології" денної та заочної форми навчання.

Виконання лабораторних робіт повинно забезпечити закріплення теоретичних знань, отриманих при вивченні лекційного матеріалу та практичної частини дисципліни.

В методичних вказівках розглянуті основні теоретичні питання, пов'язані з виконанням лабораторних робіт.

ЛАБОРАТОРНА РОБОТА 1: РОЗПОДІЛЕНА АРХІТЕКТУРА КОМПОНЕНТНИХ СИСТЕМ.

Мета роботи: Отримати практичні навички архітектурного проектування програмного забезпечення.

1.1 Теоретичні відомості

Процес визначення архітектури, компонентів, інтерфейсів та інших характеристик системи або її компонентів називається *проектуванням*. Результат процесу проектування повинен описувати архітектуру програмного забезпечення, тобто представляти декомпозицію програмної системи у вигляді організованої структури компонентів і інтерфейсів між компонентами.

Проектування переважно розглядається як двокроковий процес:

1) *архітектурне проектування* - декомпозиція структури (*статичного*) і організації (*динамічного*) компонентів;
2) *деталізація архітектури* - описує специфічну поведінку та характеристики окремих компонентів.

Виходом цього процесу є набір *моделей* і *артефактів*, що містять результати рішень, прийнятих за способами реалізації вимог в програмному коді.

У строгому значенні архітектура програмного забезпечення (*software architecture*) – опис підсистем і компонентів програмної системи, а також зв'язків між ними. Архітектура намагається визначити внутрішню структуру одержуваної системи, задаючи спосіб, яким система організована або конструюється.

Будь-яка система може розглядатися з різних точок зору – наприклад,

поведінкової (динамічної), *структурної* (статичної), *логічної* (задоволення функціональним вимогам), *фізичної* (розподіленість), *реалізації* (як деталі архітектури представляються в коді) і т.п. У результаті, ми отримуємо різні *архітектурні представлення*.

Архітектурний стиль, за своєю суттю, є шаблоном проектування макро-архітектури на рівні модулів, *"крупноблочного" погляду*. Наприклад, архітектура розподіленої *сервісно-орієнтованої* системи може будуватися в стилі обміну повідомленнями через відповідні черги повідомлень, може проектуватися на основі ідеї взаємодії між компонентами і застосунками через загальну об'єктну шину, а може використовувати концепцію *брокера* як єдиного вузла пересилання запитів.

У той час, як архітектурний стиль визначає *макро-архітектуру* системи, шаблони проектування задають *мікро-архітектуру*, тобто визначають аспекти деталей архітектури.

Найчастіше говорять про наступні групи шаблонів проектування:

- шаблони створення (*Creational patterns*): *builder*, *factory*, *prototype*, *singleton*;
- структурні шаблони (*Structural patterns*): *adapter*, *bridge*, *composite*, *decorator*, *facade*, *flyweight*, *proxy*;
- шаблони поведінки (*Behavioral patterns*): *command*, *interpreter*, *iterator*, *mediator*, *memento*, *observer*, *state*, *strategy*, *template*, *visitor*.

Певні нотації використовуються на стадії концептуального проектування, ряд нотацій орієнтований на створення детального дизайну, багато з них можуть використовуватися на обох стадіях.

Розглянемо *нотації*, виходячи з опису *структурного* (статичного) або *поведінкового* (динамічного) представлень. Наступні нотації статичного представлення, в основному, є графічними, описуючи і представляючи структурні аспекти програмного дизайну. Найчастіше вони стосуються основних компонентів і зв'язків між ними:

1) *Мови опису архітектури* (*Architecture description language*, ADL): текстові мови, часто – формальні, що використовуються для опису програмної архітектури в термінах компонентів і коннекторів (спеціалізованих *компонентів*, що реалізують не функціональність, але забезпечують взаємозв'язок функціональних компонентів між собою і з "зовнішнім світом").
2) *Діаграми класів і об'єктів* (*Class and object diagrams*): використовуються для представлення набору класів і статичних зв'язків між ними (наприклад, *спадкування*).
3) *Діаграми компонентів або компонентні діаграми* (*Component diagrams*): у певній мірі аналогічні діаграмам класів, однак, в силу специфіки концепції або поняття компонента, зазвичай, надаються в іншій візуальній формі.
4) *Картки функціональної відповідальності та зв'язків класу* (*Class responsibility collaborator card*, CRC): використовуються для позначення *імені* класу, його *відповідальності* (тобто, що він повинен робити) і інших *сутностей* (класів, компонентів, акторів/ролей і т.п.), з якими він пов'язаний; часто їх називають картками *"клас-обов'язок-кооперація"*.
5) *Діаграми розгортання* (*Deployment diagrams*): використовується для представлення (фізичних) вузлів, зв'язків між ними і моделювання інших фізичних аспектів системи.
6) *Діаграми сутність-зв'язок* (*Entity-relationship diagram*, ERD або ER): використовується для представлення концептуальної моделі даних, які зберігаються в процесі роботи інформаційної системи.
7) *Мови опису/визначення інтерфейсу* (*Interface Description Languages*, IDL): мови, подібні до мов програмування, що не включають можливостей опису логіки системи і призначені для визначення *інтерфейсів* програмних компонентів (*імен* та *типів*

експортованих або публікованих операцій).

8) *Структурні діаграми Джексона* (*Jackson structure diagrams*): використовуються для опису структур даних у термінах *послідовності*, *вибору* і *ітерацій* (повторень).
9) *Структурні схеми* (*Structure charts*): описують структуру викликів у програмах (який модуль викликає, ким і як викликається).

Наступні нотації і мови використовуються для опису динамічної поведінки програмних систем і їх компонентів:

1) *Діаграми діяльності або операцій* (*Activity diagrams*): використовуються для опису потоків робіт і управління.
2) *Діаграми співробітництва* (*Collaboration diagrams*): показують динамічну взаємодію, що відбувається в групі об'єктів, і приділяють особливу увагу *об'єктам*, *зв'язкам* між ними і *повідомленням*, якими обмінюються об'єкти за допомогою цих зв'язків.
3) *Діаграми потоків даних* (*Data flow diagrams*, DFD): описують *потоки* даних всередині набору процесів (не в термінах процесів операційного середовища, але в розумінні обміну інформацією в бізнес-контексті).
4) *Таблиці і діаграми прийняття рішень* (*Decision tables and diagrams*): використовуються для представлення складних комбінацій *умов* і *дій* (операцій).
5) *Блок-схеми і структуровані блок-схеми* (*Flowcharts and structured flowcharts*): застосовуються для представлення *потоків управління* (контролю) та пов'язаних операцій.
6) *Діаграми послідовності* (*Sequence diagrams*): використовуються для опису *взаємодій* всередині групи об'єктів з акцентом на тимчасовій послідовності *повідомлень/викликів*.

7) *Діаграми переходу і карти станів* (*State transition and statechart diagrams*): застосовуються для опису потоків управління переходами між *станами*.
8) *Формальні мови специфікації* (*Formal specification languages*): текстові мови, використовують основні поняття з математики (наприклад, *множини*) для *суворого* і *абстрактного* визначення інтерфейсів і поведінки програмних компонентів, часто в термінах *перед-* і *пост-умов*.
9) *Псевдокод і програмні мови проектування* (*Pseudocode and program design languages*, PDL): мови, що використовуються для опису поведінки *процедур* і *методів*, в основному на стадії *детального проектування*; подібні структурним мовам програмування.

Стратегії включають *функціонально-орієнтоване* або *структурне* проектування*, об'єктно-орієнтоване* проектування, проектування на основі структур даних, *компонентне проектування*.

Функціонально-орієнтоване проектування – це один із класичних методів проектування, в якому декомпозиція сфокусована на ідентифікації основних програмних функцій і потім детальній розробці та уточненні цих функцій "зверху-вниз".

Об'єктно-орієнтоване проектування являє собою безліч методів проектування, які базуються на концепції об'єктів. Головну роль відіграють *поліморфізм* і *інкапсуляція*.

При проектуванні на основі структур даних фокус сконцентрований в більшій мірі на структурах даних, якими управляє система, ніж на функціях системи. Інженери з програмного забезпечення часто спочатку описують структури даних входів (*inputs*) і виходів (*outputs*), а потім розробляють структуру управління цими даними (або, наприклад, їх *трансформації*).

Компонентне проектування засновано на створенні програмних

компонентів. Програмні компоненти є *незалежними* одиницями, які мають однозначно визначені *інтерфейси* і *залежності* і можуть збиратися і розгортатися незалежно один від одного. Даний підхід покликаний вирішити завдання *використання*, *розробки* та *інтеграції* компонентів з метою підвищення їх *повторного використання*.

Компонентно-орієнтоване проектування є однією з концепцій, що найбільш динамічно розвиваються, і може розглядатися як передвісник і основа *сервісно-орієнтованого підходу* (*Service-Oriented Architecture*, SOA) в проектуванні. *Нотація* UML 2.0 вже дозволяє вирішувати низку питань, пов'язаних з візуальним поданням відповідних архітектурних рішень, де сервіси можуть розглядатися як функціональність одиночних компонентів і груп компонентів, об'єднаних у більш "великі" блоки, що забезпечують надання відповідної сервісної функціональності.

1.2 Порядок виконання

1. Визначити стратегію проектування для реалізації програмного забезпечення.
2. Розробити моделі для статичного представлення архітектури програмного забезпечення.
3. Розробити моделі для динамічного представлення архітектури програмного забезпечення.
4. Задокументувати розроблені моделі, використовуючи Visual Paradigm for UML.
5. Оформити звіт, який повинен містити обґрунтування вибору архітектури програмного забезпечення, розроблені за допомогою Visual Paradigm for UML моделі структурного та динамічного представлення компонентів системи.

1.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

1.4 Контрольні запитання

1. Що таке архітектура програмного забезпечення?
2. Що визначає макро- та мікро-архітектуру програмного забезпечення?
3. Які нотації використовуються для статичного представлення архітектури програмного забезпечення?
4. Які нотації використовуються для динамічного представлення архітектури програмного забезпечення?
5. Обґрунтуйте вибір діаграм, розроблених для статичного та динамічного представлення архітектури програмного забезпечення, що має бути розроблене.
6. Які стратегії проектування можна виділити?
7. Обґрунтуйте обрану стратегію проектування для реалізації програмного забезпечення.

1.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ

ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з роширенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс, 2007. – 768 с.
6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.
7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.
9. Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 2: КОМПОНЕНТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ.

Мета роботи: Отримати практичні навички компонентно-орієнтованого проектування програмного забезпечення. Для цього, створити бібліотеку DLL.

2.1 Теоретичні відомості

Сучасні кросплатформні системи створюються з використанням *компонентно-орієнтованого підходу*.

Компонентне програмування є подальшим розвитком об'єктно-орієнтованого програмування (ООП) і концепції *«повторного використання»* (*reuse*).

Компонентне програмування покликане забезпечити простішу і швидшу процедуру інсталяції прикладного програмного забезпечення, а також збільшити відсоток повторного використання коду, тобто підсилити основні переваги ООП.

Перш за все, сформулюємо головне для даного підходу визначення компонента.

Під *компонентом* далі матимемо на увазі незалежний модуль програмного коду, призначений для повторного використання і *розгортання*.

Такий компонент є структурною одиницею програмної системи, що має чітко визначений *інтерфейс*.

Компонент може бути незалежно поставлений або не поставлений, доданий до складу деякої системи або видалений з неї, у тому числі, може

включатися до складу систем інших постачальників.

Таким чином, *компонент* — це виділена структурна одиниця розгортання з чітко визначеним інтерфейсом**.**

Всі залежності компонента від оточення мають бути описані в рамках цього інтерфейсу. Один компонент може також мати декілька інтерфейсів, граючи декілька різних ролей в системі. При описі інтерфейсу компонента важлива не лише сигнатура операцій, які можна виконувати з його допомогою. Стає важливим і те, які інші компоненти він може використовувати при роботі, а також яким обмеженням повинні задовольняти вхідні дані операцій і які властивості виконуються для результатів їх роботи. Ці обмеження є так званим *інтерфейсним контрактом* або *програмним контрактом компонента***.**

Властивості компонентів

1. Використання компонента (виклик його методів) можливе лише через його інтерфейси відповідно до контракту.

Інтерфейс компонента включає набір операцій, які можна викликати в будь-якого компонента, який реалізує даний інтерфейс, і набір операцій, які цей компонент може викликати у відповідь в інших компонентів. *Інтерфейсний контракт* для кожної операції самого компонента (або використовуваного ним) визначає правила взаємодії компонентів в системі.

Для опису інтерфейсів призначені мови опису інтерфейсів IDL (Interface Definition Language).

Ці мови містять *операції*, які є викликами відкритих (public) методів класів, що входять до складу компонента, і *операнди* – відкриті (public) поля класів.

2. Компонент повинен працювати в будь-якому середовищі, де є необхідні для його роботи інші компоненти.

Це вимагає наявності певної інфраструктури, яка дозволяє

компонентам знаходити один одного і взаємодіяти по певних правилах.

Набір правил визначення інтерфейсів компонентів і їх реалізацій, а також правил, за якими компоненти працюють в системі і взаємодіють один з одним, прийнято називати *компонентною моделлю* (component model).

У компонентну модель входять правила, що регламентують життєвий цикл компонента, тобто те, через які стани він проходить при своєму існуванні в рамках деякої системи (незавантажений, завантажений, пасивний, активний, знаходиться в кеші і ін.) і як виконуються переходи між цими станами).

Існують декілька компонентних моделей в різних ОС і від різних виробників. Правильно взаємодіяти один з одним можуть лише компоненти, побудовані в рамках *однієї моделі*, оскільки компонентна модель визначає «мову», на якій компоненти можуть спілкуватися один з одним.

Окрім компонентної моделі, для роботи компонентів необхідний деякий набір базових служб (basic services). Наприклад, компоненти повинні уміти знаходити один одного в середовищі, яке, можливо, розподілене на декілька машин. Компоненти повинні уміти передавати один одному дані, знову ж таки, можливо, за допомогою мережевих взаємодій, але реалізації окремих компонентів самі по собі не повинні залежати від вигляду використовуваного зв'язку і від розташування їх партнерів по взаємодії. Набір таких базових, необхідних для функціонування більшості компонентів служб, разом з підтримуваною з їх допомогою компонентною моделлю називається компонентним середовищем (або компонентним каркасом, component framework). Приклади відомих компонентних середовищ — різні реалізації J2EE, .NET Framework, CORBA, COM, COM+, DCOM.

3. Компоненти відрізняються від класів об'єктно-орієнтованих мов.

Клас визначає не лише набір інтерфейсів, що реалізуються, але і саму

їх реалізацію, оскільки він містить код визначуваних операцій. Контракт компонента, найчастіше, не фіксує реалізацію його інтерфейсів. Клас написаний на певній мові програмування. Компонент же не прив'язаний до певної мови, якщо, звичайно, його компонентна модель цього не вимагає, — компонентна модель є для компонентів тим же, чим для класів є мова програмування.

Зазвичай компонент є *крупнішою структурною одиницею*, ніж клас. Реалізація компонента часто складається з декількох тісно зв'язаних один з одним класів.

4. Компоненти не залежать від мов програмування.

Компоненти зберігаються і поширюються у бінарному вигляді і не залежать від мов програмування програмної системи.

Користувач і постачальник компонента можуть використовувати різні мови і бути територіально розподіленими.

2.2 Порядок виконання

2.2.1 Створення DLL-бібліотеки

Створення DLL-бібліотеки розглянемо на конкретному прикладі.

Запустіть Visual Studio 2008, із стартової сторінки перейдіть до створення проекту, виберіть тип проекту «**Class Library**». У вікні створення DLL всі поля заповнені значеннями за замовчанням. Як правило, їх слід перевизначити, задаючи власну інформацію.

У полі Name задайте ім'я DLL –бібліотеки, наприклад, MyLib.

У полі Location вкажіть шлях до каталогу, де зберігатиметься Рішення, що містить проект.

Розмістіть рішення в папці Lab1.

У полі Solution вибраний елемент «Create New Solution», що створює нове Рішення. Альтернативою є елемент списку, який вказує,

що проект може бути доданий до існуючого Рішення.

У вікні Solution Name задано ім'я Рішення. Виберіть Lab1.

Зверніть увагу на інші налаштування, зроблені в цьому вікні, - включений прапорець (за замовчанням) «Create directory for solution», у верхньому віконці із списку можливих каркасів вибраний каркас Framework .Net 3.5. Задавши необхідні установки і клацнувши по кнопці «OK», отримаємо автоматично побудований проект DLL, відкритий в середовищі Visual Studio 2008 .

Імена класів повинні бути змістовними. Змініть ім'я «Class1» на ім'я «MySin». Для цього у вікні коду проекту виділіть ім'я змінної об'єкту, потім в головному меню виберіть пункт *Refactor* і підпункт *Rename*. У вікні, що відкрилося, вкажіть нове ім'я. Тоді будуть показані всі місця, що вимагають перейменування об'єкту. В нашому прикладі буде лише одна очевидна заміна, але в загальному випадку замін багато, так що автоматична заміна всіх входжень корисна.

Наступний крок також продиктований правилом стилю – ім'я класу і ім'я файлу, що зберігає клас, повинні бути однаковими. Перейменування імені файлу робиться безпосередньо у вікні проектів Solution Explorer.

І наступний крок, продиктований також важливим правилом стилю, - додавання коментаря. Для цього в рядку перед заголовком класу слід набрати три слеша (три косі риски). В результаті перед заголовком класу з'явиться заголовний коментар – тег «summary», в який і слід додати короткий, але змістовний опис призначення класу. Теги «summary», якими слід супроводжувати класи, відкриті (public) методи і поля класу відіграють три важливі ролі. Вони полегшують розробку і супровід проекту, роблячи його самодокументованим. Клієнти класу при створенні об'єктів класу отримують інтелектуальну підказку, що

пояснює суть того, що можна робити з об'єктами. Спеціальний інструментарій дозволяє побудувати документацію за проектом, що включає інформацію з тегів «summary».

У нашому випадку коментар до класу MySin може бути достатньо простим – «Обчислення математичних функцій».

Напишемо реалізацію одного методу класу – обчислення функції Sin(x) через ряд Тейлора.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Спочатку напишемо коментарі до методу (у форматі XML). Далі напишемо реалізацію методу. Отримаємо код.

Лістинг 1.1 – Код бібліотечного методу

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyLib
{
///Обчислення математичних функцій

    public class MySin
    {
     /// <summary>
     /// Sin(x)
     /// </summary>
```

```
        /// <param name="x">кут в радіанах – перший
аргумент функції  Sin</param>
        ///<param name="n">показник ступеня – другий
аргумент функції  Sin</param>
        /// <returns>Повертає значення функції Sin для
заданого кута</returns>
        public static double Sin(double x, int n)
        {
            double result = 0;
            for (int i = 0; i < n; i++)
            {
                result = result + (Math.Pow((-1), i) *
Math.Pow(x, (2 * i + 1))) / F(2 * i + 1);
            }
            return result;
        }
        static double F(int n)
        {
            double tmp = 1;
            for (int i = 1; i <= n; i++)
            {
                tmp = tmp * i;
            }
            return tmp;
        }
    }
}
```

Побудуємо Рішення, що містить проект, для чого в Головному меню виберемо пункт Build|Build Solution. В результаті успішної

компіляції буде побудований файл з розширенням dll. Оскільки побудована збірка не містить виконуваного файлу, то безпосередньо запустити наш проект на виконання не вдасться. Побудуємо консольний проект, до якого приєднаємо нашу DLL, і протестуємо, наскільки коректно працюють створені нами методи.

2.2.2 Створення консольного проекту для тестування функції з бібліотеки

Виберіть пункт меню File|New|Project, задайте тип проекту ConsoleApplication, назвіть йому – ConsoleMySin, вкажіть, що проект додається до існуючого Рішення Lab1. В результаті у вже існуюче Рішення додасться ще один проект.

Напишіть код, який викликає реалізовану функцію Sin(x,n), стандартну функцію Sin(x), обчислює похибку і виводить результат на консоль.

Лістинг 2.2 – Консольний застосунок, який викликає бібліотечний метод

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleMySin
{
    class Program
    {
        /// <summary>
```

```
        /// Виклик бібліотечного методу Sin(x,n) з
MySin.dll
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
            Console.WriteLine("Введите x- угол в
радианах");
            double x =
double.Parse(Console.ReadLine());
            Console.WriteLine("Введите показатель
степени n");
            int n = int.Parse(Console.ReadLine());
            //вызов метода вычисления sin(x) из
библиотеки
            double my_sinus = MyLib.MySin.Sin(x, n);
            //вызов метода из класса Math
            double sinus = Math.Sin(x);
            double delta = sinus - my_sinus;
            Console.WriteLine("my_sinus=
{0},sin={1},delta={2}", my_sinus, sinus, delta);
            Console.ReadKey();
        }
    }
}
```

Побудуємо рішення і отримаємо *повідомлення про помилку.* Наша бібліотека не підключена до проекту.

Підключення проекту бібліотеки до консольного проекту.

Для цього додайте посилання на проект з DLL MySin. У вікні

Solution Explorer наведіть покажчик миші до імені консольного проекту і з контекстного меню виберіть пункт меню «Add Reference». Виберіть вкладку «Projects». Оскільки проект MySin включений в Рішення, то він автоматично з'явиться у вікні, Якщо посилання потрібно встановити на проект, не включений в Рішення, то у вікні додавання посилань потрібно вказати шлях до проекту.

Посилання на DLL з'явиться в папці «References» консольного проекту. Тепер проекти зв'язані і з консольного проекту доступний реалізований бібліотечний метод.

Перебудуйте рішення, щоб не було помилок.

Встановлення стартового проекту.

У вікні Solution Explorer наведіть курсор миші на заголовок консольного проекту і виберіть: *Set as StartUp Project*

Після цього його можна запустити на виконання.

2.2.3 Створення Windows-проекту в тому самому рішенні

Виберіть пункт меню File|New|Project, задайте тип *проекту Windows Forms Application***,** назвіть його – WindowsMySin, вкажіть, що проект додається до існуючого Рішення *Lab1*.

На формі створіть 2 текстові поля для введення вхідних параметрів, третє і четверте – для результатів (рисунок 2.1).

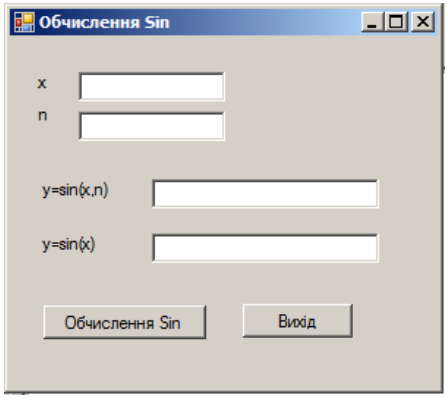


Рис. 2.1 – Форма для обчислення sin(x)

Додамо 2 кнопки. При натисканні кнопки ***"Обчислення Sin"*** виконується виклик функцій, *"Вихід"* – завершення роботи.

Лістинг 2.3 – Код форми

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsMySin
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }

        private void button1_Click(object sender,
EventArgs e)
        {
            double x = double.Parse(txt_x.Text);
            int n = int.Parse(txt_n.Text);
            //виклик метода обчислення sin(x) із
бібліотеки
            double my_sinus = MyLib.MySin.Sin(x, n);
            //вызов метода из класса Math
            double sinus = Math.Sin(x);
            txt_y1.Text = my_sinus.ToString();
            txt_y2.Text = sinus.ToString();

        }

        private void button2_Click(object sender,
EventArgs e)
        {
            this.Close();
        }
    }
}
```

Робимо проект стартовим і запускаємо на виконання. Результат:

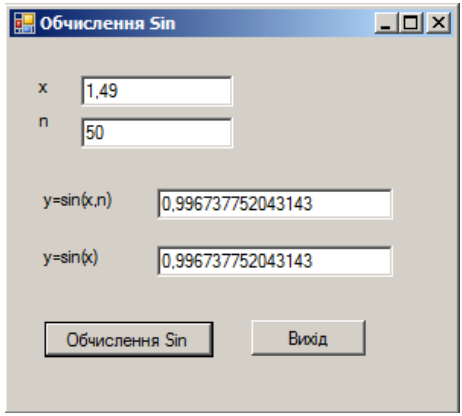


Рис. 2.2 – Результат роботи форми

2.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

2.4 Контрольні запитання

1. Що таке компонентна модель і яке її призначення?
2. Що таке DLL-бібліотека? Чим вона відрізняється від консольного проекту?
3. Чому бібліотечні методи повинні визначатися з модифікатором доступу public?
4. Що є інтерфейсним контрактом бібліотеки?

2.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з рошириенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс, 2007. – 768 с.
6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.
7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.
   Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 3: ФОРМАЛЬНІ ТА ВІЗУАЛЬНІ МЕТОДИ КОНСТРУЮВАННЯ КОМПОНЕНТІВ.

Мета роботи: Ознайомитись із формальними та візуальними методами конструювання компонентів.

3.1 Теоретичні відомості

Для створення конкретної фізичної системи необхідно деяким чином реалізувати всі елементи логічного подання у конкретні матеріальні сутності. Насамперед необхідно розробити вихідний текст програми на деякій мові програмування. При цьому уже в тексті програми передбачається така організація програмного коду, що припускає його розбивку на окремі модулі.

Проте, вихідні тексти програми ще не є остаточною реалізацією проекту, хоча й служать фрагментом його фізичного подання. Очевидно, програмна система може вважатися реалізованою в тому випадку, коли вона буде здатна виконувати функції свого цільового призначення. Для цього необхідно, щоб програмний код системи був реалізований у формі виконуваних модулів, бібліотек класів і процедур, стандартних графічних інтерфейсів, файлів баз даних. Саме ці компоненти є необхідними елементами фізичного подання.

У мові UML для фізичного подання моделей систем використовуються діаграми реалізації, які включають дві окремі канонічні діаграми: діаграму компонентів і діаграму розгортання. Діаграма розгортання описує платформу й обчислювальні засоби для роботи системи.

Діаграма компонентів розробляється для наступних цілей:

1. візуалізації загальної структури вихідного коду програмної системи;
2. специфікації варіанта виконуваної програмної системи;
3. забезпечення багаторазового використання окремих фрагментів програмного коду;
4. подання концептуальної й фізичної схеми баз даних.

У розробці діаграми компонентів беруть участь системні аналітики, архітектори системи й програмісти. Одні компоненти існують на етапі компіляції програмного коду (вихідні модулі), інші на етапі його виконання.

У бізнес-системах діаграми включають відділи, служби й документи, які реально існують у системі. Фізична сутність в UML називається компонентом. Компонент реалізує деякий набір інтерфейсів. На діаграмі він позначається таким способом (рисунок 3.1):

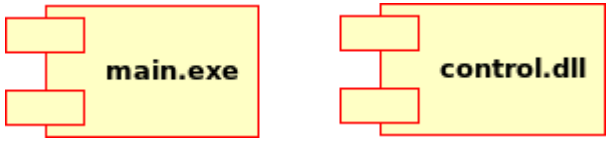


Рисунок 3.1 – Представлення компонентів

Окремий компонент може бути представлений на рівні типу або на рівні компілятора. Їхні імена при цьому розрізняються. У першому випадку записується тільки ім'я типу із заголовної букви. У другому випадку записується ім'я компонента, а потім через двокрапку - ім'я типу. Весь рядок імені в цьому випадку підкреслюється.

Ім'я компонента й ім'я типу може складатися з будь-якого числа букв, цифр і деяких розділових знаків. Іноді для імені екземпляра використовують іншу нотацію: ім'я компонента починається з малої літери, а підкреслення імені не використовується.

У якості простих імен компонентів використовують імена файлів: *.exe, *.dll, *.htm, *.txt, *.doc, *.hlp, *.db, *.mdb, *.h, *.cpp, *.java, *.class, *.pl, *.asp і т.д.

У ряді випадків символ компонента розділяється на секції, щоб явно вказати імена реалізованих у ньому інтерфейсів (розширене позначення компонента).

Оскільки компонент, як елемент фізичної реалізації моделі представляє окремий модуль коду, іноді його коментують із вказівкою додаткових графічних символів, що ілюструють конкретні особливості реалізації: для *.dll - два зубчасті колеса на листі, для *.htm – зафарбоване коло, для *.hlp - рядок тексту із крапками переходу у вигляді зафарбованих прямокутників на листі, для *.cpp - рядок тексту й т.д. Ці додаткові позначення не специфіковані в мові UML. У мові UML виділяють тільки три види компонентів:

1. компоненти розгортання, які забезпечують безпосереднє виконання системою своїх функцій: *.dll, *.htm, *.hlp;
2. компоненти - робочі продукти, як правило - це файли з вихідними текстами програм: *.h, *.cpp і т.і.;
3. компоненти виконання - файли *.exe.

Ці компоненти називають артефактами розробки. Інший спосіб специфікації різних видів компонентів - явна вказівка стереотипу компонента перед його ім'ям:

- бібліотека (library) - для динамічної або статичної бібліотеки;
- таблиця (table) -для таблиці бази даних;
- файл (file) - для файлів з вихідними текстами програм;
- документ (document) - для документів;

- виконуваний (executable) - для компонентів, які можуть виконуватися у вузлі.

Компонентами також є інтерфейси, які зображаються кружечком, що з'єднують із програмними компонентами відрізками ліній без стрілок. При цьому ім'я інтерфейсу повинне починатися з букви “I” й записується поруч із окружністю. Лінія означає, що програмний компонент реалізує цей інтерфейс.

Іншим способом подання інтерфейсу є його зображення у вигляді прямокутника класу зі стереотипом Interface й можливими секціями атрибутів й операцій. Якщо компонент реалізує деякий інтерфейс, то такий інтерфейс називається експортованим. Використовуваний інтерфейс іншого модуля називається імпортованим й зображується пунктирною лінією зі стрілкою, спрямованою до інтерфейсу.

На діаграмі компонентів необхідно вказати інші залежності, зображувані також лінією зі стрілкою, спрямованою від залежного елемента (клієнта) до незалежного елемента (джерела).

Такими залежностями можуть бути, наприклад, зв'язки модулів програми на етапі компіляції й генерації об'єктного коду. В іншому випадку, залежність може відбивати наявність у незалежному компоненті описів класів, які використовуються в залежному компоненті для створення відповідних об'єктів. Дуже важливим випадком відносин залежності є відносини між різними видами компонентів, що означає наступне: внесення змін у вихідні тексти компонентів приводять до змін компонента-клієнта. Нарешті, на діаграмі компонентів можуть бути представлені відносини залежності між компонентами й реалізованими в них класами.

Для позначення такого компонента використовується розширений символ прямокутника компонента. Він ділиться на дві частини. У верхній секції записується ім'я, а в нижній - перераховуються реалізовані класи. Якщо компонент використовує об'єкти, то вони також можуть бути зображені у вигляді пойменованих прямокутників в нижній секції. Подібна вкладеність означає, що виконання компонента тягне виконання відповідних об'єктів, тобто існування компонента протягом часу виконання програми забезпечує існування, а можливо, і доступ всіх вкладених в нього об'єктів.

Рекомендації по побудові діаграми компонентів:

1. максимально враховувати інформацію при поданні моделі системи;
2. точно знати обчислювальні платформи й операційні системи, на яких буде реалізовуватися система;
3. добре представляти можливості обраної мови програмування;
4. визначитися з вибором бази даних або знань;
5. при розподілі програмної системи на фізичні модулі або файли варто домагатися такої реалізації, що забезпечила би можливість повторного використання коду й створення об'єктів тільки при

їхній необхідності. Для цієї мети необхідно більшу частину описів класів, їхніх операцій і методів винести в DDL, залишивши у виконуваних компонентах тільки самі необхідні для ініціалізації програми фрагменти виконуваного коду;

6. після загальної структуризації фізичного подання системи необхідно доповнити її інтерфейсами й схемами бази даних, звертаючи особливу увагу на узгодження різних частин програмної системи, специфікацію таблиць і встановлення зв'язків між таблицями;
7. завершувати побудову діаграми компонентів необхідно встановленням і нанесенням взаємозв'язків між компонентами, в тім числі й відносин реалізації, використовуючи при цьому різні види графічного зображення компонентів. Отримана діаграма повинна ілюструвати всі найважливіші аспекти фізичної реалізації системи, починаючи з особливостей компіляції вихідних текстів програм і закінчуючи виконанням окремих частин програми на етапі її виконання;
8. при розробці діаграми компонентів необхідно використати уже наявні в мові UML компоненти й стереотипи, прибігаючи лише у виняткових випадках до механізмів розширення: додаткові стереотипи й позначені значення.

Приклад діаграми компонентів показано на рис 6.2:

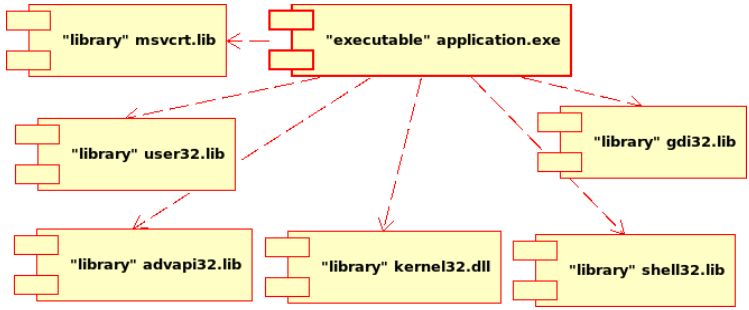


Рисунок 3.2 – Приклад діаграми компонентів

1. Діаграма класів (class diagram)

Діаграма класів служить для подання статичної структури моделі системи в термінах класів об’єктно-орієнтованого програмування. Вона являє собою деякий граф, вершинами якого є елементи типу “класифікатор”, які зв'язані різними типами структурних відносин. На діаграмі можуть бути також інтерфейси, пакети й навіть об'єкти класів, при цьому пакети використовуються для подання більш загальної моделі системи.

Графічно клас зображується у вигляді прямокутника із вказівкою імені класу й списку атрибутів, а також операцій. Атрибути й операції утворюють секції й відокремлюються один від одного й від імені горизонтальними лініями. При проектуванні в прямокутнику спочатку записується ім'я класу, а атрибути й операції записуються по мірі проробляння програмної системи. Іноді використовується й четверта секція для вказівки виняткових ситуацій або визначення семантики класу.

Ім'я класу повинно відображати призначення класу й записуватися на англійській або російській/українській мові з великої літери. Ім'я абстрактного класу записується курсивом. Кожен атрибут класу має певну область видимості:

+ загальнодоступний (public);

захищений (protected);

- закритий (private).

Іноді область видимості зовсім не вказується. Квантор видимості ставиться перед ім'ям атрибута. Після імені атрибута указується його кратність у класі й тип. Кратність задається діапазоном цілих чисел, розділених двома крапками. Верхня границя діапазону може бути зірочкою, що означає довільне ціле число. Тип атрибута відповідає типу мови реалізації. Операція класу представляє деякий сервіс, що надається даним класом. У записі операції вказується квантор видимості, ім'я операції, список параметрів у круглих дужках, а також тип значення, що повертає. У фігурних дужках далі можна вказати тип операції:

{ concurrency = sequential } - послідовна;

{ concurrency = concurrent } - можливе розпаралелення;

{ concurrency = guarded } - охоронювана.

Між класами можуть бути відносини залежності (---→), асоціації (-), агрегації(<>-), узагальнення(<-). На рис. 6.3 приведено приклад діаграми класів, що складається з трьох класів:

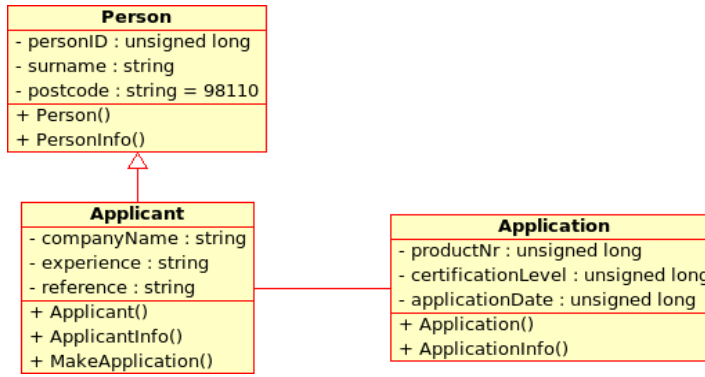


Рис. 6.3. Приклад діаграми класів

3.2 Порядок виконання

Розробити діаграми компонентів і діаграми класів з детальним описом програми (лабораторної роботи №2).

3.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

3.4 Контрольні запитання

1. Що таке компонентна модель і яке її призначення?
2. Що таке DLL-бібліотека? Чим вона відрізняється від консольного проекту?
3. Чому бібліотечні методи повинні визначатися з модифікатором доступу public?
4. Що є інтерфейсним контрактом бібліотеки?

3.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з роширенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс, 2007. – 768 с.
6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.
7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.
Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 4: БРОКЕРИ ОБ'ЄКТНИХ ЗАПИТІВ.

Мета роботи: Познайомитися з технологією CORBA. Ознайомитись з мовою IDL та описом інтерфейсів. Освоїти класи бібліотеки org.omg. Застосувати отримані знання на практиці.

4.1 Теоретичні відомості

Узагальнена архітектура побудови брокера об'єктних запитів розроблена для підтримки інтеграції найрізноманітніших об'єктних систем. Специфікація CORBA встановлює принципи створення брокера об'єктних Запитів**,** які і допускають таку інтеграцію.

Запит надсилається від клієнта до сервера. *Клієнт* це застосунок, або щось інше, що виконує операцію над об'єктом, а *реалізація об'єкта* - це код і дані, які насправді виконують цю операцію. ORB здатний виконати всі дії, необхідні для знаходження реалізації зазначеного об'єкта, підготовці цієї реалізації до обробки запиту і передачі даних, що відносяться до запиту. Інтерфейс, що надається клієнтові абсолютно не залежить від місця розташування реалізації об'єкта, мови програмування, на якому він написаний або яких-небудь інших аспектів, які не впливають на визначення інтерфейсу для даного об'єкта.

При визначенні конкретної архітектури брокер об'єктних запитів зовсім необов'язково повинен бути реалізований як один компонент, але кожна реалізація повинна реалізовувати три категорії операцій:

- Операції, які однакові для всіх реалізацій ORB-а.
- Операції, специфічні для конкретного об'єктного типу.
- Операції, специфічні для окремих видів реалізацій об'єктів.

Різні реалізації ORB-а можуть підтримувати різні види реалізацій, а різні адаптери об'єктів можуть забезпечувати різні набори сервісів для

клієнта і реалізацій. Ядро Брокера об'єктних Запитів забезпечує основні механізми для маніпуляцій об'єктами і виконання запитів. Специфікація CORBA призначається для підтримки різних механізмів реалізації об'єктів, тому структура ядра не визначається. Замість цього задається набір інтерфейсних функцій, які повинні бути присутніми в кожній реалізації ORB-а і тим самим маскують відмінності між різними реалізаціями брокера об'єктних Запитів.

Об'єкти

Система об'єктів забезпечує клієнта набором сервісів. Клієнт здатний запросити деякий сервіс. Об'єкт - це щось, що забезпечує один або більше сервісів, які клієнт може запросити.

Приклад брокера об'єктних запитів

Доступно широке безліч способів реалізації конкретних ORB-ів. Далі будуть наведені приклади таких реалізацій. Слід мати на увазі, що конкретний ORB може бути реалізований відразу декількома способами.

ORB, що включається в клієнтське і серверне застосування

Якщо мається відповідний механізм комунікацій, то можлива реалізація ORB-а у вигляді набору підпрограм як з боку клієнта, так і з боку реалізації об'єкта. Виклики методів можуть транслюватися в роботу із засобами взаємодії процесів (Inter Process Communication - IPC).

ORB, виконаний у вигляді сервера

З метою забезпечення централізованого збору і керування всілякою інформацією, ORB може бути реалізований у вигляді окремого застосунка. Взаємодіючі застосунка встановлюють контакт з ORB-ом за допомогою нормальних механізмів IPC.

ORB як частина системи

Для підвищення надійності, захисту даних і досягнення кращої продуктивності ORB може бути реалізований як частина операційної системи. При цьому посилання на об'єкт можуть бути зроблені постійними,

таким чином, зменшуючи час, необхідний для обробки кожного запиту. При реалізації ORB-а як частини операційної системи можливі всілякі види оптимізації, такі як уникнути кодування і декодування даних, якщо клієнт і сервер знаходяться на одній і тій же машині.

ORB, заснований на бібліотеках

Якщо код об'єкта займає невеликий об'єм і не вимагає ніяких додаткових коштів, то він може бути виконаний у вигляді бібліотеки. При цьому всі заглушки насправді будуть бути справжніми методами. При цьому передбачається, що маючи доступ до даних реалізації, клієнт не зруйнує ці дані.

Реалізації об'єктів

Реалізація об'єкта забезпечує саме поняття об'єкта, зазвичай задаючи дані для конкретного екземпляра об'єкта і код для виконання методів об'єкта. Часто реалізація буде використовувати інші об'єкти або допоміжні програми для забезпечення функціонування об'єктів. У деяких випадках виконання операції над об'єктом тягне якісь побічні дії не над об'єктами.

Конкретний ORB може підтримувати широкий набір об'єктних реалізацій: окремі сервери, бібліотеки, об'єктно-орієнтовані системи управління базами даних та ін За допомогою використання додаткових адаптерів Об'єктів теоретично можна підтримувати будь-яку реалізацію об'єкта.

Адаптери об'єктів

Адаптер об'єктів – це первинний шлях для забезпечення сервісу конкретною реалізацією об'єкта. Передбачається, що є кілька адаптерів об'єктів, кожний з яких забезпечує доступ до об'єктів певного виду.

Сервіси, які забезпечуються ORB-ом за допомогою адаптерів об'єктів часто включають: генерацію та інтерпретацію посилань на об'єкти, виклик методів, активацію і деактивацію реалізацій об'єктів, а також реєстрацію конкретних реалізацій і відображення об'єктних посилань і реалізацій.

Інформація, яка знаходиться в кожному зі сховищ може бути довільно змінена в будь-який момент часу за допомогою методів, забезпечуваних реалізацією ORB-а. Однак, необережне зміна, зроблена під час роботи може призвести порушити цілісність інформації, що знаходиться в кожному з сховищ і зробити неможливим подальше функціонування ORB-а.

Скелет реалізації

Для конкретного відображення і, можливо, використовуваного адаптера об'єктів визначається свій порядок виклику методів кожного об'єкта. Цей інтерфейс в загальному випадку є інтерфейсів зворотних викликів. При необхідності ORB викликає необхідні процедури.

Динамічна обробка запитів

Також доступний інтерфейс для динамічної обробки вступників запитів. У цьому випадку реалізація об'єкта взаємодіє із заданим інтерфейсом за аналогією з інтерфейсом динамічних викликів.

Підпрограми динамічної відпрацювання запитів можуть викликатися як за допомогою інтерфейсу динамічних викликів, так і за допомогою процедур-заглушок, кожен метод дає однаковий результат.

Запити

Клієнт запитує сервіси ініціюванням запитів. Запит - це подія, тобто дію, що відбувається в конкретний момент. Із запитом пов'язана інформація, яка складається із операції, об'єкта, у якої запитується сервіс, нуля або більше дійсних параметрів виклику і необов'язковий контекст запиту. Форма запиту - це опис або шаблон, який може бути виконаний довільну кількість разів. Форма запиту визначається відображенням для конкретної мови програмування. Альтернативною формою запиту є використання Інтерфейсу Динамічних Викликів, який дозволяє створити запит, додати аргументи і виконати запит. Під значенням розуміється допустимий параметр запиту. Значення яке визначає об'єкт, називається посиланням на об'єкт, пов'язаної з конкретним екземпляром об'єкта. Виконання запиту

викликає виконання відповідного сервісу. Після завершення запиту клієнту повертається результат запиту (якщо він є). У разі ненормального завершення запиту клієнту повертається виняток. Виняток може містити специфічні параметри, специфічні для даного типу виключень.

Параметри

Параметр характеризується режимом передачі і своїм типом. Режим визначає, чи повинно передаватися значення параметра від клієнта до сервера (in), від сервера до клієнта (out) або в обох напрямках (inout).

- Повертає значення. Якщо є значення, що повертається, то воно розглядається як параметр типу out.
- Винятки. Виняток свідчить про те, що операція не була успішно виконана. Виняток може містити додаткову інформацію, специфічну для конкретного винятку.
- Контекст. Контекст запиту забезпечує передачу додаткової, специфічною для операції інформації, яка може вплинути на виконання запиту.

Параметри запитів визначаються їх позицією. Параметри можуть бути вхідні, вихідні і вхідні і вихідні одночасно. Як результат запит може повернути одне значення, як, втім, і будь-які вихідні параметри. У разі виникнення виключення значення всіх вихідних параметрів не визначено.

Інтерфейси

Інтерфейс - це опис безлічі можливих операцій, які клієнт може виконувати над об'єктом. Об'єкт задовольняє інтерфейсу, якщо він може бути зазначений як кінцевим об'єктом для кожного потенційного запиту, описаного в інтерфейсі. Типом інтерфейсу задовольняють тільки об'єктні типи.

Інтерфейс ORB-а

Інтерфейс ORB-а є функціям, викликуваним безпосередньо у Брокера об'єктних Запитів і ідентичним для всіх ORB-ів, не залежних від конкретного об'єкта або адаптера об'єктів. Але так як більшість дій з

об'єктами виконується за допомогою адаптерів об'єктів, існує всього декілька загальних операцій, які можуть бути виконані над кожним об'єктом. Ці операції можуть викликатися як клієнтом, так і реалізацією об'єкта.

4.2 Порядок виконання

Розробити поштовий клієнт і сервер. Клієнт – віконне застосунок, яке дозволятиме відсилати і отримувати з сервера повідомлення. Ідентифікація клієнтів на сервері, протокол передачі повідомлень – на розсуд студентів. Сервер може бути консольним застосунком. Зберігати повідомлення можна в текстовому файлі. Рекомендується зробити сервер багатопоточність.
Для взаємодії клієнта і сервера використовувати технологію CORBA.
Як доповнення пропонується сервер або клієнт реалізувати не на Java.

4.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

4.4 Контрольні запитання

1. Що таке CORBA?
2. Що таке IDL? Для чого він потрібен?
3. Як здійснюється взаємодія клієнта і сервера в CORBA?
4. Як передаються дані між ними?

5. Для чого потрібен сервер імен?
6. Як запускається CORBA-сервер?

4.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з роширенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс, 2007. – 768 с.
6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.
7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е

изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.

9. Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 5: МОНІТОРИ ОБРОБЛЕННЯ ТРАНЗАКЦІЙ.

Мета роботи: Вивчити способи забезпечення збереженості і несуперечності інформації, що знаходиться в базі даних за допомогою трансакцій, блокувань і механізму обробки помилок.

5.1 Теоретичні відомості

Використання моніторів обробки транзакцій є одним з методів досягнення більш високої продуктивності для наявної конфігурації, особливо в режимі клієнт / сервер. Іноді монітори обробки транзакцій виявляються дуже корисними для створення гетерогенних баз даних, що дозволяють зберігати деякі дані в одному форматі (наприклад, Oracle на Sun), а інші дані в іншому (можливо Ingres на VAX або IMS на мейнфрейми IBM). Крім того, деякі TP-монітори надають сервіс для компонента уявлення. Добре відомий TP-монітор компанії IBM - Customer Information Control System (CISC). Декілька реалізацій CICS (MicroFocus, XDB, VI Systems, Integris) доступні в даний час на більшості апаратних платформ. Іншими відомими моніторами є Tuxedo / T компанії USL, TopEnd від NCR і Encina від Transarc.

TP-монітори являють собою проміжний шар програмного забезпечення, який розташовується між застосунком і системою або системами СКБД. При цьому програма має бути модифіковано так, щоб вона могла видавати транзакції, написані на мові монітора транзакцій, а не звертатися прямо до бази даних за допомогою звичайних механізмів (подібних різним формам вбудованого SQL). Програмісти прикладних систем є також відповідальними за складання файлу опису, який відображає транзакції в певні звернення до бази даних рідною мовою звернень СКБД

(майже для всіх СКБД під UNIX це SQL).

Гнучкість доступу до даних

Використання моніторів транзакцій практично не накладає жодних обмежень на різноманіття або складність запитів доступу до СКБД. Наприклад, цілком осмисленою виявляється транзакція, що видає запит на набір даних з бази даних DB2 на мейнфрейми IBM, що працює під управлінням ОС MVS, а інший набір даних з локальної бази даних Sybase, і потім зливаються обидва набори разом для представлення застосунком. У результаті створюється ілюзія того, що дані зберігаються в уніфікованій базі даних, розміщеної в одному і тому ж "сховищі даних" (data warehause).

У зв'язку з тим, що у багатьох "старих" (legacy) системах використовуються монітори обробки транзакцій CICS, мігруюча частина або всі бази даних, пов'язані з цими системами, можуть працювати з невеликими змінами існуючих застосунків. Все що потрібно - це фізичне перенесення даних на нову платформу і модифікація опису транзакцій для використання даних на новому місці. Однак складність такого перенесення не повинна недооцінюватися, оскільки він часто потребує внесення змін до подання даних (трансляцію COBOL "PIC 9 (12) V99S" в C + + float) і організацію даних (наприклад, з мережевої архітектури IMS в реляційну архітектуру, використовувану майже повсюдно СКБД на базі UNIX). Але можливість зберегти частини, пов'язані з прикладною обробкою і представленням існуючих застосунків істотно скорочує складність і ризик, пов'язані з таким перенесенням.

Питання продуктивності

Окрім досягнення певної гнучкості за рахунок використання ТР-моніторів, така організація виявляється вигідною і з точки зору збільшення продуктивності системи. ТР-монітор завжди являє собою багатопотокову програму. Оскільки ТР-монітор відкриває своє власне з'єднання з СКБД, одночасно усуваючи необхідність виконання кожним прикладним процесом

прямих запитів до СКБД, число одночасно працюючих користувачів СКБД істотно скорочується. В переважній більшості випадків СКБД обслуговує тільки одного "користувача" - ТР-монітор. Це особливо важливо, коли СКБД відноситься до класу "2N", оскільки в цьому випадку використовується тільки один тіньовий процес (для забезпечення з'єднання з ТР-монітором), а не по одному процесу на кожний процес кінцевого користувача (рисунок 5.1). Це може істотно скоротити накладні витрати, пов'язані з перемиканням контексту на сервері СКБД.

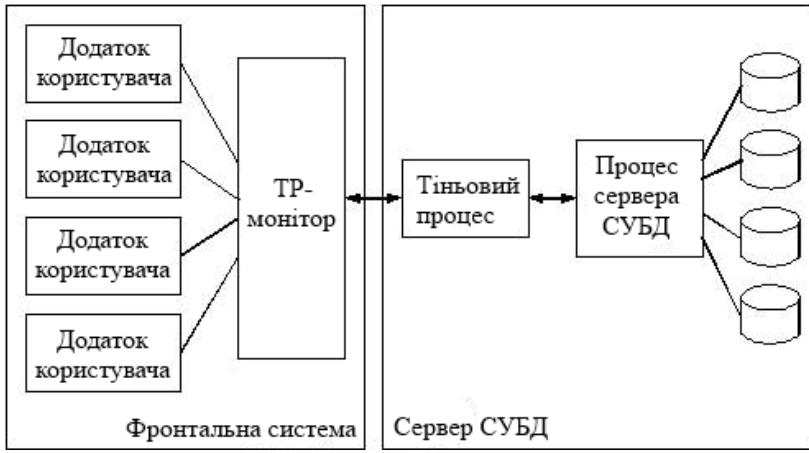


Рисунок 5.1 – Взаємодія компонентів СУБД (СКБД) із фронтендом

Системи СКБД клієнт / сервер сконфігуровані для роботи з моніторами обробки транзакцій. Тіньовий процес на серверній системі з'являється, коли СКБД використовує архітектуру "2N".

ТР-монітори дозволяють також поліпшити продуктивність за рахунок скорочення обсягу інформації, що пересилається між СКБД та прикладним процесом. Оскільки певну частину кожної транзакції становлять тільки мінімально необхідні дані, загальний обсяг пересилання даних зазвичай може бути скорочений. Це особливо важливо, коли клієнт і сервер з'єднані

між собою за допомогою досить зайнятої мережі та / або мережі з низькою смугою пропускання, подібної глобальної мережі на супутникових каналах зв'язку.

Рекомендації:

- TP-монітори можуть розглядатися як кандидати для включення до складу системи, коли мають початкові коди застосунка.
- TP-монітори слід використовувати для інтеграції в корені відмінних джерел інформації бази даних.
- TP-монітори особливо корисні для скорочення трафіку клієнт / сервер на мережах з низькою смугою пропускання або глобальних мережах.

Застосування системи TP-монітора слід розглядати, коли повинна обслуговуватися велика кількість користувачів і використовувана СКБД воліє або вимагає реалізації архітектури "2N".

5.2 Порядок виконання

1. Запустити застосунок Query Analyzer і активізувати свій проект бази даних.
2. Створити просту трансакцію, яка змінює інформацію, що міститься в таблиці AddressType і наочно демонструє стан даних на всіх етапах свого виконання.
3. Переглянути список ресурсів, блокованих користувачем, використовуючи застосунок Enterprise Manager.
4. Спробувати внести запис у таблицю Client без відповідного їй батьківського запису в таблиці Person і проаналізувати повідомлення MS SQL 2000.
5. Реалізувати механізм обробки помилок у збереженій процедурі PersonClientlnsert, змінивши текст відображуваних користувачем

повідомлень.

6. Здійснити внесення інформації в таблиці Person і Client за допомогою запуску збереженої процедури PersonClientInsert і переконатися в наявності нових повідомлень.
7. Реалізувати механізм обробки помилок у збереженій процедурі PersonEmpInsert, змінивши текст відображуваних користувачем повідомлень.
8. Здійснити внесення інформації в таблиці Person і Employee за допомогою запуску збереженої процедури PersonEmpInsert і переконатися в наявності нових повідомлень.

5.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

5.4 Контрольні запитання

1. Значення терміна "трансакція".
2. Сутність ACID-правил.
3. Основні варіанти визначення трансакцій.
4. Склад оператора явного оголошення трансакції.
5. Призначення і принцип роботи двофазної фіксації змін.
6. Способи підвищення ефективності трансакцій.
7. Суть і призначення блокувань.

8. Типові проблеми, пов'язані з одночасним доступом до даних декількох користувачів.
9. Що таке грануляція?
10. Ресурси, що дозволяють блокувати SQL Server 2000.
11. Режими блокування ресурсів.
12. Що таке взаємне блокування?
13. Базові правила при розробці стратегії блокування даних.
14. Призначення механізму обробки помилок.
15. Спосіб визначення номера помилки.

5.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з роширенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс, 2007. – 768 с.
6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ;

пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.

7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.
9. Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 6: ВИКЛИКИ ВІДДАЛЕНИХ ПРОЦЕДУР.

Мета роботи: Ознайомитися з RMI. Написати клієнт і сервер RMI.

6.1 Теоретичні відомості

Виклик віддалених процедур (*Remote procedure call, **RPC***) — протокол, що дозволяє програмі, запущеній на одному комп'ютері бути викликаною на іншому комп'ютері без написання безпосередньо коду для цієї операції.

Концепція віддаленого виклику процедур

Ідея виклику віддалених процедур (Remote Procedure Call — RPC) полягає у розширенні добре відомого і зрозумілого механізму передачі управління і даних усередині програми, що виконується на одній машині, на передачу керування й даних через мережу. Засоби віддаленого виклику процедур призначені для полегшення організації розподілених обчислень. Найбільша ефективність використання RPC досягається в тих застосунках, в яких існує інтерактивний зв'язок між віддаленими компонентами з невеликим часом відповідей і відносно малою кількістю переданих даних. Такі програми називаються RPC-орієнтованими.

Існують безліч технологій, що забезпечують RPC:

- Sun RPC (бінарний протокол на базі TCP та UDP)
- Net Remoting (бінарний протокол на базі TCP, UDP, HTTP)
- XML-RPC (текстовий протокол на базі HTTP)
- SOAP — Simple Object Access Protocol (текстовий протокол на базі HTTP)
- Java RMI — Java Remote Method Invocation
- JSON-RPC JavaScript Object Remote Procedure Calls (текстовий, на базі HTTP)

Характерними рисами виклику локальних процедур є:

- Асиметричність, тобто одна з взаємодіючих сторін є ініціатором
- Синхронність, тобто виконання викликаючої процедури призупиняється з моменту видачі запиту і відновлюється тільки після повернення з викликаної процедури.

Реалізація віддалених викликів істотно складніше реалізації викликів локальних процедур. Почнемо з того, що оскільки викликаюча і викликана процедури виконуються на різних машинах, то вони мають різні адресні простори, і це створює проблеми при передачі параметрів і результатів, особливо якщо машини не ідентичні. Так як RPC не може розраховувати на розподілену пам'ять, то це означає, що параметри RPC не повинні містити покажчиків на клітинки нестекової пам'яті і що значення параметрів повинні копіюватися з одного комп'ютера на інший.

Але в реалізації RPC беруть участь як мінімум два процеси — по одному в кожній машині. У випадку, якщо один з них аварійно завершиться, можуть виникнути такі ситуації: при аварії викликаючої процедури віддалено викликані процедури стануть «осиротілими», а при аварійному завершенні віддалених процедур стануть «знедоленими батьками» викликаючі процедури, які будуть безрезультатно чекати відповіді від віддалених процедур.

Крім того, існує ряд проблем, пов'язаних з неоднорідністю мов програмування та операційних середовищ: структури даних і структури виклику процедур, підтримувані в будь-якому однією мовою програмування, не підтримуються точно так само у всіх інших мовах.

Ці та деякі інші проблеми вирішує широко поширена технологія RPC, що лежить в основі багатьох розподілених операційних систем, наприклад Plan 9.

Базові операції RPC

Щоб зрозуміти роботу RPC, розглянемо спочатку виконання виклику

локальної процедури у звичайній машині, що працює автономно. Нехай це, наприклад, буде системний виклик: count = read (fd, buf, nbytes); де fd — ціле число, buf — масив символів, nbytes — ціле число.

Щоб здійснити виклик, викликаюча процедура заштовхує параметри в стек у зворотному порядку (Мал. 1). Після того, як виклик read виконаний, він поміщає значення, що повертається в регістр, переміщує адресу повернення і повертає управління викликаючій процедурі, яка вибирає параметри з стека, повертаючи його в початковий стан. Зауважимо, що в мові С параметри можуть викликатися або по посиланню (by name), або за значенням (by value). По відношенню до викликаючої процедури параметри-значення є ініціалізованими локальними змінними. Викликана процедура може змінити їх, і це не вплине на значення оригіналів цих змінних в викликаючій процедурі. Якщо в викликану процедуру передається покажчик на змінну, то зміна значення цієї змінної викликаної процедури тягне зміну значення цієї змінної і для викликаючої процедури. Цей факт дуже істотний для RPC.

Існує також інший механізм передачі параметрів, який не використовується в мові С. Він називається call-by-copy/restore і полягає в необхідності копіювання викликаючою програмою змінних в стек у вигляді значень, а потім копіювання назад після виконання виклику поверх оригінальних значень викликаючої процедури.

Рішення про те, який механізм передачі параметрів використовувати, приймається розробниками мови. Іноді це залежить від типу переданих даних. У мові С, наприклад, цілі та інші скалярні дані завжди передаються за значенням, а масиви — по посиланню. Ідея, покладена в основу RPC, полягає в тому, щоб зробити виклик віддаленої процедури по можливості схожим до виклику локальної процедури. Іншими словами — зробити RPC прозорим: викликаючій процедурі не потрібно знати, що викликається процедура знаходиться на іншій машині, і навпаки.

RPC досягає прозорості наступним шляхом. Коли викликана процедура дійсно є віддалена, в бібліотеку поміщається замість локальної процедури інша версія процедури, яка називається клієнтським стабом (stub — заглушка). Подібно оригінальної процедурі, стаб викликається з використанням викликаючої послідовності (як на мал. 1), так само відбувається переривання при зверненні до ядра. Тільки на відміну від оригінальної процедури він не поміщає параметри в регістри і не запитує у ядра дані, замість цього він формує повідомлення для відправки ядру віддаленої машини.

Етапи виконання RPC

Взаємодія програмних компонентів при виконанні віддаленого виклику процедури ілюструється малюнком 2. Після того, як клієнтський стаб був викликаний програмою-клієнтом, його першим завданням є заповнення буфера відправлені повідомленням. У деяких системах клієнтський стаб має єдиний буфер фіксованої довжини, що заповнюється щоразу з самого початку при вступі кожного нового запиту. В інших системах буфер повідомлення являє собою пул буферів для окремих полів повідомлення, причому деякі з цих буферів вже заповнені. Цей метод особливо підходить для тих випадків, коли пакет має формат, що складається з великої кількості полів, але значення багатьох з цих полів не змінюються від виклику до виклику.

Потім параметри повинні бути перетворені у відповідний формат і вставлені в буфер повідомлення. До цього моменту повідомлення готове до передачі, тому виконується переривання за викликом ядра.

Коли ядро отримує управління, воно перемикає контексти, зберігає регістри процесора і карту пам'яті (дескриптори сторінок), встановлює нову карту пам'яті, яка буде використовуватися для роботи в режимі ядра. Оскільки контексти ядра і користувача розрізняються, ядро має точно скопіювати повідомлення в свій власний адресний простір, так, щоб мати до

нього доступ, запам'ятати адресу призначення (а, можливо, і інші поля заголовка), а також воно має передати його мережевому інтерфейсу. На цьому завершується робота на клієнтській стороні. Включається таймер передачі, і ядро може або виконувати циклічне опитування наявності відповіді, або передати управління планувальником, який обере будь-який інший процес на виконання. У першому випадку прискорюється виконання запиту, але відсутнє мультипрограмування.

На стороні сервера біти, що надходять, поміщаються приймаючої апаратурою або у вбудований буфер, або в оперативну пам'ять. Коли вся інформація буде отримана, генерується переривання. Обробник переривання перевіряє правильність даних пакета і визначає, якому стабу слід їх передати. Якщо жоден із стабів не очікує цей пакет, обробник повинен або помістити його в буфер, або взагалі відмовитися від нього. Якщо є очікуючий стаб, то повідомлення копіюється йому. Нарешті, виконується переключення контекстів, в результаті чого відновлюються регістри і карта пам'яті, приймаючи ті значення, які вони мали в момент, коли стаб зробив виклик receive.

Тепер починає роботу серверний стаб. Він розпаковує параметри і поміщає їх відповідним чином в стек. Коли все готово, виконується виклик сервера. Після виконання процедури сервер передає результати клієнту. Для цього виконуються всі описані вище етапи, тільки в зворотному порядку.

14 етапів виконання RPC:

Клієнт 1.

Виклик стабу 2.

Підготувати буфер 3.

Упакувати параметри 4.

Заповнити поле заголовка 5.

Обчислити контрольну суму в повідомленні 6.

Переривання до ядра 7.

Черга пакету на виконання 8.

Передача повідомлення контролеру по шині QBUS 9.

Сервер 10.

Отримати пакет від контролера 11.

Процедура обробки переривання 12.

Обчислення контрольної суми 13.

Переключення контексту в простір користувача 14.

Динамічне зв'язування

Розглянемо питання про те, як клієнт задає місце розташування сервера. Одним з методів вирішення цієї проблеми є безпосереднє використання адресу сервера в клієнтській програмі. Недолік такого підходу — його надзвичайна негнучкість: при переміщенні сервера, або при збільшенні числа серверів, або при зміні інтерфейсу у всіх цих та багатьох інших випадках необхідно перекомпілювати всі програми, які використовували жорстке завдання адреси сервера. Для того, щоб уникнути всіх цих проблем, в деяких розподілених системах використовується так зване динамічне зв'язування.

Початковим моментом для динамічного зв'язування є формальне визначення (специфікація) сервера. Специфікація містить ім'я файлу-сервера, номер версії і список процедур-послуг, що надаються даним сервером для клієнтів. Для кожної процедури дається опис її параметрів із зазначенням того, чи є даний параметр вхідним чи вихідним щодо сервера. Деякі параметри можуть бути одночасно вхідними і вихідними — наприклад, деякий масив, який надсилається клієнтом на сервер, модифікується там, а потім повертається назад клієнтові.

Формальна специфікація сервера використовується в якості вихідних даних для програми-генератора стабів, яка створює як клієнтські, так і серверні стаби. Потім вони поміщаються у відповідні бібліотеки. Коли користувальницька (клієнтська) програма викликає якусь процедуру,

визначену в специфікації сервера, відповідна стаб-процедура зв'язується з двійковим кодом програми.

При запуску сервера найпершою його дією є передача свого серверного інтерфейсу спеціальною програмою, так званим binder'ом. Цей процес, відомий як процес реєстрації сервера, включає передачу сервером свого імені, номера версії, унікального ідентифікатора і описувача місцезнаходження сервера. Описувач системно незалежний і може представляти собою IP, Ethernet, X.500 або ще яку-небудь адресу. Крім того, він може містити й іншу інформацію, наприклад дані, що стосуються аутентифікації.

Коли клієнт викликає одну з віддалених процедур перший раз, наприклад, read, клієнтський стаб бачить, що він ще не підключений до сервера, і надсилає повідомлення binder-програмі з проханням про імпорт інтерфейсу потрібної версії потрібного сервера. Якщо такий сервер існує, то binder передає описувач і унікальний ідентифікатор клієнтському стабу.

Клієнтський стаб при посилці повідомлення із запитом використовує в якості адреси описувач. У повідомленні містяться параметри і унікальний ідентифікатор, який ядро сервера використовує для того, щоб направити надійшло повідомлення в потрібний сервер у випадку, якщо їх декілька на цій машині.

Цей метод, що полягає в імпорті / експорті інтерфейсів, володіє високою гнучкістю. Наприклад, може бути кілька серверів, що підтримують один і той же інтерфейс, і клієнти розподіляються по серверах випадковим чином. У рамках цього методу стає можливим періодичне опитування серверів, аналіз їх працездатності та, у разі відмови, автоматичне відключення, що підвищує загальну відмовостійкість системи. Цей метод може також підтримувати аутентифікацію клієнта. Наприклад, сервер може визначити, що він може бути використаний тільки клієнтами з певного списку.

Однак у динамічного зв'язування є недоліки, наприклад, додаткові накладні витрати (тимчасові витрати) на експорт та імпорт інтерфейсів. Величина цих витрат може бути значною, тому що багато клієнтські процеси існують короткий час, а при кожному старті процесу процедура імпорту інтерфейсу повинна бути знову виконана. Крім того, у великих розподілених системах може бути вузьким місцем програма binder, а створення декількох програм аналогічного призначення також збільшує накладні витрати на створення і синхронізацію процесів.

Семантика RPC в разі відмов

В ідеалі RPC повинен функціонувати правильно і у випадку відмов. Розглянемо наступні класи відмов:

- Клієнт не може визначити місцезнаходження сервера, наприклад, у разі відмови потрібного сервера, або через те, що програма клієнта була скомпільована давно і використовувала стару версію інтерфейсу сервера. У цьому випадку у відповідь на запит клієнта надходить повідомлення, що містить код помилки.
- Втрачено запит від клієнта до сервера. Найпростіше рішення — через певний час повторити запит.
- Втрачено повідомлення-відповідь від сервера до клієнта. Цей варіант складніший від попереднього, так як деякі процедури не є ідемпотентними. Ідемпотентною називається процедура, запит на виконання якої можна повторити кілька разів, і результат при цьому не зміниться. Прикладом такої процедури може бути читання файлу. Але ось процедура зняття деякої суми з банківського рахунку не є ідемпотентною, і в разі втрати відповіді повторний запит може істотно змінити стан рахунку клієнта. Одним з можливих рішень є приведення всіх процедур до ідемпотентного виду. Однак на практиці це не завжди вдається, тому може бути використаний інший метод — послідовна нумерація всіх запитів клієнтським ядром. Ядро сервера запам'ятовує номер самого останнього

запиту від кожного з клієнтів, і при отриманні кожного запиту виконує аналіз — чи є цей запит первинним або повторним.

- Сервер зазнав аварію після отримання запиту. Тут також важливо властивість Ідемпотентний, але на жаль не може бути застосований підхід з нумерацією запитів. У цьому випадку має значення, коли відбулася відмова — до чи після виконання операції. Але клієнтське ядро не може розпізнати ці ситуації, для нього відомо тільки те, що час відповіді вичерпано.

Існує три підходи до цієї проблеми:

- Чекати до тих пір, поки сервер не перезавантажиться і намагатися виконати операцію знову. Цей підхід гарантує, що RPC був виконаний до кінця принаймні один раз, а можливо і більше.
- Відразу повідомити застосунка про помилку. Цей підхід гарантує, що RPC був виконаний не більше одного разу.
- Третій підхід не гарантує нічого. Коли сервер відмовляє, клієнтові не надається ніякої підтримки. RPC може бути або не виконано взагалі, чи виконано багато разів. В усякому разі цей спосіб дуже легко реалізувати.

6.2 Порядок виконання

1. Написати і відкомпілюватиJavaкод для інтерфейсів.
2. Написати і відкомпілюватиJavaкод для класів реалізації.
3. Створити файли класів заглушки і скелета з класів реалізації.
4. НаписатиJavaкод програмихоста для віддаленого обслуговування.
5. РозробитиJavaкод для клієнтської програмиRMI.
6. ВстановитиіzапуститиRMIсистему.

6.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

6.4 Контрольні запитання

1. Поняття виклику віддаленої процедури. Навести приклад.
2. Роль RMI в архітектурі Java EE.

6.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з роширенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс,

2007. – 768 с.

6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.
7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.
9. Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 7: ВИБІР ЗАСТОСУВАННЯ, СЕРВІСІВ, КОМПОНЕНТІВ І ПРОТОКОЛІВ ЗВ'ЯЗКУ.

Мета роботи: Навчитися вибирати середовища розробки, сервіси. компоненти і протоколи зв’язку відповідно до певних типів проекту.

7.1 Теоретичні відомості

Eclipse як найпопулярніше середовище розробки під Java, підтримується IBM і практично всі компанії пишуть під нього плагіни. Вибір плагінів великий, як і покриття функціоналу. Тут існує менеджер пакунків Marven (є в коробці з NetBeans). Цей збирач настроюється таким чином, що він знає які бібліотеки потрібні для роботи програми та знає з яких репозитаріїв їх забирати. Це зручно. Про те що для роботи яких те застосунків потрібні величезні набори інших бібліотек це звичайно пригнічує, але коріння цього в ідеології Java. Для прикладу нам потрібний візуальний редактор UML. Для нього існує багато різних плагінів. Eclipse, як і Netbeans, дозволяє встановлювати плагіни, використовуючи репозитарії плагінів. Є пункт меню Install new software, де настроюється доступ до репозитарію і потім виводиться список всіх наявних у ньому плагінів. Недоліком рішення є надто громіздка ієрархія залежностей, які необхідно встановлювати додатково.

NetBeans підтримується компанією Sun. Вона працює «з коробки». Із дистрибутивом для веб розробки поставляється відразу і сервер застосунків Glassfish. NetBeans, у порівнянні з Eclipse, має більш доброзичливий і симпатичний інтерфейс. Для роботи з UML є два плагіни: перший – офіційний, і найкращий – від Sun (навіть існує функція для Reverse Enginering). NetBeans схожа своєю ідеологією на VisualStudio, яка є еталоном. Реалізація багатьох речей на NetBeans набагато простіша.

Згідно з проведеним у 2011 році компанією ZeroTurnaround опитуванням 65% використовують Eclipse, 22% Intellij IDEA, 12% NetBeans, 4% MyEclipse, 3% Oracle JDeveloper (таблиця 7.1).

Таблиця 7.1 – Результати опитування BZ Research

Опитування BZ Research						
IDE	**2002**	**2003**	**2004**	**2005**	**2006**	**2007**
Eclipse	—	35%	56%	65%	70%	63%
NetBeans	—	13%	18%	18%	23%	24%
Oracle JDeveloper	25%	21%	17%	15%	19%	20%
IBM WebSphere Studio	—	25%	22%	20%	22%	19%
Oracle Workshop for WebLogic	12%	12%	9%	7%	—	9%
JetBrains IntelliJ IDEA	4%	8%	12%	11%	9%	9%
Borland JBuilder	35%	37%	24%	19%	14%	5%
Microsoft Visual J++/J#.NET	25%	15%	9%	7%	—	—

7.2 Порядок виконання

1. Створити Java Web-сервіс за допомогою середовища розробки Eclipse.
2. Створити Java Web-сервіс за допомогою середовища розробки NetBeans.
3. Описати переваги і недоліки кожного з використаних середовищ.

7.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

7.4 Контрольні запитання

1. Опишіть середовище розробки Eclipse.
2. Опишіть середовище розробки NetBeans.
3. Яке середовище розробки Java-застосунків є найпопулярнішим?
4. Які середовища розробки Java-застосунків ви знаєте?

7.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з роширенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн :

УниверсалПресс, 2007. – 768 с.

6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.

7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.

8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.

9. Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.

ЛАБОРАТОРНА РОБОТА 8: ОСНОВИ JAVA EE.

Мета роботи: Познайомитися з технологією Java EE.

8.1 Теоретичні відомості

До найбільш потужних технологій Java, які дозволили піднятися їй з самих низів до таких вершин належать:

1. Технологія платформенної незалежності.
2. Технологія забезпечення захисту користувачів від потенційно небезпечних дії.
3. Технологія аплетів.
4. Технологія сервлетів.
5. Технологія JSP-сторінок.
6. Технологія багатопоточності.
7. Технологія роботи з мережами.
8. Технологія JavaBeans та інші.

Технологія платформенної незалежності в Java будується на основі використання байт-коду, тобто проміжного представлення програм, яке не залежить від архітектури комп'ютера чи операційної системи, на які виконуватимуться застосунки. Це проміжне представлення перетворюється у виконуваний код на комп'ютері користувача за рахунок віртуальної машини Java, яка на сьогоднішній день включається у всі операційні системи виробниками цих ОС. При цьому розробники нових ОС широко використовують самі технології Java для побудови своїх систем та надання їм більшої надійності.

Технологія забезпечення захисту від потенційно небезпечних дій здійснюється на основі аналізу віртуальною машиною Java виконуваного

коду, та дозволу чи забороні виконання цього коду. Java обмежує вас у декількох ключових областях, і в такий спосіб сприяє виявленню помилок на ранніх стадіях розробки програми. У той же час у ній відсутні багато джерел помилок, властивих іншим мовам програмування.

Більшість використовуваних сьогодні програм відмовляють в одній із двох ситуацій: при виділенні пам'яті, або при виникненні виняткових ситуацій. У традиційних середовищах програмування при розподілі пам'яті програмісту приходиться самому стежити за усією використовуваною в програмі пам'яттю, не забуваючи звільняти її в міру того, коли вона стає лишньою. Найчастіше програмісти забувають звільняти захоплену ними пам'ять або, що ще гірше, звільняють ту пам'ять, що усе ще використовується якою-небудь частиною програми. Виняткові ситуації в традиційних середовищах програмування часто виникають у таких, наприклад, випадках, як ділення на нуль або спроба відкрити неіснуючий файл, і їх приходиться обробляти за допомогою складних конструкцій. Java фактично знімає обидві ці проблеми, використовуючи збирач сміття для звільнення незайнятої пам'яті й убудовані об'єктно-орієнтовані засоби для обробки виняткових ситуацій.

Однією з найбільших переваг мови Java - можливість створення аплетів, маленьких програм, які працюють всередині WEB-броузера. Проте, на аплети накладені певні обмеження в зв'язку з тим, що вони виконуються на комп'ютері користувача.

Обмеження аплету:

1. Аплет не має доступу до жорсткого диску. Проте для них існує система цифрових підписів, за допомогою якої користувач може визначати чи дані аплети отримані з надійних джерел, а отже може зняти більшість обмежень.

2. Аплетам необхідно певний час, щоб загрузитися з Інтернету. Для зменшення цього часу всі дані, які необхідно аплету для роботи, як правило

включають в jar-архів, що дозволяє швидше загрузити аплет.

Переваги мови Java:

1. Для аплетів немає необхідності встановлювати їх як інші програми. Цим можна скористатися, коли необхідно постійно загружати обновлені версії програм.
2. Немає необхідності хвилюватись, що загружений аплет виконає потенційно небезпечні дії. Основні дії, які можуть привести до втрати важливої інформації, пошкодження чи зміна вмісту файлів для аплетів є заборонені.

Багатопоточна архітектура

Реалізацію багатопоточної архітектури найпростіше уявити для системи, в якій існує декілька центральних обчислювальних процесорів. У цьому випадку для кожного з них можна виділити задачу, яку він буде виконувати. В результаті декілька задач будуть обслуговуватись одночасно.

Але виникає питання – яким чином забезпечується багатопоточність у системах з одним центральним процесором, який виконує лише одне обчислювання в один момент часу? В таких системах застосовується процедура квантування часу (time-slicing). Час поділяється на невеликі інтервали. Перед початком кожного інтервалу приймається рішення, який потік виконання буде відпрацьовуватись на протязі цього кванту часу. За рахунок такого частого переключення між задачами емулюється багатопоточна архітектура.

Процедура квантування часу підтримує пріоритети (priority) задач. В Java пріоритет представляється цілим числом. Чим більше число, тим вище пріоритет. Суворих правил роботи з пріоритетами немає, кожна реалізація може вести себе різноманітно на різних платформах. Однак є загальне правило – потік із більш високим пріоритетом буде отримувати більшу кількість квантів часу на виконання, і таким чином зможе швидше виконати свої дії і реагувати на данні, що надходять.

Базові класи для роботи з потоками

Клас Thread

Потік виконання в Java представляється екземпляром класу `Thread`. Для того, щоб написати свій потік виконання, необхідно наслідуватись від цього класу і перевизначати метод `run()`. Наприклад,

```
public class MyThread extends Thread {
     public void run() {
       //
       long sum=0;
       for (int I=0; i<1000; i++) {
         sum+=I;
       }
       System.out.printin(sum);
     }
}
```

Метод `run()` складає дії, які повинні виконуватись в новому потоці виконання. Щоб запустити його, необхідно створити екземпляр класу-спадкоємця, і викликати успадкований метод `start()`, який повідомляє віртуальній машині, що необхідно запустити новий потік виконання і почати в ньому виконувати метод `run()`.

```
MyThread t = new MyThread();
t.start();
```

Робота з пріоритетами

Розглянемо, як в Java потокам можна назначати пріоритети. Для цього у класі Thread існують методи getPriority() і setPriority(), а також оголошені три константи:

- `MIN_PRIORITY`
- `MAX_PRIORITY`
- `NORM_PRIORITY`

З назви очевидно, що їх значення описує мінімальне, максимальне і нормальне (за замовчуванням) значення пріоритету.

Розглянемо наступний приклад:

```
public class ThreadTest implements Runnable {
     public void run() {
       double calc;
       for (int i=0; i<50000; i++) {
            calc=Math.sin(i*i);
            if (i%10000==0) {
              System.out.println(getName()+"count:"
              +i/10000);
            }
     }
}

public String getName() {
  return Thread.currentThread().getName();
}

public static void main(String s[]) {
//
     Thread t[] = new Thread[3];
     for (int i=0; i<t.length; i++) {
       t[i]=new Thread(new ThreadTest(),
"Thread"+i);
     }

//
```

```
    for (int i=0; i<t.length; i++) {
        t[i].start();
        System.out.println(t[i].getName()+"started");
    }
}
```

У прикладі використовується декілька нових методів класу `Thread`:

`getName()`

Зверніть увагу, що конструктору класу Thread передається два параметра. До реалізації Runnable додається рядок. Це ім'я потоку, яке використовується тільки для спрощення його ідентифікації. Імена декількох потоків можуть співпадати. Якщо його не задавати, то Java генерує простий рядок виду «Thread» та номер потоку (обчислюється простим лічильником). Саме це ім'я повертається методом getName(). Його можна змінити за допомогою методу setName().

`currentThread()`

Цей статичний метод дозволяє в будь-якому місці коду отримати посилання на об'єкт класу Thread, який представляє поточний потік виконання. Результат роботи такої програми буде мати наступний вигляд:

```
Thread 0 started
Thread 1 started
Thread 2 started
Thread 0 counts 0
Thread 1 counts 0
Thread 2 counts 0
Thread 0 counts 1
Thread 1 counts 1
Thread 2 counts 1
Thread 0 counts 2
Thread 2 counts 2
```

```
Thread 1 counts 2
Thread 2 counts 3
Thread 0 counts 3
Thread 1 counts 3
Thread 2 counts 4
Thread 0 counts 4
Thread 1 counts 4
```

Можна бачити, що всі три потоки були запущені один за одним, і почали проводити обчислення. Також бачимо, що потоки виконуються без певного порядку випадковим чином. Тим паче, в середньому вони рухаються з однією швидкістю, ніхто не відстає, і ніхто не здоганяє.

8.2 Порядок виконання

Створити довільну програму для запуску декількох потоків та перевірити їх коректну роботу.

8.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

8.4 Контрольні запитання

1. Що таке потік програми?
2. Що таке багатопоточність?

3. Що таке синхронізація потоків?
4. Як організовується потік програми в Java?

1.

8.5 Перелік джерел

1. Вандер Вер Эмили JavaScript для "чайников": Уч. пос./Под ред. В.М.Неумоина .-3-е изд.-М.:Изд. дом Вильямс,2001. – 304 с.-(Ил.) .-5-8459-0134-0 Шифр: 681.3 Авторський знак: В17
2. Янг Майкл Дж.Visual C++6. Полное руководство. Т.1.- К.:Ирина,1999 .-544 Шифр: 681.3.06 Авторський знак: Я60
3. Янг Майкл Дж. Visual C++6.Полное руководство. Т.2 .- К.:Ирина,1999 .-560 Шифр: 681.3.06 Авторський знак: Я60
4. Монсон Хейфел Р. Enterprise JavaBeans / Р. Хейфел Монсон ; пер. с англ. – 3-е изд. – СПб. : СимволПлюс, 2002. – 672 с. : ил.
5. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы : учебник для вузов / В. Г. Олифер, Н. А. Олифер. – 4-е изд. – СПб. : Питер, 2010. – 944 с. : ил.
6. Таменбаум Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. – СПб. : Питер, 2003. – 877 с. : ил.
7. Heffelfinger D. Java EE 6 with GlassFish 3 Application Server / D. Heffelfinger – Packt Publishing.
8. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ЛАБОРАТОРНА РОБОТА 9: СЕРВЛЕТИ ТА JSP.

Мета роботи: Ознайомитися із поняттям сервлети. Написати сервлет.

9.1 Теоретичні відомості

У Java-програмуванні прийнято розрізняти Java-застосунки та Java-аплети. Перше з точки зору програмування цінне саме по собі, тому що нічого крім області виконання Java не вимагає. А друге цікаво тільки в контексті здатності браузерів виконувати Java-код аплетів.

Є, однак, і третя життя Java - це програми, які виконуються http-серверами. Ось вони і називаються сервлети. Звичайно, сам по собі сервер не може виконувати Java-код. В сервер вбудовується модуль, який викликає Java-машину, виконуючу сервлети.

Як аплет не є самостійною Java-програмою, так і сервлет нею також не є. І те й інше, умовно кажучи, «латочки», які програміст вбудовує в більш загальний Java-код. Прикладному програмісту дається можливість написати тіла заздалегідь визначених через механізм інтерфейсів методів.

Сервлети в деякому сенсі схожі на CGI-скрипти. Як і в скриптах у сервлетов жорстко визначений механізм отримання даних, тобто вхідний потік, і механізм формування вихідного потоку. При цьому сервлетам крім усього іншого ще дозволено отримувати точно такі ж значення, як ті, що визначені в змінних оточення скриптів.

У свою чергу сервлети є базою для Java Server Pages. Про це механізмі кажуть, що він є повнофункціональним аналогом Active Server Pages компанії Microsoft. Насправді механізм JSP ширше ASP, і набагато більш гнучкий.

JPS - це послідовний розвиток ідеї вбудовування програмного коду в HTML і XML сторінки. Тільки це не проста мова SSI і не VBScript, а

повноцінні Java-конструкції.

Насправді це тільки з точки зору людини, яка створює JSP, йдеться про «нашінковке» кодом старого доброго HTML. З точки зору сервера така сторінка - це сервлет. При чому реально трансляція файлу спасає в пам'яті Java-код відбувається тільки при першому зверненні до JSP-сторінці. При всіх наступних зверненнях ніякої трансляції відбуватися вже не буде, а буде використовуватися раніше відтрансльований сервлет.

З вище сказаного слідують ті плюси, про які так любить розмірковувати фахівці, які «захворіли» даною технологією, не всі вони є абсолютними «плюсами», але тим не менше їх слід перерахувати:

- Ефективність. Не треба створювати новий процес. У стандартній ситуації все вже в «голові» і відразу починає виконуватися. Зрозуміло, що тут сервлети порівнюють зі CGI_скріптамі. В принципі, навіть інтегрований в браузер Perl, модуль mod_perl в Apache, наприклад, не повинен теоретично забезпечувати такої продуктивності, яку обіцяють сервлети.
- Єдність середовища розробки програмного забезпечення. Вивчив Java і пиши собі код на здоров'я. Не треба вчити інші мови програмування, наприклад, Perl, VBScript або C. Правда, сам цей мова, м'яко кажучи, не дуже проста.
- Потужність. Java - мова багата. Написати на ньому можна багато чого. Правда, Perl в цьому сенсі не гірше. У всякому разі, там, де не треба розписувати віконечка і кнопочки екранних інтерфейсів. А програмування на стороні сервера - це якраз такий випадок. Perl та Java - це основні мови розробки Web-застосунків. З точки зору складності вони приблизно однакові. Правда, кожен по своєму.
- Переносимість. Інтерпретовані програми по великому рахунку всі перенесимі. Був би на відповідній платформі потрібний

інтерпретатор. Java краще тільки в тому плані, що для нього централізовано підтримується єдина специфікація мови.

- Безкоштовність. Може бути для наших закордонних колег це і велика перевага, але для нас, де 80% рідного Web-а працює на безкоштовному Apache, це норма. Але добре, що й за кордонами магістральний шлях розвитку поки залишається безкоштовним.

9.2 Порядок виконання

Для виконання вам знадобиться сервер JakartaTomcatServer, що можна скачати на web-вузлі java.sun.com. Встановіть дистрибутив. Оголосіть перемінні середовища JAVA_HOME = «<path>/jdk…» і CATALINA_HOME =”<path>jakarta-tomcat…” (ControlPanel / System / Advanced /EnvironmentVariables). Тепер Ви має встановлений tomcat. Якщо до вас треба щоб якісь бібліотеки підключалися в CLASSPATH – покладіть в каталог lib. За умовчанням сервер використовує порт 8080. Для запуску Tomcat використовуйте binstartup.bat, для зупинки – binshutdown.bat. Кореневим каталогом для Ваших докуметів буде webapps/ROOT/

Для класів – webapps/ROOT/WEB-INF/classes/. Наприклад, клас myServlet, що у цьому каталозі викликається: localhost:8080 /servlet/myServlet. Клас myPackage.myServlet (що у пакеті myPackage і в каталозі webapps/ ROOT/ WEB-INF/ classes/myPackage/) викликається: localhost:8080/servlet/myPackage.myServlet

Приклади

Наведемо кілька прикладівсервлетов.

```
//HelloWorld.java виводить на браузер напис HelloWorld
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;
```

```
public class HelloWorld extends HttpServlet {
    public void doGet (HttpServletRequestrequest,
HttpServletResponseresponse)
    throws IOException, ServletException
    {
        response.setContentType («>text/html»);
        PrintWriter out =response.getWriter();
        out.println («<html>»);
        out.println («<>body>»);
        out.println («<>head>»);
        out.println («<>title>HelloWorld!
    </title>»);
        out.println («</>head>»);
        out.println («<>body>»);
        out.println («<>h1>HelloWorld!</h1>»);
        out.println («</>body>»);
        out.println («</html>»);
    }
}
```

Яким ви бачите,сервлет обробляє Get–запит браузера.

`response.setContentType` («text/html»); – встановлюємо тип відповіді

`PrintWriter out = response.getWriter();` – відкриваємо потік, з якого відбувається запис вихідних даних на браузер.

Наведемо приклад сервлета, обробного вхідні параметри.

```
 // код HTML-сторінки, викликає сервлет
<>formaction= «>RequestParamExample»method=POST>
FirstName:
```

```
     <>inputtype=textsize=20name=firstname>
     <>br>
     >LastName:
     <>inputtype=textsize=20name=lastname>
     <>br>
     <>inputtype=submit>
     </>form>
     //RequestParamExample.java отримує вхідні
параметри і виводить на браузер їх значення
     import java.io.*;
     import java.util.*;
     import javax.servlet.*;
     import javax.servlet.http.*;
     public class RequestParamExample extends
HttpServlet {
          public void doPost
     (HttpServletRequestrequest,
     HttpServletResponseres)
          throws IOException, ServletException
          {
               Enumeration e =
          request.getParameterNames();
               PrintWriter out = res.getWriter();
               while (e.hasMoreElements()) {
                    Stringname =
               (String)e.nextElement();
                    Stringvalue =
               request.getParameter(name);
                    out.println (name + «=» +value);
               }
```

```
        }
    }
```

Тут ми з викликаної сторінки отримуємо параметри `firstname` і `lastname`.

9.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

9.4 Контрольні запитання

1. Що таке Java-servlet?
2. Які класи Java використовуються для написання сервлетів?

9.5 Перелік джерел

1. Вандер Вер Эмили JavaScript для "чайников": Уч. пос./Под ред. В.М.Неумоина .-3-е изд.-М.:Изд. дом Вильямс,2001. – 304 с.-(Ил.) .-5-8459-0134-0 Шифр: 681.3 Авторський знак: В17
2. Янг Майкл Дж. Visual C++6.Полное руководство. Т.2 .-К.:Ирина,1999 .-560 Шифр: 681.3.06 Авторський знак: Я60
3. Монсон Хейфел Р. Enterprise JavaBeans / Р. Хейфел Монсон ; пер. с англ. – 3-е изд. – СПб. : СимволПлюс, 2002. – 672 с. : ил.
4. Олифер В.Г. Компьютерные сети. Принципы, технологии,

протоколы : учебник для вузов / В. Г. Олифер, Н. А. Олифер. – 4-е изд. – СПб. : Питер, 2010. – 944 с. : ил.

5. Перри Б. Java сервлеты и JSP: сборник рецептов // Б. Перри ; пер с англ. – М. : Кудиц-пресс, 2006. – 768 с.
6. Heffelfinger D. Java EE 6 with GlassFish 3 Application Server / D. Heffelfinger – Packt Publishing.
7. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ЛАБОРАТОРНА РОБОТА 10: РІВЕНЬ БІЗНЕС-ЛОГІКИ В JAVA EE.

Мета роботи: Навчитись розрізняти рівні бізнес-логіки, зокрема в Java EE. Навчитись розробляти застосунки із використанням різних рівнів бізнес-логіки.

10.1 Теоретичні відомості

Існують такі рівні *бізнес-логіки* технології Java EE:

- *Верхній рівень* (рівень *дизайнерів*, *верстальників*) – рівень представлення, визначає інтерфейс користувача. Це може бути HTML, CSS, Flash, зображення та інші технології, за допомогою яких створюється зовнішній вигляд застосунка. Рекомендується вивчати: JSF.
- *Середній рівень* (рівень *програмістів*) – рівень бізнес-логіки. Рівень на якому і відбувається програмування. Створюються класи, реалізується бізнес логіка програми (наприклад, авторизація користувача в системі, обробка тих чи інших подій). Рекомендується вивчати: EJB.
- *Нижній рівень* (також рівень *програмістів*) – рівень даних, забезпечує зберігання даних, найчастіше в базах даних. Рекомендується вивчати: JPA.

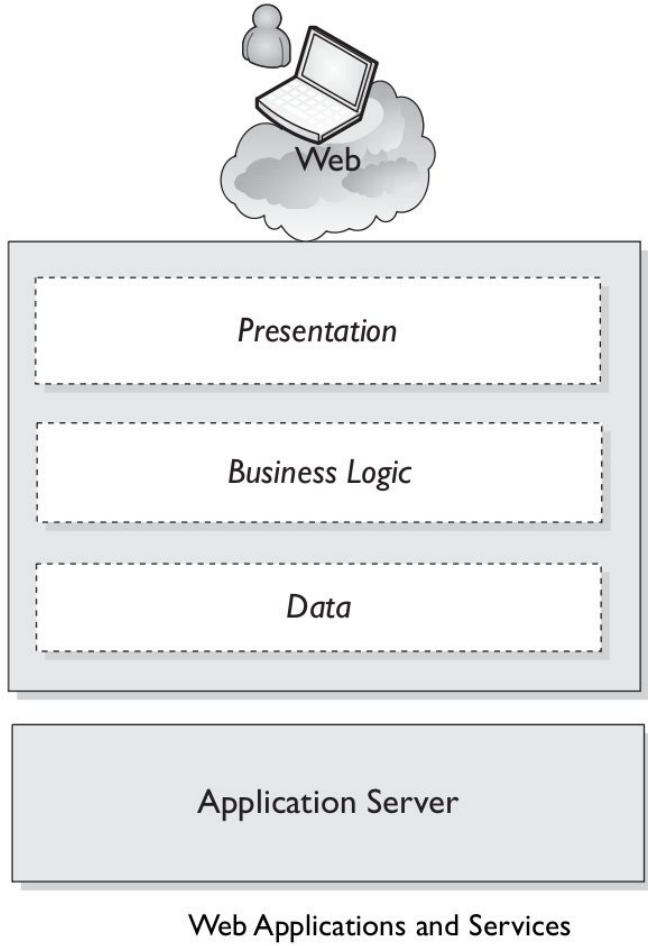


Рисунок 10.1 – Web служби-застосунки

Технології Java EE, за допомогою яких можна реалізувати роботу того чи іншого рівня наведено на рисунку 10.2.

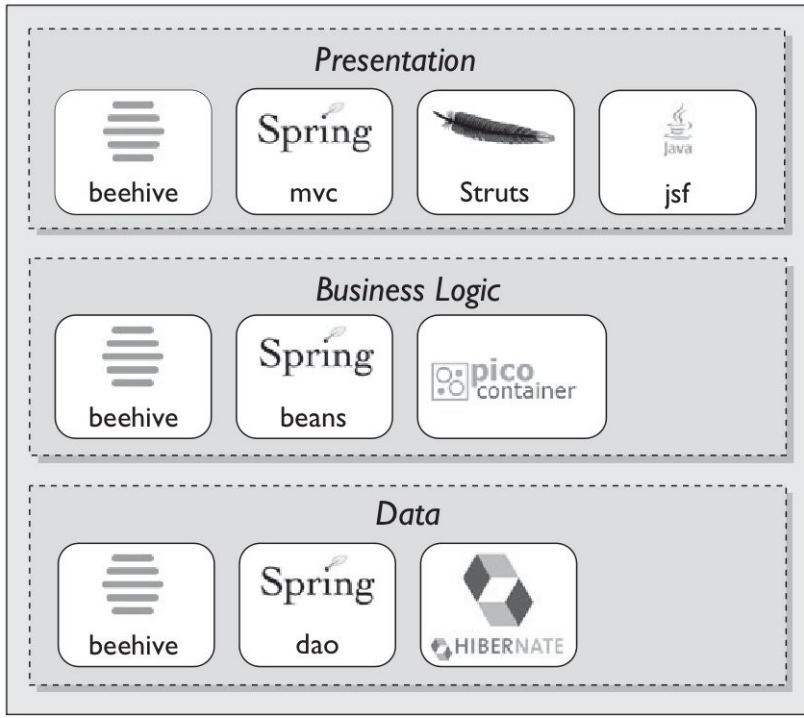


Рисунок 10.2 – Техонології Java EE, за допомогою яких можна реалізувати роботу того чи іншого рівня

На рисунку 13.2 на рівні даних відсутній *EclipseLink* (попередня його назва *Oracle TopLink*. *TopLink* був переданий Eclipse корпорацією Oracle).

13.2 Enterprise JavaBeans (EJB)

У минулому зміни, що вносяться в специфікації EJB, робилися без втрати сумісності. При цьому у версії 3,0 був серйозно змінений синтаксис

програмної моделі EJB. Тим не менш минулі програмні моделі - з версій 2,1 і більш ранніх - як і раніше підтримуються. Навіть програмна модель версії 3,0 залишилась несумісна, що дозволяє реалізовувати клієнтські і серверні компоненти на різних версіях специфікації:

- Сесійні об'єкти на даний момент досить *стабільні*, і фундаментальних змін їх поведінки не передбачається, хоча деякі нові функції можуть з'явитися.
- Також не повинно бути змін в об'єктах, керованих повідомленнями (*Message-Driven Beans*).
- Частина специфікації, що відповідає за CMP / BMP-збереженню, досить стабільна, але її можуть замінити нові програмні інтерфейси JPA (*Java Persistence API*) у випадках, коли потрібно збереження даних в реляційних базах даних, і архітектура JCA (*Java EE Connector Architecture*), у решти випадках.
- Одним з можливих серйозних змін в наступній версії специфікації може стати інтеграція сесійних об'єктів з об'єктами, керованими повідомленнями. Таким чином, з'явиться новий тип компонентів, які можуть взаємодіяти як з синхронною, так і по асинхронною схемою. Але на даний момент все говорить про те, що розвиток технології буде відбуватися розумно і без втрати сумісності.

10.2 Порядок виконання

1. Розробити програмний компонент (JavaBean) “тип іграшки”.
2. Розробити програмний компонент (JavaBean) для зберігання інформації про іграшку.
3. Для вказання типу іграшки і назви додаткового поля використати композицію.
4. Створити клас “Модель таблиці” для відображення та модифікації

інформації про іграшки, що зберігаються у колекції.

5. Використовуючи компоненти, створені в пп.1-4 розробити програму, що матиме зручний інтерфейс користувача і дозволятиме виводити перелік іграшок, ціна яких не перевищує вказану і які підходять дітям 5 років у порядку зростання ціни.

10.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

10.4 Контрольні запитання

1. Які рівні бізнес-логіки Jave EE ви знаєте?
2. За допомогою яких технологій Java EE можна реалізувати роботу того чи іншого рівня бізнес-логіки?

10.5 Перелік джерел

1. Монсон Хейфел Р. Enterprise JavaBeans / Р. Хейфел Монсон ; пер. с англ. – 3-е изд. – СПб. : СимволПлюс, 2002. – 672 с. : ил.
2. Heffelfinger D. Java EE 6 with GlassFish 3 Application Server / D. Heffelfinger – Packt Publishing.
3. Mike Keith, ol. Pro JPA 2. Mastering the Java™ Persistence API / Mike Keith. Merrick Schincari. – New York : Apress, 2009. – 238 p.
4. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної

дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ЛАБОРАТОРНА РОБОТА 11: ОБ'ЄКТНО-РЕЛЯЦІЙНЕ ВІДОБРАЖЕННЯ В JAVA EE.

Мета роботи: Познайомитися з технологією ORM в Java EE.

11.1 Теоретичні 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 зберігати 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;

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.

11.2 Порядок виконання

Реалізуємо ORM-сумісний застосунок для зображення відношення *has_many* в парі із відношенням *belongs_to* із використанням мови PHP.

Припустимо в нас є модель повідомлень із назвою Post, яка має відношення до моделі User. Оскільки кожне повідомлення обов'язково має свого єдиного автора, то, з точки зору моделі Post, вона відноситься до моделі User belongs_to. З точки зору ж моделі User, вона відноситься до моделі Post has_many, оскільки користувач може мати більше одного повідомлення.

На практиці перевірено що зазначення $_primary_key у моделях є регістрозалежним! Тобто якщо у вас в базі даних $_primary_key = 'ID', а ви у моделях пропишете $_primary_key= 'id', Кохана не видасть винятків, але і не буде працювати коректно

Створимо відповідні моделі.

APPPATH/classes/model/post.php

```
    <?php defined('SYSPATH') or die('No direct access
allowed.');

    class Model_Post extends ORM {
        protected $_db_group = 'forum'; // Назва групи
конфігурації бази даних
        protected $_table_name = 'posts'; // Назва
таблиці
        protected $_primary_key = 'pid'; // Назва
первинного ключа в таблиці posts
        protected $_belongs_to = array(
                'author' => array(     // Аліас author
для таблиці users
                        'model'       => 'user', //
Назва пов'язаної моделі
                        'foreign_key' => 'uid', //
Зовнішній ключ в моделі posts
                ),
        );
    }
```

APPPATH/classes/model/user.php

```
    <?php defined('SYSPATH') or die('No direct access
allowed.');

    class Model_User extends ORM {
        protected $_db_group = 'forum'; // Назва групи
конфігурації бази даних
        protected $_table_name = 'users'; // Назва
```

```
таблиці
        protected $_primary_key = 'uid'; // Назва
первинного ключа в таблиці users
        protected $_has_many = array(
                'posts' => array(      // Аліас posts
для таблиці posts
                        'model'       => 'post', //
Назва пов'язаної моделі
                        'foreign_key' => 'uid', //
Зовнішній ключ в моделі posts
                ),
        );
    }
```

Припустимо, що в нас конфігураційний файл для бази даних, до якого ми будемо звертатись із моделей по назві групи конфігурації (в даному випадку forum), такий:

APPPATH/config/database.php

```
    <?php defined('SYSPATH') or die('No direct access
allowed.');

    return array
    (
       'forum' => array
       (
               'type'       => 'mysql',
               'connection' => array(
                       'hostname'   => 'localhost',
```

```
                        'database'   =>
'database_name',
                        'username'   => 'Username',
                        'password'   => 'password',
                        'persistent' => TRUE,
                ),
                'table_prefix' => '',
                'charset'      => 'utf8',
                'caching'      => FALSE,
                'profiling'    => TRUE,
        ),
    );
```

В наступному коді дані розпечатуються прямо з контролера лише для прикладу. На практиці дані потрібно передавати в представлення (View).

Припустимо що нам потрібно викликати ці моделі із контролера Forum:

APPPATH/classes/controller/forum.php

```
    <?php defined('SYSPATH') or die('No direct access
allowed.');

    class Controller_Forum extends Controller {

        public function action_index()
        {
            // Створюємо об'єкт моделі Post із primary
key = 5
            $posts = ORM::factory('post', 5);

            // Зі сторони моделі Post звертаємось до
```

```
aліасу author (для таблиці user)
            // і витягуємо значення з існуючого поля
login
            echo $posts->author->login.':<br>';

            // Зі сторони моделі Post витягуємо
значення з існуючого поля post_c в таблиці posts
            echo $posts->post_c.'<hr>';

                // Створюємо об'єкт моделі User із
primary key = 1
                $user = ORM::factory('user', 1);

                // Зі сторони моделі User витягуємо
значення з існуючого поля login в таблиці users
                echo $user->login.':<br>';

                // Зі сторони моделі User звертаємось
до аліасу posts (для таблиці posts)
            // і витягуємо всі значення з існуючого
поля post_c
                foreach($user->posts->find_all() as
$posts_all)
                {
                        echo $posts_all->post_c.'<br>';
                }
        }
    }
```

11.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

11.4 Контрольні запитання

1. Що таке ORM?
2. Які проблеми зберігання об’єктів в реляційній базі данних?
3. Які основні особливості ORM ви знаєте?

11.5 Перелік джерел

1. Mike Keith, ol. Pro JPA 2. Mastering the Java™ Persistence API / Mike Keith. Merrick Schincari. – New York : Apress, 2009. – 238 p.
2. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ЛАБОРАТОРНА РОБОТА 12: ОСНОВНІ ЕЛЕМЕНТИ ТЕХНОЛОГІЇ JSF.

Мета роботи: Отримати необхідні навики роботи з JSF.

12.1 Теоретичні відомості

Технологія JSP (Java Server Pages) – це технологія створення серверних сторінок Java в специфікації JSP як розширення Java Servlet API для генерації динамічних веб-сторінок на веб-сервері. Крос-платформа альтернативна технології ASP корпорації Microsoft.

Альтернативою технології JSP є специфікація Sun – JSF Java Server Faces, призначена для опису правил створення веб-застосувань зі зручним для користувача інтерфейсом та компонентів, що реалізують цей інтерфейс.

Засоби розробки Java-застоснків, що підтримують зазначену специфікацію і створюють веб-застосування, засновані на J2EE з тією ж швидкістю і ступенем зручності, що і засоби розробки .NET-застосувань.

JSF JSR-127 - це технологія для побудови користувальницьких веб-інтерфейсів на основі компонентів.

Java Server Faces включає в себе:

- набір API для відображення UI компонентів, управління їх станом,
- відстеження подій,
- перевірки користувальницького введення,
- визначення навігації між сторінками та підтримки інтернаціоналізації.
- бібліотеку тегів JSP для відображення JSF інтерфейсу за допомогою JSP-сторінок.

JSF надзвичайно гнучка, так як не прив'язує розробника до

конкретної мови розмітки, протоколів і клієнтським пристроям.

Основна мета JSF - чітко розділити логіку застосунку від представлення, при цьому дозволяючи легко використовувати їх разом. Архітектура дозволяє кожному члену команди з розробки веб-застосунка зайнятися своєю частиною програми і надає просту програмну модель для з'єднання частин в єдине ціле.

12.2 Порядок виконання

1. Вибрати File New Project Ctrl-Shift-N;-Shift-N на Mac, щоб відкрити майстер створення проекту. У списку Категорії виберіть Web; в рамках проектів, виберіть веб-застосунків. Натисніть кнопку Далі.
2. Назва проекту jAstrologer вкажіть місце для проекту на комп'ютері, а потім натисніть кнопку Далі.
3. На третьому кроці майстра налаштування сервера вкажіть сервер і Java версію, яка буде використовуватися з проектом, або прийняти налаштування за замовчуванням. Натисніть кнопку Далі.
4. У кроці Framework, виберіть JavaServer Faces і натисніть кнопку Готово.
5. У результаті створюється шаблон для всього застосунку, і відкривається порожня сторінка JSP welcomeJSF.jsp в редакторі вихідного коду. Ви можете переглянути логічні структури нового проекту у вікні проектів Ctrl-1; -1 на Mac. Розгорнути бібліотеки проекту GlassFish вузла. Зверніть увагу, що JSF бібліотеки, такі як jsf-impl.jar включені в GlassFish і додані до класів проекту. Розгорнути конфігураційні файли і зауважити, що IDE створив faces-config.xml файл, який керує поведінкою компонентів JSF в

веб-застосунку. IDE також зареєстрував сервлету Faces в web.xml дескрипторі розгортання. Сервлет Faces знаходиться між сторінками JSP контрольованих рамок JSF.

12.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

12.4 Контрольні запитання

1. Де розташовується посилання на ваш стиль JSP-сторінки?
2. Які можливості ви отримуєте в результаті створення шаблону для всього застосунку?
3. Для чого потрібна greeting.jsp сторінка?
4. Для чого необхідна success.jsp сторінка?
5. Як створити вітальну сторінку?
6. Як відбувається настройка і запуск застосунків?
7. Що необхідно зробити для того щоб додати широкі функціональні можливості веб-застосунків JSF?
8. Як підключити резервний Bean?
9. Для чого необхідна настройка Page Navigation?
10. Що являє собою JSF?

12.5 Перелік джерел

1. Глушаков С.В., Коваль А.В., Черепнин С.А. Программирование на Visual C++ .-Харьков:Фолио,2002 .-726 с.-Учебный курс .-966-03-1776-X Шифр: 681.3(075.8) Авторський знак: Г55
2. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ЛАБОРАТОРНА РОБОТА 13: WEB-СЛУЖБИ НА ПЛАТФОРМІ JAVA EE.

Мета роботи: Познайомитися з технологіями розробки веб-сервісів в Java EE.

13.1 Теоретичні відомості

ап

13.2 Порядок виконання

Необхідно розробити веб-сервіс, що надає користувачам інформацію про погодні умови в містах світу. Інформація про погоду зберігається в БД і включає в себе три характеристики:

- температура;
- хмарність;
- опади.

Користувачеві-клієнту надається веб-інтерфейс де він має можливість вибрати місто і отримати інформацію про погоду в цьому місті.

Таким чином, архітектура системи на логічному рівні складається з чотирьох шарів, де кожний наступний шар є клієнтом попереднього. У дужках вказані відповідне ПЗ для розміщення і підтримки задачі:

1. База даних (Derby DB).
2. Веб-сервіс JAX-WS (JBoss WS, вбудований в JBoss AS).
3. Веб-застосунок, що складається з JSP-сторінок і сервлетів (Tomcat,

вбудований в JBoss AS).

4. Робоче місце клієнта (браузер).

Фізично кожен шар може розміщуватися на окремому комп'ютері і взаємодіяти з іншими шарами по мережі. У нашому випадку всі чотири шари будуть виконуватися на одному комп'ютері.

Для вирішення поставленого завдання необхідно виконати наступні кроки:

1. Створити новий проект для веб-сервісу.
2. Створити таблицю weather та заповнити її даними про міста і погодні умови.
3. Створити Java-клас WeatherInfo для представлення та даних про погоду.
4. Розробити веб-сервіс WeatherWS, що містить метод getWeatherInfo(), що приймає як параметр назву міста та повертає дані у вигляді об'єкта WeatherInfo. Також сервіс повинен містити метод getCities () для отримання списку міст.
5. Розгорнут и веб-сервіс на сервері застосунків.
6. Створити новий проект для веб-клієнта.
7. Використо вуючи WSDL-опис розгорнутого веб-сервісу згенерувати класи-заглушки для доступу до веб-сервісу.
8. Розробити клас-фільтр, що забезпечує отримання від веб-сервісу списку

міст та передачу його JSP-сторінці.

9. Розробити JSP сторінку для введення даних і відображення результатів.
10. Розробити сервлет, що забезпечує виклик веб-сервісу для отримання інформації про погоду по обраному місту.
11. Упакувати веб-застосунок і розгорнути на сервері.
12. Протестувати роботу застосунка в браузері.

13.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

13.4 Контрольні запитання

1. Що таке веб-сервіс?
2. Що таке WSDL?
3. Розробка веб-сервісів в Java EE.

13.5 Перелік джерел

1. Янг Майкл Дж. Visual C++6.Полное руководство. Т.2 .- К.:Ирина,1999 .-560 Шифр: 681.3.06 Авторський знак: Я60

2. Heffelfinger D. Java EE 6 with GlassFish 3 Application Server / D. Heffelfinger – Packt Publishing.
3. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ЛАБОРАТОРНА РОБОТА 14: РОЗРОБЛЕННЯ RICH INTERNET APPLICATIONS.

Мета роботи: Познайомитися з технологіями розробки Rich Internet Application.

14.1 Теоретичні відомості

Якими характеристиками повинно володіти якісне програмне забезпечення? Чи існують інструменти, що допомагають програмістові оптимізувати процес розробки? Ні для кого не секрет, що більшість розробників світу віддають свою перевагу технології Java, так як саме вона дозволяє створювати крос-платформові, надійні і розширювані застосунки.

Java активно розвивається вже протягом сімнадцяти років, і самим одним з істотних останніх нововведень є створення додаткової платформи JavaFX, що стала серйозним конкурентом для фактичного лідера галузі розробки Rich Internet Application (далі - RIA) Adobe Flex і технології Microsoft Silverlight , що активно розвивається.

JavaFX дозволяє створювати застосунки для роботи з мультимедійним контентом, графічні інтерфейси користувача для бізнес-застосунків, ігри для персональних комп'ютерів і мобільних пристроїв, насичені графікою, мультимедіа веб-сайти та ін Крім того, суттєвою перевагою даної платформи є повна інтеграція з JavaAPI. Це означає, що більшість програмістів, приступаючи до освоєння нової області, віддадуть перевагу максимально звичному і знайомоиу інструменту. І в такому випадку на стороні JavaFX виявиться «армія» діючих Java - розробників.

Так як реліз технології відбувся відносно недавно, а за минулий рік концепція платформи істотно змінилася, на сьогоднішній день не існує допоміжного інструменту, який міг би спростити розробку застосунків на

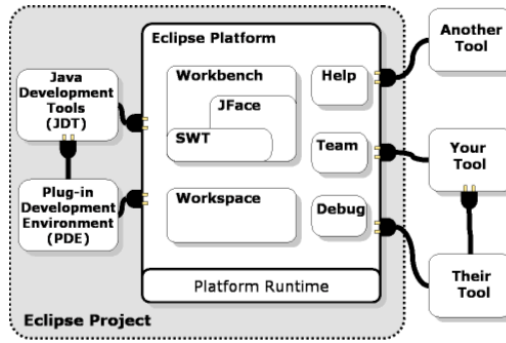
базі JavaFX 2.x, тим самим вирішивши наступні проблеми:

1. Відсутність інструменту, який підтримує розробку застосунків у повному життєвому циклі (побудова \ компіляція \ відладка \ запуск).
2. Низьку продуктивність розробника через відсутність засобів для швидкого створення графічного інтерфейсу користувача (далі - GUI).
3. Відсутність зручного інструменту для огляду всього спектру наявних універсальних GUI - об'єктів платформи, що вимагає від новачка витрачати додатковий час для пошуку існуючих рішень у документації.

В даній роботі буде розглянута розробка плагіна для однієї з найпопулярніших інтегрованих середовищ розробки (далі - IDE) Eclipse.

Eclipse є платформою з відкритим вихідним кодом і з продуманою, добре спроектованою і розширюваною архітектурою, що дозволяє будь-якому бажаючому створювати інструментальні засоби, які легко інтегруються в це середовище.

В Eclipse входять три підпроекти, що розробляються більш-менш незалежно один від одного - Platform, JDT (Java development tools) і PDE (Plug-in development environment). Platform надає базові сервіси, JDT дозволяє розробляти застосунки Java, а PDE - нові компоненти (плагіни) Eclipse.



Малюнок .1 Архітектура Eclipse

Таким чином, виходячи з існуючих проблем і спираючись на вище описані вимоги, можна відзначити кінцеву мету лабораторної роботи: спроектувати інструмент, який спростить розробку RIA на платформі JavaFX 2.x.

14.2 Порядок виконання

Написати довільний RIA-застосунок для здійснення колективного онлайн-спілкування між його розподіленими в мережі учасниками, які ідентифікуються псевдонімами. Реалізувати сервісні сповіщення про учасників, які заходять на сервер чи залишають його.

14.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

14.4 Контрольні запитання

1. Що таке RIA (Rich Internet Application)?
2. Що таке фронтенд?
3. Що таке бекенд?
4. Що таке MVC?

14.5 Перелік джерел

1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. JSR 224: The Java API for XML-Based Web Services (JAX-WS) 2.2 Rev a. Maintenance Release. Oracle. – May 13, 2011. – 181 p. [Electronic resource]. – Access mode : http://jcp.org/en/jsr/detail?id=224.

ЛАБОРАТОРНА РОБОТА 15: ОГЛЯД СУЧАСНИХ JAVA-ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ БАГАТОЛАНКОВИХ ЗАСТОСУНКІВ.

Мета роботи: Ознайомитись із поняттям багатоланковості. Навчитися розробляти багатоланкові застосунки.

15.1 Теоретичні відомості

ап

15.2 Порядок виконання

Для виконання цієї роботи необхідні наступні речі:

1) Eclipse IDE 3.4+ з застосунками Spring IDE, m2Eclipse (деталі установки можна знайти в Spring MVC туторіалі).
2) Maven 2+
3) Java 5+
4) MySQL 5+

Ці інструменти повинні бути налаштовані та робочі.

Почнемо з створення проекту. В Eclipse слід вибрати File->New->Other->Maven Project.

Create a simple project повинен бути не відмічений. При потребі можна вказати розташування проекту, відчекнувши Use default Workspace location.

На наступному діалозі слід вибрати архетип maven-archetype-quickstart. Після вибору архетипу необхідно вказати ідентифікатор групи: com.rozrobka та артефакту: hiberdemo.

Коли все вказано, Eclipse створить проект. Для того щоб підключити Hibernate та Spring до pom.xml файлу, в корені проекту, слід додати

наступний текст у групу:

```
  <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring</artifactId>
        <version>2.5.6</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate</artifactId>
        <version>3.2.6.ga</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-annotations</artifactId>
        <version>3.3.0.ga</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-commons-
annotations</artifactId>
        <version>3.3.0.ga</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
    </dependency>
    <dependency>
        <groupId>commons-dbcp</groupId>
```

```
        <artifactId>commons-dbcp</artifactId>
        <version>1.2.2</version>
    </dependency>
```

Це вкаже Maven-у використовувати Hibernate, MySQL бібліотеки та Spring як залежності для проекту.

Можна приступати до створення класів. Моделі будуть розміщуватись у пакеті *com.rozrobka.model*. Для зручності створимо маркер інтерфейс DataEntity, який будуть реалізовувати класи моделі.

Предметна область у цьому прикладі доволі проста — Оголошення (ad) яке належить до Категорії (category) у співвідношенні багато-до-одного (many-to-one).

Створимо першу модель Category:

```
package com.rozrobka.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="categories")
public class Category implements DataEntity {

        @Id
        @GeneratedValue
        @Column(name="id")
        private Integer id;
```

```
        @Column(name="name")
        private String name;

        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

Ця модель використовує анотації з пакетів JPA. Hibernate надає реалізації цих анотації, які сумісні з специфікаціями JPA.

Анотація @Entity вказує що цей клас є сутністю. Анонтація @Table дозволяє керувати налаштуваннями мапування сутності на таблицю. В нашому випадку ми вказуємо що сутності Category будуть зберігатись у таблиці caregories.

Далі ідуть анотації рівня полів. @Id вказує що поле є ключовим. @GeneratedValue вказує що значення поля генерується базою даних, наприклад авто-інкремент. Анотація @Column допомагає вказати характеристики поля бази даних, наприклад назву, довжину, можливість пустих значень та інше.

Створимо наступну модель Ad:

```
package com.rozrobka.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name="ads")
public class Ad implements DataEntity {

        @Id
        @GeneratedValue
        @Column(name="id")
        private Integer id;

        @Column(name="name", nullable=false)
        private String name;

        @Column(name="description")
        private String description;

        @Column(name="email")
        private String email;

        @JoinColumn(name = "category_id")
        @ManyToOne(optional = false)
```

```
        private Category category;

        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }

        public String getDescription() {
                return description;
        }
        public void setDescription(String description)
{
                this.description = description;
        }
        public String getEmail() {
                return email;
        }
        public void setEmail(String email) {
                this.email = email;
        }
        public Category getCategory() {
```

```
                return category;
        }
        public void setCategory(Category category) {
                this.category = category;
        }
}
```

Цей клас представляє оголошення і має звязок з класом Category. Використані подібні анотації як і для Category.
Крім стандартних, для мапування використана анотація @JoinColumn, яка вказує поле яке відповідає за звязок на рівні бази даних, та анотація @ManyToOne, яка вказує мапування на логічному рівні класів.

Обидва класи моделі мають відповідати специфікації JavaBeans, тобто мати правильно проіменовані гетери та сетери та інші речі.
Моделі готові, можна приступити до налаштування Spring.
Для цього слід створити папку resources у папці src/main. У ній слід створити файл *appCtx.xml* з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:util="http://www.springframework.org/schema/util
" xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd

http://www.springframework.org/schema/util
```

```
http://www.springframework.org/schema/util/spring-
util-2.5.xsd">

        <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.A
nnotationSessionFactoryBean">
                <property name="dataSource"
ref="dataSource"/>
                <property name="configLocation">

<value>classpath:/hibernate.cfg.xml</value>
                </property>
                <property name="configurationClass"
value="org.hibernate.cfg.AnnotationConfiguration"/>
                <property name="hibernateProperties"
ref="hibernatePropertiesConfigurer"/>
        </bean>

        <bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource">
                <property name="driverClassName"
value="${database.driverClass}"/>
                <property name="url" value="$
{database.url}"/>
                <property name="username" value="$
{database.username}"/>
                <property name="password" value="$
{database.password}"/>
        </bean>
```

```
        <util:properties
id="hibernatePropertiesConfigurer"
location="classpath:/hibernate.properties"/>

        <bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.Proper
tyPlaceholderConfigurer">
                <property name="locations">
                        <list>

<value>classpath:/database.properties</value>
                        </list>
                </property>
        </bean>

        <bean id="adsDao"
class="com.rozrobka.hiberdemo.AdsDao">
                <property name="sessionFactory"
ref="sessionFactory"/>
        </bean>
</beans>
```

Це є контекст файл Spring, у якому описуються класи та звязки між ними. Давайте пройдемось по головним бінам.

sessionFactory описує допоміжний Spring клас AnnotationSessionFactoryBean який дає можливість використовувати анотовані моделі. Йому вказуються файли налаштувань та джерело даних.

dataSource визначає джерело даних. Параметри підключення автоматично вказуються Spring-ом на етапі запуску.

hibernatePropertiesConfigurer та propertyConfigurer є допоміжними

бінами які допомагають вказати параметри налаштувань Hibernate та джерела даних.

adsDao це є наш бін, який буде містити логіку доступу до бази даних. Його ми зараз створимо.

Для цього у пакеті *com.rozrobka.hiberdemo* слід створити клас AdsDao, який розширяє клас HibernateDaoSupport:

```
package com.rozrobka.hiberdemo;

import java.util.List;

import
org.springframework.orm.hibernate3.support.HibernateDa
oSupport;

import com.rozrobka.model.Ad;
import com.rozrobka.model.DataEntity;

public class AdsDao extends HibernateDaoSupport {

        public void save(DataEntity entity) {
                getHibernateTemplate().save(entity);
        }

        public List<Ad> getAllAds() {
                return
getHibernateTemplate().loadAll(Ad.class);
        }

}
```

Клас містить два методи: save та getAllAds. За допомогою save можна зберігати всі класи, які реалізують інтерфейс DataEntity, у нашому випадку це Ad та Category.

Метод getAllAds вертає з бази даних всі сутності типу Ad. метод getHibernateTemplate надається суперкласом HibernateDaoSupport і є зручним та практичним шляхом взаємодії з Hibernate.

Реалізація готова. Тепер слід доналаштувати Hibernate та параметри підключення до бази даних.

Для цього створимо у папці resources файл *hibernate.cfg.xml* з наступним вмістом:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                <mapping class="com.rozrobka.model.Ad"
/>
                <mapping
class="com.rozrobka.model.Category" />
        </session-factory>
</hibernate-configuration>
```

Це є конфігураційний файл Hibernate. У нашому випадку він є малим, так як ми винесли налаштування підєднання до бази даних окремо. Він описує два класи, які є сутностями Hibernate.

Для опису необхідно вказувати канонічну назву класу з назвою пакету.

Далі слід створити додатковий конфігураційний файл *hibernate.properties*, який міститиме налаштування, які можуть мінятись з часом:

```
hibernate.show_sql = true
hibernate.dialect=org.hibernate.dialect.MySQLDialect
# or validate
hibernate.hbm2ddl.auto = update
```

Цей файл містить три налаштування.

show_sql вказує чи виводити до логу запити які робить Hibernate. dialect вказує який використовувати діалект SQL. у нашому випадку це діалект MySQL. Для інших типів баз даних слід вибрати інший діалект. Діалект має вплив на згенерований SQL код.

hbm2ddl.auto вказує поведінку при роботі з структурою бази даних. update означає що Hibernate сам згенерує структуру на базі мапувань, або анотації. validate означає, що Hibernate просто перевірить структуру на відповідність моделям.

Далі створимо конфігураційний файл бази даних database.properties:

```
database.driverClass=com.mysql.jdbc.Driver
database.url=jdbc:mysql://localhost/hiberdemo
database.username=user
database.password=password
```

Приступимо до самої аплікації.

Слід змінити вміст класу App з пакету com.rozrobka.hiberdemo на

наступний:

```
package com.rozrobka.hiberdemo;

import java.util.List;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplic
ationContext;

import com.rozrobka.model.Ad;
import com.rozrobka.model.Category;

/**
 * Hello world!
 *
 */
public class App {
    public static void main( String[] args ) {
        String[] configs = {"classpath:/appCtx.xml"};
        ApplicationContext ctx = new
ClassPathXmlApplicationContext(configs);

        AdsDao adsDao = (AdsDao)
ctx.getBean("adsDao");

        Category cat1 = new Category();
        cat1.setName("First Category");
```

```
        adsDao.save(cat1);

        Ad ad1 = new Ad();
        ad1.setCategory(cat1);
        ad1.setName("Ad 1");
        ad1.setEmail("mail@inet.com");
        ad1.setDescription("description");

        Ad ad2 = new Ad();
        ad2.setCategory(cat1);
        ad2.setName("Ad 2");
        ad2.setEmail("mail@inet.com");
        ad2.setDescription("description");

        adsDao.save(ad1);
        adsDao.save(ad2);

        List<Ad> ads = adsDao.getAllAds();
        System.out.println("Array size: " +
ads.size());

    }
}
```

Давайте розберемо цей код.

```
        String[] configs = {"classpath:/appCtx.xml"};
        ApplicationContext ctx = new
ClassPathXmlApplicationContext(configs);
```

```
        AdsDao adsDao = (AdsDao)
ctx.getBean("adsDao");
```

Вказування розташування файлу контексту, та ініціалізація Spring контексту. ctx.getBean вертає бін по імені з ініціалізованого контексту. Повернений бін це є наше DAO яке пряцює з Hibernate.
Решта коду взаємодіять з цим DAO: створення та збереження однієї категорії, та створення двох оголошень, визначення звязку з категорією та подальше збереження.

Останні дві стрічки роблять запит до бази даних та перевіряють чи дійсно два оголошення збереглись.

Перед запуском слід створити саму базу даних:

```
create database hiberdemo;
```

Коли все готово, запускаємо клас App!

Має бути багато логів на виході, які закінчаться приблизно такими стрічками:

```
Hibernate: insert into categories (name) values (?)
Hibernate: insert into ads (CLIENT_ID, description,
email, name) values (?, ?, ?, ?)
Hibernate: insert into ads (CLIENT_ID, description,
email, name) values (?, ?, ?, ?)
Hibernate: select this_.id as id0_1_, this_.CLIENT_ID
as CLIENT5_0_1_, this_.description as descript2_0_1_,
this_.email as email0_1_, this_.name as name0_1_,
category2_.id as id1_0_, category2_.name as name1_0_
from ads this_ inner join categories category2_ on
this_.CLIENT_ID=category2_.id
```

```
Array size: 2
```

Слід памятати що після створення схеми, слід змінити конфігурацію:

```
hibernate.hbm2ddl.auto=validate
```

15.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

15.4 Контрольні запитання

1. Що таке Hibernate?
2. З яких компонентів складається архітектура Hibernate?
3. З яких компонентів складається архітектура застосунків Hibernate?
4. З яких компонентів складається Hibernate?
5. Що таке Spring Framework? Ki можливості надає розробнику Spring MVC?
6. Які особливості надає Spring для фреймворків?

15.5 Перелік джерел

1. JSR 245: Expression Language Specification, Version 2.2. Maintenance Release. A component of the JavaServer™ Pages

Specification. Version 2.2. – December 10, 2009. 594 p. [Electronic resource]. – Access mode : http://jcp.org/en/jsr/detail?id=245.

2. JSR 314: JavaServer™ Faces Specification, Version 2.1. Maintenance Release 2. Oracle. – November 8, 2010. – 468 p. [Electronic resource]. – Access mode : http://jcp.org/en/jsr/detail?id=314.
3. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012

ДОДАТКОВА ЛІТЕРАТУРА

1. Грайворонський М.В. Безпека інформаційно-комунікаційних систем / М.В. Грайворонський, О.М. Новіков. – К.: Видавнича група BHV, 2009. – 608 с.
2. Олифер В.Г. Сетевые операционные системы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2001. – 544 с.
3. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы / В.Г. Олифер, Н.А. Олифер. – СПб: Издательство «Питер», 2000. – 672 с.
4. Соломон Д. Внутреннее устройство Microsoft Windows 2000 / Д. Соломон, М. Руссинович. – М: Русская Редакция, 2001. – 752 с.
5. Столингс В. Криптография и защита сетей. Принципы и практика, 2-е изд. : Пер. с англ. / В. Столингс. – М. : Издательский дом “Вильямс”, 2001. – 672 с.
6. Медведовский И.Д. Леонов Д. Г. Атака на Internet. – 2-е изд. / И.Д. Медведовский, П.В. Семьянов. – М.: ДМК, 1999. – 336 с.
7. Лукацкий А. Обнаружение атак / А. Лукацкий. – СПб.: БХВ-Петербург, 2001. – 624 с.
8. Анин Б. Защита компьютерной информации / Б. Анин. – СПб.: БХВ-Петербург, 2000. – 384 с.
9. Белкин П.Ю. Программно-аппаратные средства обеспечения информационной безопасности. Защита программ и данных / П.Ю. Белкин, О.О. Михальский, А. С. Першаков и др. – М.: Радио и связь, 2000. – 168 с.
10. Богуш В.М. Інформаційна безпека від А до Я / В.М. Богуш, А.М. Кудін. – К.: МОУ, 1999. – 456 с.
11. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C (Second Edition) / B. Schneier. – N.Y.: John Wiley & Sons, Inc., 1996.

– 758 р.

12. Касперски К. Техника сетевых атак / К. Касперски. – М.: «СОЛОН-Р», 2001. – 396 с.
13. Касперски К. Техника и философия хакерских атак / К. Касперски. – М.: «СОЛОН-Р», 2001. – 272 с.
14. Касперски К. Фундаментальные основы хакерства / К. Касперски. – М: «СОЛОН-Р», 2005. – 448 с.
15. Шеховцов В.А. Операційні системи / В.А. Шеховцов. – К: Видавнича група BHV, 2005. – 576 с.
16. Немет Э. UNIX: руководство системного администратора / Э. Немет, Г. Снайдер, С. Сибасс, Т.Р. Хейн. – К.: BHV, 1998 – 832 с.
17. Seth T.R. UNIX System Security Tools / T.R. Seth. – N.Y.: McGraw-Hill, 2000. – 444 p.
18. Кастер Х. Основы Windows® NT и NTFS / Х. Кастер. – М.: Издательский отдел «Русская Редакция» ТОО «Channel Trading Ltd.», 1996. – 440 с.

НАВЧАЛЬНЕ ВИДАННЯ

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни «Крос-платформне програмування»

для студентів напряму підготовки 6.050101 – Комп'ютерні науки.

Укладачі: Руслан Орестович Козак,

Тарас Володимирович Михайлович.

Підписано до друку _____ Формат 60x84 1/16. Ум. др. арк. 7.

Друк лазерний. Замовлення № _____. Наклад 100 пр.

Віддруковано у видавництві ТНТУ.