МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

# КУРСОВА РОБОТА

з дисципліни: «Інженерія програмного забезпечення»

на тему: «Система програмно-цільового підходу. Функціональність для експерта»

Студента 2-го курсу групи ІО-22

Напряму підготовки: 6.050102 Комп’ютерна інженерія

Спеціальності: 7.8.05010201 Комп’ютерні системи та мережі

Сагача Ярослава Ігоровича

Керівник к.т.н., доцент Абу Усбах О. Н.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

Київ 2014

**АНОТАЦІЯ**

Сьогодні жоден експеримент не відбувається без попереднього його планування. Тому й жодна поставлена задача не може бути вирішена без детальної оцінки можливостей , ресурсів та йнших факторів. При чому, це може бути задача уряду, будівельної фірми, компанії ,що розробляє програмне забезпечення тощо. У цій курсовій роботі описана система, яка дозволяє впорядкувати процес аналізу задачі та оцінки можливостей її вирішення. Робота описує функціонал експерта. Експерт – людина, яка має повний доступ до бази ресурсів, володіє усією інформаціею про поставлену задачу. Його обов'язком є детальний аналіз задачі та складання оцінки щодо її вирішення(можливість вирішення, кількість ресурсів що необхідні для вирішення та ін.).

Курсова робота містить такі розділи: огляд MVC, проектування програмного додатку, розробка програмного додатку, тестування програмного забезпечення. Розділ “огляд MVC” містить опис та аналіз «Model-View-Controller». Розділ проектування містить приблизний макет програми , описує її структуру, UML діаграмми. Розробка програмного додатку містить те, що безпосередньо належить до написання коду програми. Розділ “тестування програмного додатку” описує детальну перевірку програми на працездатність.

**ЗМІСТ**

ВСТУП………………………………………………………………………………..3

1.ОГЛЯД MODEL-VIEW-CONTROLLER…………….……………………......…4

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ…………………………..…..10

2.1 ПРЕЦЕДЕНТИ…………………………………………………………….12

2.1.1 ПОЧАТКОВИЙ ОПИС ПОТОКУ ПОДІЙ РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ………………..………………………………….…13

2.1.2 РОЗРОБКА ОРІЄНТИРІВ……….…………………………………. .14

2.1.3 РОЗРОБКА АТРИБУТІВ………………….…………………….. .…15

2.1.4 РОЗРОБКА ІНТЕНСИВНОСТІ ВИКОРИСТАННЯ………….…....16

2.1.5 РЕЗУЛЬТУЮЧИЙ ОПИС ПОТОКУ ПОДІЙ ПРОГРАМИ..…...…17

2.1.6 ДОПОВНЕННЯ ОПИСУ ІЛЮСТРОВАНОГО СЦЕНАРІЮ ОБ’ЄКТАМИ ДІАГРАМИ ……………………………………………………..….18

2.2 ДІАГРАМА СУТНОСТЕЙ………………………………………………19

2.3 ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА…………………………..20

2.4 ПРОЕКТУВАННЯ СТРУКТУРИ………………………………………..21

3.РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ……….………………….…............ ..22

3.1 РОЗРОБКА СТРУКТУРИ…………………………………………….…..24

3.2 ІНСТРУКЦІЇ ДЛЯ КОРИСТУВАЧА………………………………….....26

3.3 СПЕЦИФІКАЦІЇ КЛАСІВ……………………………………….…….....28

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.……………………….…32

ВИСНОВКИ…………….…………………………………………....………………40

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……………………......………………….41

ДОДАТКИ………………………………………………………...............................42

**ВСТУП**

Метою курсової роботи є закріплення теоритичних знань і отримання практичних навичок з проектування, моделювання, розробки та тестування програмного забезпечення, у використанні шаблонів проектування. Також отримання навичок у написанні програмного забезпечення, що підтримує зв’язок з базою даних, матиме механізми конфігурації та зберігання файлів незалежно від платформи.

**1.ОГЛЯД MODEL-VIEV-CONTROLLER**

Архітектурний шаблон програмного забезпечення – це шаблон програмного забезпечення, що вирішує архітектурні проблеми розробки програмного забезпечення. Архітектурні шаблони виражають фундаментальну схему структурної організації певної програмної системи. Програмна схема складається із визначених наперед підсистем, а також точно визначає їхні сфери відповідальності та взаємовідносини.

Порівняно з шаблонами проектування, архітектурні шаблони мають ширший масштаб. Основними елементами структури архітектурних шаблонів є не класи й об’єкти, а умовні частини програмної системи.

Model-View-Controller – це один з архітектурних шаблонів , який використувується під час проектування та розробки програмного забезпечення.

Даний шаблон поділяє систему на три частини: *модель* даних, *вигляд* даних та керування.

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

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

Наприклад, можна оновити *вигляд* або стиль GUI (графічного інтерфейсу програми) без необхідності зміни моделі даних або *контролер*у.

Дана схема проектування часто використовується для побудови архітектурного каркаса, коли переходять від теорії до реалізації в конкретній предметній області.

**Історія.** Вперше концепція MVC була описана в 1979 році Трюгве Реєнскаугом, який працював тоді над мовою програмування Smalltalk в Xerox PARC. Оригінальна реалізація була описана у статті «Applications Programming in Smalltalk-80: How to use Model-View-Controller».

У оригінальній концепції була описана сама ідея і роль кожного з елементів моделі, *вигляд*у та керування. Але зв’язки між ними були описані без конкретизації.

Розрізняли дві основні модифікації MVC:

1. Пасивна *модель* – *модель*, що не має ніяких способів діяти на представлення або керування, і використовується ними в якості джерела даних для відображення. Всі зміни моделі відслідковуються *контролер*ом. Він також відповідає за перетворення *вигляду*, якщо це необхідно. Дана *модель* найчастіше використовується у структурному програмуванні, так як у цьому випадку *модель* – просто структура даних, без методів та обробників.
2. Активна *модель* – *модель*, яка сповіщує *вигляд* про те, що в ній пройшли зміни, а *вигляди*, які зацікавленні в сповіщеннях, підписуються на ці повідомлення. Це дозволяє зберегти незалежність моделі як від керування, так і від *вигляду*.

**Призначення.** Основною метою застосування цієї концепції полягає в розділенні бізнес-логіки (моделі) від її візуалізації(представлення, *вигляду*). За рахунок такого поділу з’являється можливість повторного користування. Тобто система складається з декількох частин, написаних раніше компонентів або частин інших систем, і ці компоненти можуть застосовуватись більше ніж один раз. Повторне користування – основна методологія, яка застосовується для скорочення трудових витрат при розробці складних систем.

Найбільш корисне є застосування цієї концепції в тих випадках, коли користувач повинен бачити одні і ті ж самі дані одночасно у різноманітних контекстах або з різними точками зору.

Зокрема, виконуються наступні завдання :

1. До однієї моделі можна приєднати декілька *виглядів*, при цьому не змінюючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у *вигляді* електронної таблиці, гістограми і кругової діаграми.
2. Не порушуючи реалізацію *вигляді*, можна змінити реакції на дії користувача (натиснення клавіші, введення даних), для цього достатньо використовувати інший *контролер*.
3. Ряд розробників спеціалізується лише в одній з областей : або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можливо, що програмісти, які займаються розробкою бізнес-логіки (моделі), зовсім не будуть знати, який *вигляд* буде використовуватись.

**Компоненти.** Як відомо, архітектурний шаблон MVC поділяє програму на три частини. Тоді відповідальність за різні функції розподіляється між різними компонентами.

1. ***Модель*.** *Модель* даних – абстрактне представлення реального світу, що відображає тільки ті об’єкти, що безпосередньо стосуються програми. Це, як правило визначає специфічну групу об'єктів, їх атрибутивне значення і відношення між ними.

*Модель* надає знання: дані і методи роботи з цими даними, реагує на запити, змінюючи свій стан. Не містить інформації, як ці знання можна візуалізувати.

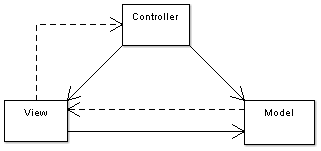
1. ***Вигляд*.** Відповідає за відображення інформації (візуалізацію). Часто в якості *вигляд*у виступає форма(вікно) з графічними елементами.
2. ***Контролер*.** Забезпечує зв’язок між користувачем і системою: контролює введення даних користувачем і використовує *модель* і представлення для реалізації необхідної реакції.

Рис.1.1 Діаграма взаємодії між компонентами шаблону

MVC не описує взаємодію моделі з даними – рівень доступу до даних розглядають інші шаблони.

Отже, можна зробити такі висновки. *Модель* повинна мати зв’язок зі сховищем даних (База даних, файли). Обробка даних відбувається у *контролер*і. А їх представлення забезпечує *вигляд*.

Важливо, що як *вигляд*, так і *контролер* залежать від моделі. Проте *модель* не залежить ні від *вигляд*у, ні від *контролер*а. Тим самим досягається призначення такого розподілу. Це дозволяє будувати *модель* незалежно від візуального представлення, а також створювати декілька різноманітних представлень для однієї моделі.

Хоча *модель* не повинна мати обмежень, вона може бути обмежена типом даних, який вона представляє, і котрий наслідується від суперкласу для усіх можливих типів - класа Object. Тип даних, яким оперує *модель* – властивість моделі.

Для реалізації Model-View-Controller використовується велика кількість шаблонів проектування (в залежності від складності архітектурного вирішення). Основними шаблонами є Observer, Strategy та Composite.

Основною реалізацією є відокремлення *вигляд*у від моделі шляхом встановлення між ними протоколу взаємодії, використовуючи апарат подій (підписка/сповіщення). При кожній зміні внутрішніх даних у моделі вона сповіщує усі представлення, які від неї залежать. Ті ж у свою чергу оновлюються. Для виконання таких операцій використовується шаблон проектування Observer. При обробці реакції користувача *вигляд* обирає, залежно від потрібної реакції, потрібний *контролер*, який забезпечить той чи інший зв’язок з моделлю. Для цього використовується шаблон Strategy (також, при певній модифікації, можна використовувати шаблон Command). Для можливості однотипного звернення з підоб’єктами складного ієрархічного *вигляд*у може використовуватись шаблон Composite. Крім того, також можуть використовуватись інші шаблони проектування, наприклад, Factory Method, який дозволить задати за замовчуванням тип *контролер*у для відповідного *вигляд*у.

MVC задає не стільки правила поділу програми на окремі компоненти, скільки правила їх взаємодії.

Отже, вибір та використання шаблонів проектування, під час реалізації архітектурного шаблону MVC, повністю залежить від розробника.

Тому в різноманітних реалізаціях цього шаблону є певні відступи від оригінального. По-перше, відсутній чіткий поділ між *вигляд*ом та *контролер*ом. По-друге, деякі реалізації не потребують виділення бізнес-логіки програми в окремий компонент – *модель*. Програміст самостійно приймає рішення : витрачати зусилля на виділення в програмі моделі або ж відразу описувати у *контролер*і обробку даних, наприклад, звернення до бази даних. Проте таке рішення призведе до проблеми у проектуванні, для боротьби з яким і був створений шаблон MVC: якщо не виділити окрему сутність – *модель*, то уникнути залежності бізнес-логіки від *контролеру* неможливо. Даний вибір реалізації частіше всього помилково обирають неосвідчені програмісти. Хоча певні відхилення від оригіналу не є порушенням основної ідеї шаблону.

Жодний із шаблонів не є ідеальним. Model-View-Controller не є виключенням. Тому у ньому є як і переваги, так і недоліки.

Переваги:

- можливість прикріпити декілька *вигляд*ів до моделі, для того щоб забезпечити презентацію даних з різних поглядів;

- додання інших *вигляд*ів не впливає на *модель*;

- повторне використання.

Недоліки:

- складність (вводить нові рівні абстракції, що дещо ускладнює рішення)

- висока вартість частих оновлень.

Так як, для реалізації шаблону MVC використовуються багато різноманітних шаблонів проектуваня, які в свою чергу містять свої плюси та мінуси, то їх переваги і недоліками, також є перевагами і недоліками MVC.

Існує багато варіацій MVC. Наприклад, шаблон Document-View розпізнає усі три ролі *модель*, *вигляд*, *контролер*, але об'єднує *контролер* з *вигляд*ом.

**Цикл роботи компонентів MVC.** При ініціалізації *вигляд*у користувачем він звертається до моделі і встановлює свій стан. При цьому *вигляд* відправляє команду *контролер*у. *Контролер* інтерпретує команду і змінює *модель*. *Вигляд* регіструє зміну моделі і знову змінюється відповідно до нового стану.

**Висновок.** Так як, він спричинює появу нових рівнів абстракції і дещо ускладнює продуктивність, то найкраще його використовувати для невеликих бізнес-проектів, програм тощо. Для написання дуже великої структурно-ємнісної системи він буде не досить доречним. Найкращою вважається реалізація MVC із активною моделлю. З часів першої реалізації цього шаблону пройшло майже 30 років, проте він до цих пір є стандартом програм та інтернет-додатків.

**2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ**

**2.1 ПРЕЦЕДЕНТИ**

Користувач може відкрити вікно для роботи з задачами, натиснувши кнопку «Задачі» на верхній панелі (Рис.2.1.1).



Рис.2.1.1 Діаграма прецедентів операції відкриття вікна задач

Користувач може переглянути інформацію про задачу, натиснувши кнопку «Відкрити», на бічній правій панелі (Рис.2.1.2).



Рис.2.1.2 Діаграма прецедентів операції відкриття інформації про задачу

Користувач може переглянути детальну інформацію про задачу, натиснувши кнопку «Детальніше», на бічній правій панелі (Рис.2.1.3).



Рис.2.1.3 Діаграма прецедентів операції перегляду детальнішої інформації про задачу

Користувач може зберегти інформацію про задачу, натиснувши кнопку «Зберегти», на бічній правій панелі (Рис.2.1.4)



Рис.2.1.4 Діаграма прецедентів операції збереження інформації про задачу

Користувач може відкрити вікно для роботи з ресурсами, натиснувши кнопку «Ресурси» на верхній панелі (Рис.2.1.5).



Рис.2.1.5 Діаграма прецедентів операції відкриття вікна ресурсів

Користувач може переглянути інформацію про ресурс, натиснувши кнопку «Відкрити», на бічній правій панелі (Рис.2.1.6).



Рис.2.1.6 Діаграма прецедентів операції відкриття інформації про ресурс

Користувач може додати ресурс до бази, натиснувши кнопку «Додати», на бічній правій панелі (Рис.2.1.7)



Рис.2.1.7 Діаграма прецедентів операції додавання ресурса до бази

Користувач може видалити ресурс із бази, натиснувши кнопку «Видалити», на бічній правій панелі (Рис.2.1.8).



Рис.2.1.8 Діаграма прецедентів операції видалення ресурса з бази

Користувач може зберегти інформацію про ресурс, натиснувши кнопку «Зберегти», на бічній правій панелі (Рис.2.1.9)



Рис.2.1.9 Діаграма прецедентів операції збереження інформації про ресурс

Користувач може відкрити вікно для роботи з тестами, натиснувши кнопку «Тести » на верхній панелі (Рис.2.1.10).



Рис.2.1.10 Діаграма прецедентів операції відкриття вікна тестів

Користувач може відкрити тест, натиснувши кнопку «Відкрити» на бічній правій панелі (Рис.2.1.11)



Рис.2.1.11 Діаграма прецедентів операції відкриття тесту

Користувач може зберегти результат теста, натиснувши кнопку «Зберегти» на бічній правій панелі (Рис.2.1.12)



Рис.2.1.12 Діаграма прецедентів операції збереження результату теста

2.1.1 ПОЧАТКОВИЙ ОПИС ПОТОКУ ПОДІЙ РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
3. Відкрити інформацію про задачу.
4. Переглянути детальнішу інформацію про задачу.
5. Змінити дані про задачу.
6. Зберегти дані про задачу.
7. Відкрити інформацію про ресурс.
8. Додати ресурс до бази.
9. Видалити ресурс з бази.
10. Змінити дані про ресурс.
11. Зберегти дані про ресурс.
12. Відкрити тест.
13. Перевірити тест.
14. Зберегти результат тесту.

3) Прецедент закінчується, коли користувач завершує роботу з програмою.

2.1.2 РОЗРОБКА ОРІЄНТИРІВ

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
3. Відкрити інформацію про задачу.[Після відкриття стає активною кнопка «Зберегти»]
4. Переглянути детальнішу інформацію про задачу.
5. Змінити дані про задачу.
6. Зберегти дані про задачу.[Користувач повинен підтвердити зміни у задачі]
7. Відкрити інформацію про ресурс.
8. Додати ресурс до бази.
9. Видалити ресурс з бази. [Користувач повинен підтвердити видалення ресурсу]
10. Змінити дані про ресурс.
11. Зберегти дані про ресурс. [Користувач повинен підтвердити зміни ресурсу]
12. Відкрити тест.
13. Перевірити тест.
14. Зберегти результат тесту. [Користувач повинен підтвердити збереження результату тесту]

3) Прецедент закінчується, коли користувач завершує роботу з програмою.

2.1.3 РОЗРОБКА АТРИБУТІВ

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
3. Відкрити інформацію про задачу.
4. Переглянути детальнішу інформацію про задачу.
5. Змінити дані про задачу.
6. Зберегти дані про задачу.
7. Відкрити інформацію про ресурс.
8. Додати ресурс до бази.
9. Видалити ресурс з бази.
10. Змінити дані про ресурс.
11. Зберегти дані про ресурс.
12. Відкрити тест.{Містить приблизно 10 питань}
13. Перевірити тест.
14. Зберегти результат тесту.{Результат – число правильних відповідей}

3) Прецедент закінчується, коли користувач завершує роботу з програмою.

2.1.4 РОЗРОБКА ІНТЕНСИВНОСТІ ВИКОРИСТАННЯ

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
3. Відкрити інформацію про задачу.(Використовується у 75% випадків)
4. Переглянути детальнішу інформацію про задачу. (Використовується у 55% випадків)
5. Змінити дані про задачу. (Використовується у 45% випадків)
6. Зберегти дані про задачу. (Використовується у 45% випадків)
7. Відкрити інформацію про ресурс. (Використовується у 65% випадків)
8. Додати ресурс до бази. (Використовується у 30% випадків)
9. Видалити ресурс з бази. (Використовується у 30% випадків)
10. Змінити дані про ресурс. (Використовується у 70% випадків)
11. Зберегти дані про ресурс. (Використовується у 70% випадків)
12. Відкрити тест. (Використовується у 50% випадків)
13. Перевірити тест. (Використовується у 50% випадків)
14. Зберегти результат тесту. (Використовується у 95% випадків)

3) Прецедент закінчується, коли користувач завершує роботу з програмою.

2.1.5 РЕЗУЛЬТУЮЧИЙ ОПИС ПОТОКУ ПОДІЙ ПРОГРАМИ

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
3. Відкрити інформацію про задачу.(Використовується у 75% випадків); [Після відкриття стає активною кнопка «Зберегти»]
4. Переглянути детальнішу інформацію про задачу. (Використовується у 55% випадків)
5. Змінити дані про задачу. (Використовується у 45% випадків)
6. Зберегти дані про задачу. (Використовується у 45% випадків); [Користувач повинен підтвердити зміни у задачі]
7. Відкрити інформацію про ресурс. (Використовується у 65% випадків)
8. Додати ресурс до бази. (Використовується у 30% випадків)
9. Видалити ресурс з бази. (Використовується у 30% випадків); [Користувач повинен підтвердити видалення ресурсу]
10. Змінити дані про ресурс. (Використовується у 70% випадків)
11. Зберегти дані про ресурс. (Використовується у 70% випадків); [Користувач повинен підтвердити зміни ресурсу ]
12. Відкрити тест. (Використовується у 50% випадків) ;{Містить приблизно 10 питань}
13. Перевірити тест. (Використовується у 50% випадків)
14. Зберегти результат тесту. (Використовується у 95% випадків); {Результат – число правильних відповідей};[Користувач повинен підтвердити збереження результатів тесту]

3) Прецедент закінчується, коли користувач завершує роботу з програмою.

2.1.6 ДОПОВНЕННЯ ОПИСУ ІЛЮСТРОВАНОГО СЦЕНАРІЮ ОБ’ЄКТАМИ ДІАГРАМИ

1. Прецедент починається, коли користувач запускає програму.
2. Користувач може виконати одну з наступних дій:
3. Відкрити інформацію про задачу.(Використовується у 75% випадків) ; [Після відкриття стає активною кнопка «Зберегти»]. Кнопка «Відкрити»
4. Переглянути детальнішу інформацію про задачу. (Використовується у 55% випадків). Кнопка «Детальніше»
5. Змінити дані про задачу. (Використовується у 45% випадків)
6. Зберегти дані про задачу. (Використовується у 45% випадків) [Користувач повинен підтвердити зміни у задачі]. Кнопка «Зберегти»
7. Відкрити інформацію про ресурс. (Використовується у 65% випадків). Кнопка «Відкрити»
8. Додати ресурс до бази. (Використовується у 30% випадків). Кнопка «Додати»
9. Видалити ресурс з бази. (Використовується у 30% випадків) ; [Користувач повинен підтвердити видалення ресурсу]. Кнопка «Видалити»
10. Змінити дані про ресурс. (Використовується у 70% випадків).
11. Зберегти дані про ресурс. (Використовується у 70% випадків) ; [Користувач повинен підтвердити зміни ресурсу ]. Кнопка «Зберегти»
12. Відкрити тест. (Використовується у 50% випадків) ; {Містить приблизно 10 питань}. Кнопка «Відкрити»
13. Перевірити тест. (Використовується у 50% випадків).
14. Зберегти результат тесту. (Використовується у 95% випадків); {Результат – число правильних відповідей};[Користувач повинен підтвердити збереження результатів тесту]. Кнопка «Зберегти»

3) Прецедент закінчується, коли користувач завершує роботу з програмою.

**2.2 ДІАГРАМА CУТНОСТЕЙ**



На рисунку 2.2.1 показана діаграма сутностей програмного додатку:

Рис.2.2.1 Діаграма сутностей програмного додатку

Таблиця 2.3.1 Відповідність сутностей з класами графічного пакету Swing

|  |  |
| --- | --- |
| Сутність | Клас бібліотеки Swing |
| Головне меню | JFrame |
| Кнопка | JButton |
| Верхня панель | JBox |
| Верхнє меню | JMenuBar |
| Кнопка верхнього меню | JMenuBar |
| Панель відображення інформації | JEditorPane |
| Назва | JLabel |

**2.3 ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА**

На рисунку 2.3.1 показаний ескіз моделі графічного інтерфесу програмного додатку. Як приклад було взяте вікно «Задачі». Графічний інтерфейс розроблявся таким чином, щоб користувач міг змогу бачити область даних з якою він працює. Також, щоб усі він швидко й доступно міг обрати потрібну йому операцію. Даний інтерфейс програмного додатку є досить зручним і простим у користуванні, що дозволяє користувачу зосередитись на роботі.



Рис.2.3.1 Ескіз графічного інтерфейсу програмного додатку

Графічний інтерфейс розроблений за допомогою стандартних засобів Java, а саме графічного пакету Swing.

**2.4 ПРОЕКТУВАННЯ СТРУКТУРИ**



Рис.2.4.1 Діаграма класів програмного додатку

3.РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Структура програмного додатку

Діаграма класів програмного додатку створена для розуміння структури програми в цілому. На ній зображена відповідність всіх класів(див. Додаток Б).

3.2 Інструкції для користувача

Для відображення панелі «Задачі» потрібно натиснути кнопку «Задачі» у верхньому меню. Панель «Задачі» з правого боку вікна містить три кнопки: «Показати», «Детальніше», та «Результат». Для відображення задач у таблиці потрібно натиснути кнопку «Показати». Для детальнішого перегляду окремої задачі потрібно натиснути кнопку «Детальніше». Для формування висновку по задачі потрібно натиснути кнопку «Результат».

Для відображення панелі «Ресурси» потрібно натиснути кнопку «Ресурси» у верхньому меню. Панель «Ресурси» з правого боку вікна містить чотири кнопки: «Показати», «Додати», «Змінити», «Видалити». Кнопка «Показати» використовується для відображення ресурсів у таблиці. Кнопка «Додати» використовується для додання ресурсу у базу даних та у таблицю. Кнопка «Змінити» використовується для детального перегляду та зміни обраного ресурсу. Кнопка «Видалити» використовується для видалення ресурсів з бази даних та таблиці.

Для відображення панелі «Тести» потрібно натиснути однойменну кнопку у верхньому меню. Панель «Тести» з правого боку вікна містить дві кнопки: «Відкрити», «Результат». Кнопка «Відкрити» використовується для відкриття певного тесту. Для формування результату тесту потрібно натиснути кнопку «Результат».

При натисканні кнопок з правого боку вікна у будь-якій панелі відкривається окремі вікна. Вони містять поля для зміни чи перегляду, і одну кнопку «ОК» . Після натиснення цієї кнопки зміни набудуть чинності.

Верхнє меню у правому боці містить дві кнопки : «Конфігурація» та «Інформація». При натисненні кнопки «Конфігурація» відкриється вікно з налаштуваннями для додатку. При натисненні кнопки «Інформація» відкриється вікно з інформацією про додаток.

* 1. Специфікації класів

SRC

Автор : Сагач Ярослав

⁃⁃⁃ Package app.model

‣‣ public class Resourсе extends java.lang.Object

Клас що відповідає за представлення ресурсу з бази даних

Конструктори:

• Resource

public Resource()

Конструктор без параметрів

• Resource

public Resource(java.lang.String type,

int number,

java.lang.String unit,

java.lang.String comment)

Parameters:

type - тип ресурсу

number - кількість

unit - одиниці вимірювання

comment - коментар

• Resource

public Resource(int id,

java.lang.String type,

int number,

java.lang.String unit,

java.lang.String comment)

Parameters:

id - ідентифікатор

type - тип ресурсу

number - кількість

unit - одиниці вимірювання

comment – коментар

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор ресурсу

• setId

public void setId(int id)

Встановлює ідентифікатор ресурсу

Parameters:

id - ідентифікатор

getType

public java.lang.String getType()

Returns:

Повертає тип ресурсу

• setType

public void setType(java.lang.String type)

Встановлює тип ресурсу

Parameters:

type - тип ресурсу

getNumber

public int getNumber()

Returns:

Повертає кількість ресурсу

• setNumber

public void setNumber(int number)

Встановлює кількість

Parameters:

number - кількість ресурсу

getUnit

public java.lang.String getUnit()

Returns:

Повертає одиницю вимірювання

• setUnit

public void setUnit(java.lang.String unit)

Встановлює одиницю вимірювання

Parameters:

unit - одиниця вимірювання

getComment

public java.lang.String getComment()

Returns:

Повертає коментар

• setComment

public void setComment(java.lang.String comment)

Встановлює коментар

Parameters:

comment – коментар

‣‣ public class ResourceUse

extends java.lang.Object

Клас, що відповідає за представлення використання ресурсу з бази даних

Конструктори:

• ResourceUse

public ResourceUse()

Конструктор без параметрів

• ResourceUse

public ResourceUse(int count,

int resource\_id,

int task\_id)

Parameters:

count - кількість ресурсу

resource\_id - ідентифікатор ресурсу

task\_id - ідентифікатор задачі

•ResourceUse

public ResourceUse(int id,

int count,

int resource\_id,

int task\_id)

Parameters:

id - ідентифікатор використання ресурсу

count - кількість ресурсу

resource\_id - ідентифікатор ресурсу

task\_id - ідентифікатор задачі

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор використання ресурсу

• setId

public void setId(int id)

Встановлює ідентифікатор використання ресурсу

Parameters:

id - ідентифікатор

• getCount

public int getCount()

Returns:

Повертає кількість ресурсу

• setCount

public void setCount(int count)

Встановлює кількість ресурсу

Parameters:

count - кількість ресурсу

• getResource\_id

public int getResource\_id()

Returns:

Повертає ідентифікатор ресурсу, що використовується

• setResource\_id

public void setResource\_id(int resource\_id)

Встановлює ідентифікатор ресурсу, що використовується

Parameters:

resource\_id - ідентифікатор ресурсу

• getTask\_id

public int getTask\_id()

Returns:

Повертає ідентифікатор задачі

• setTask\_id

public void setTask\_id(int task\_id)

Встановлює ідентифікатор задачі

Parameters:

task\_id - ідентифікатор задачі

‣‣ public class ResultTest

extends java.lang.Object

Клас, що відповідає за представлення результату тесту з бази даних

Конструктори:

• ResultTest

public ResultTest()

Конструктор без параметрів

• ResultTest

public ResultTest(java.lang.String result,

java.lang.String comment,

int test\_id)

Parameters:

result - результат тесту

comment - коментар

test\_id - ідентифікатор тесту

• ResultTest

public ResultTest(int id,

java.lang.String result,

java.lang.String comment,

int test\_id)

Parameters:

id - ідентифікатор

result - результат тесту

comment - коментар

test\_id - ідентифікатор тесту

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор

• setId

public void setId(int id)

Встановлює ідентифікатор

Parameters:

id - ідентифікатор

• getResult

public java.lang.String getResult()

Returns:

Повертає результат тесту

• setResult

public void setResult(java.lang.String result)

Встановлює результат тесту

Parameters:

result - результат тесту

• getComment

public java.lang.String getComment()

Returns:

Повертає коментар

• setComment

public void setComment(java.lang.String comment)

Встановлює коментар

Parameters:

comment - коментар

• getTest\_id

public int getTest\_id()

Returns:

Повертає ідентифікатор тесту

• setTest\_id

public void setTest\_id(int test\_id)

Встановлює ідентифікатор тесту

Parameters:

test\_id - ідентифікатор тесту

‣‣ public class Task

extends java.lang.Object

Клас, що відповідає за представлення задачі з бази даних

Конструктори:

• Task

public Task()

Конструктор без параметрів

• Task

public Task(java.lang.String name,

java.lang.String formulation)

Parameters:

name - назва задачі

formulation - формулювання задачі

• Task

public Task(int id,

java.lang.String name,

java.lang.String formulation)

Parameters:

id - ідентифікатор задачі

name - назва задачі

formulation - формулювання задачі

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор задачі

• setId

public void setId(int id)

Встановлює ідентифікатор задачі

Parameters:

id - ідентифікатор задачі

• getName

public java.lang.String getName()

Returns:

Повертає назву задачі

• setName

public void setName(java.lang.String name)

Встановлює назву задачі

Parameters:

name - назва задачі

• getFormulation

public java.lang.String getFormulation()

Returns:

Повертає формулювання задачі

• setFormulation

public void setFormulation(java.lang.String formulation)

Встановлює формулювання задачі

Parameters:

formulation - формулювання задачі

‣‣ public class TaskResult

extends java.lang.Object

Клас, що відповідає за представлення результату аналізу задачі з бази даних

Конструктори:

• TaskResult

public TaskResult()

Конструктор без параметрів

• TaskResult

public TaskResult(boolean result,

java.lang.String comment,

int task\_id)

Parameters:

result - результат аналізу

comment - коментар

task\_id - ідентифікатор задачі

• TaskResult

public TaskResult(int id,

boolean result,

java.lang.String formulation,

int task\_id)

Parameters:

id - ідентифікатор результату аналізу задачі

result - результат аналізу

comment - коментар

task\_id - ідентифікатор задачі

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор результату аналізу задачі

• setId

public void setId(int id)

Встановлює ідентифікатор результату аналізу задачі

Parameters:

id - ідентифікатор

• getResult

public boolean getResult()

Returns:

Повертає результат аналізу задачі

• setResult

public void setResult(boolean result)

Встановлює результат аналізу задачі

Parameters:

result - результат аналізу задачі

• getComment

public java.lang.String getComment()

Returns:

Повертає коментар

• setComment

public void setComment(java.lang.String comment)

Встановлює коментар

Parameters:

comment - коментар

• getTask\_id

public int getTask\_id()

Returns:

Повертає ідентифікатор задачі

• setTask\_id

public void setTask\_id(int task\_id)

Встановлює ідентифікатор задачі

Parameters:

task\_id - ідентифікатор

java.lang.Object

app.model.Test

‣‣public class Test

extends java.lang.Object

Клас, що відповідає за представлення тесту з бази даних

Конструктори:

• Test

public Test()

Конструктор без параметрів

• Test

public Test(java.lang.String nameOfTested,

java.lang.String task,

java.lang.String comment)

Parameters:

nameOfTested - ім'я того хто пройшов тест

task - завдання тесту

comment - коментар

• Test

public Test(int id,

java.lang.String nameOfTested,

java.lang.String task,

java.lang.String comment)

Parameters:

id - ідентифікатор тесту

nameOfTested - ім'я того хто пройшов тест

task - завдання тесту

comment - коментар

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор тесту

• setId

public void setId(int id)

Встановлює ідентифікатор тесту

Parameters:

id - ідентифікатор

• getNameOfTested

public java.lang.String getNameOfTested()

Returns:

Повертає ім'я того хто пройшов тест

• setNameOfTested

public void setNameOfTested(java.lang.String nameOfTested)

ім'я того хто пройшов тест

Parameters:

nameOfTested - ім'я того хто пройшов тест

• getTask

public java.lang.String getTask()

Returns:

Повертає завдання тесту

• setTask

public void setTask(java.lang.String task)

Встановлює завдання тесту

Parameters:

task - завдання

• getComment

public java.lang.String getComment()

Returns:

Повертає коментар

• setComment

public void setComment(java.lang.String comment)

Встановлює коментар

Parameters:

comment - коментар

‣‣public class TestQuestion

extends java.lang.Object

Клас, що відповідає за представлення питання тесту з бази даних

Конструктори:

• TestQuestion

public TestQuestion()

Конструктор без параметрів

• TestQuestion

public TestQuestion(java.lang.String question,

java.lang.String answer,

int test\_id)

Parameters:

question - питання

answer - відповідь

test\_id - ідентифікатор тесту

• TestQuestion

public TestQuestion(int id,

java.lang.String question,

java.lang.String answer,

int test\_id)

Parameters:

id - ідентифікатор питання

question - питання

answer - відповідь

test\_id - ідентифікатор тесту

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор питання

• setId

public void setId(int id)

Встановлює ідентифікатор питання

Parameters:

id - ідентифікатор

• getQuestion

public java.lang.String getQuestion()

Returns:

Повертає питання

• setQuestion

public void setQuestion(java.lang.String question)

Встановлює питання

Parameters:

question - питання

• getAnswer

public java.lang.String getAnswer()

Returns:

Повертає відповідь

• setAnswer

public void setAnswer(java.lang.String answer)

Встановлює відповідь

Parameters:

answer - відповідь

• getTest\_id

public int getTest\_id()

Returns:

Повертає ідентифікатор тесту

• setTest\_id

public void setTest\_id(int test\_id)

Встановлює ідентифікатор тесту

Parameters:

test\_id - ідентифікатор тесту

‣‣public class User

extends java.lang.Object

Клас, що відповідає за представлення користувачів з бази даних

Конструктори:

• User

public User()

Конструктор без параметрів

• User

public User(java.lang.String name,

java.lang.String login,

java.lang.String password)

Parameters:

name - ім'я

login - логін

password - пароль

• User

public User(int id,

java.lang.String name,

java.lang.String login,

java.lang.String password)

Parameters:

id - ідентифікатор користувача

name - ім'я

login - логін

password – пароль

Методи:

• toString

public java.lang.String toString()

Overrides:

toString in class java.lang.Object

• getId

public int getId()

Returns:

Повертає ідентифікатор користувача

• setId

public void setId(int id)

Встановлює ідентифікатор користувача

Parameters:

id - ідентифікатор користувача

• getName

public java.lang.String getName()

Returns:

Повертає ім'я

• setName

public void setName(java.lang.String name)

Встановлює ім'я

Parameters:

name - ім'я

• getLogin

public java.lang.String getLogin()

Returns:

Повертає логін

• setLogin

public void setLogin(java.lang.String login)

Встановлює логін

Parameters:

login - логін

• getPassword

public java.lang.String getPassword()

Returns:

Повертає пароль

• setPassword

public void setPassword(java.lang.String password)

Встановлює пароль

Parameters:

password - пароль

⁃⁃⁃ app.view

‣‣public class AppWindow

extends java.lang.Object

Клас, що відповідає за зовнішній вигляд головного вікна програмного додатку

Поля:

• static javax.swing.JTable tasksTable

Таблиця задач

• static javax.swing.JTable testsTable

Таблиця тестів

• static javax.swing.JTable resourcesTable

Таблиця ресурсів

• static javax.swing.JTextField nameOfTested

Ім’я кандидату

Конструктори :

• AppWindow

public AppWindow()

Створює додаток

Методи:

• main

public static void main(java.lang.String[] args)

Запускає програмний додаток

‣‣public class InfoFrame

extends javax.swing.JFrame

Клас, що відповідає за вікно інформації

Конструктори:

• InfoFrame

public InfoFrame()

Створює вікно інформації

‣‣public class ConfigFrame

extends javax.swing.JFrame

Клас, що відповідає за вікно конфігурації

Конструктори:

• ConfigFrame

public ConfigFrame()

Створює вікно конфігурації

⁃⁃⁃ config

‣‣public interface Builder

Відповідає за завантаження конфігурації з файлу

Методи:

• load

void load(java.io.File file,

Config conf)

throws java.util.InvalidPropertiesFormatException,

java.io.IOException

Parameters:

file - файл

conf - об'єкт конфігурації

Throws:

java.util.InvalidPropertiesFormatException

java.io.IOException

‣‣ public interface Factory

Відповідає за створення потрібних об'єктів

Методи:

• createWriter

Writer createWriter()

Returns:

Повертає посилання на об'єкт, що відповідає за запис конфігурації

• createBuilder

Builder createBuilder()

Returns:

Повертає посилання на об'єкт, що відповідає за зчитування конфігурації

‣‣public interface Writer

Відповідає за зберігання конфігурації у файл

Методи:

• save

void save(java.io.File file,

Config conf)

Parameters:

file - файл

conf - об'єкт конфігурації

‣‣public class AppConfig

extends Config

Клас, що відповідає за налаштування програмного додатку

Методи:

• main

public static void main(java.lang.String[] args)

Головний метод

Parameters:

args -

• getInstance

public static Config getInstance()

Returns:

Повертає об'єкт конфігурації

• setInstance

public static void setInstance(Config instance)

Встановлює об'єкт конфігурації

Parameters:

instance -

‣‣public class Config

extends java.util.Properties

Клас конфігурації

‣‣public class XMLBuilder

extends java.lang.Object

implements Builder

Реалізація інтерфейсу Builder

Методи:

• load

public void load(java.io.File file,

Config conf)

throws java.util.InvalidPropertiesFormatException,

java.io.IOException

Specified by:

load in interface Builder

Parameters:

file - файл

conf - об'єкт конфігурації

Throws:

java.util.InvalidPropertiesFormatException

java.io.IOException

‣‣public class XMLFactory

extends java.lang.Object

implements Factory

Реалізація інтерфейсу Factory

Методи:

• createWriter

public XMLWriter createWriter()

Specified by:

createWriter in interface Factory

Returns:

Повертає посилання на об'єкт, що відповідає за запис конфігурації

• createBuilder

public XMLBuilder createBuilder()

Specified by:

createBuilder in interface Factory

Returns:

Повертає посилання на об'єкт, що відповідає за зчитування конфігурації

‣‣public class XMLWriter

extends java.lang.Object

implements Writer

Реалізація інтерфейсу Writer

Методи:

• save

public void save(java.io.File file,

Config conf)

Specified by:

save in interface Writer

Parameters:

file - файл

conf - об'єкт конфігурації

⁃⁃⁃ controller

‣‣public class Controller

extends java.lang.Object

implements java.lang.Runnable

Клас, що відповідає за контролер у програмному додатку

Поля:

• queve

public static java.util.concurrent.LinkedBlockingQueue<Command> queve

Черга, що використовується для посліжовного виклику команд

Конструктори:

• Controller

public Controller()

Методи:

• addCommand

public void addCommand(Command command)

Додає команду у чергу

Parameters:

command - команда

• executeCommand

public void executeCommand(Command command)

Виконує команду

Parameters:

command - команда

• run

public void run()

Забезпечує послідовне, неперервне виконання команд

Specified by:

run in interface java.lang.Runnable

‣‣public class SingleController

extends java.lang.Object

Забезпечує один екземпляр класу Controller

Методи:

• getInstance

public static Controller getInstance()

Returns:

Повертає екземпляр класу Controller

• setInstance

public static void setInstance(Controller instance)

Встановлює екземпляр класу Controller

Parameters:

instance -

‣‣public class Validator

extends java.lang.Object

Відповідає за правильність команди

Методи:

• ifValid

public boolean ifValid(Command command)

Returns:

Повертає true, якщо команда - вірна false, якщо - ні

Parameters:

command - команда

⁃⁃⁃ controller.commands

‣‣public interface Command

Команда

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class addResourceCommand

extends java.lang.Object

implements Command

Додає ресурс у базу та таблицю

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class changeConfigCommand

extends java.lang.Object

implements Command

Змінює налаштування програмного додатку

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class closeConnectionCommand

extends java.lang.Object

implements Command

Зачиняє з'єднання з базою даних

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class deleteResourceCommand

extends java.lang.Object

implements Command

Видаляє ресурс з бази даних та таблиці

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class modResourceCommand

extends java.lang.Object

implements Command

Змінює ресурс у базі даних та таблиці

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class openConnectionCommand

extends java.lang.Object

implements Command

Відкриває з'єднання з базою даних

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class openTestCommand

extends java.lang.Object

implements Command

Відкриває конкретний тест

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class resultTasksCommand

extends java.lang.Object

implements Command

Зберігає результат аналізу задачі

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class saveResultTestCommand

extends java.lang.Object

implements Command

Зберігає результат тесту

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class showResourceCommand

extends java.lang.Object

implements Command

Відображає у таблиці дані про ресурси

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class showTasksCommand

extends java.lang.Object

implements Command

Відображає у таблиці дані про задачі

Методи:

• execute

void execute()

Виконавчий метод команди

‣‣public class viewTasksCommand

extends java.lang.Object

implements Command

Забезпечує детальніший перегляд задачі

Методи:

• execute

void execute()

Виконавчий метод команди

⁃⁃⁃ dao

‣‣public interface CRUDInterface

Інтерфейс, що забезпечує зчитування даних з бази даних

Методи:

• open

void open()

throws dao.exception.DAOException

Відкриває зв'язок з базою даних

Throws:

dao.exception.DAOException

• close

void close()

throws dao.exception.DAOException

Закриває зв'язок з базою даних

Throws:

dao.exception.DAOException

• insert <T> T insert(T instance)

throws dao.exception.DAOException,

java.sql.SQLException

Вставляє об'єкт у базу даних

Parameters:

instance - об'єкт

Returns:

об'єкт, який був вставлений

Throws:

dao.exception.DAOException

java.sql.SQLException

• read <T> T read(java.lang.Class entityClass,

int id)

throws dao.exception.DAOException,

java.sql.SQLException

Зчитує з таблиці об'єкт за ідентифікатором

Parameters:

entityClass - клас, що представляє таблицю

id - ідентифікатор

Returns:

об'єкт, що був зчитаний

Throws:

dao.exception.DAOException

java.sql.SQLException

• update <T> void update(T instance)

throws dao.exception.DAOException,

java.sql.SQLException

Оновлює об'єкт у базі даних

Parameters:

instance - об'єкт

Throws:

dao.exception.DAOException

java.sql.SQLException

• delete <T> void delete(T instance)

throws dao.exception.DAOException,

java.sql.SQLException

Видаляє об'єкт з бази даних

Parameters:

instance - об'єкт

Throws:

dao.exception.DAOException

java.sql.SQLException

• select <T> java.util.List<T> select(java.lang.Class entityClass,

DAOFilter filter)

throws dao.exception.DAOException,

java.sql.SQLException

Повертає список , який містить таблицю з бази даних

Parameters:

entityClass - клас, що представляє таблицю

filter - фільтр

Returns:

список

Throws:

dao.exception.DAOException

java.sql.SQLException

• select <T> java.util.List<T> select(java.lang.String SQLString)

throws dao.exception.DAOException

не обов'язково реалізовувати

Parameters:

SQLString -

Returns:

Throws:

dao.exception.DAOException

‣‣public class CRUDImpl

extends java.lang.Object

implements CRUDInterface

Реалізація CRUD інтерфейсу

Методи:

• maxIdInTable

public int maxIdInTable(java.lang.Class entityClass)

throws dao.exception.DAOException,

java.sql.SQLException

Метод, що повертає максимальний ідентифікатор у заданій таблиці

Parameters:

entityClass - клас, що представляє таблицю

Returns:

максимальний ідентифікатор

Throws:

dao.exception.DAOException

java.sql.SQLException

• open

void open()

throws dao.exception.DAOException

Відкриває зв'язок з базою даних

Throws:

dao.exception.DAOException

• close

void close()

throws dao.exception.DAOException

Закриває зв'язок з базою даних

Throws:

dao.exception.DAOException

• insert <T> T insert(T instance)

throws dao.exception.DAOException,

java.sql.SQLException

Вставляє об'єкт у базу даних

Parameters:

instance - об'єкт

Returns:

об'єкт, який був вставлений

Throws:

dao.exception.DAOException

java.sql.SQLException

• read <T> T read(java.lang.Class entityClass,

int id)

throws dao.exception.DAOException,

java.sql.SQLException

Зчитує з таблиці об'єкт за ідентифікатором

Parameters:

entityClass - клас, що представляє таблицю

id - ідентифікатор

Returns:

об'єкт, що був зчитаний

Throws:

dao.exception.DAOException

java.sql.SQLException

• update <T> void update(T instance)

throws dao.exception.DAOException,

java.sql.SQLException

Оновлює об'єкт у базі даних

Parameters:

instance - об'єкт

Throws:

dao.exception.DAOException

java.sql.SQLException

• delete <T> void delete(T instance)

throws dao.exception.DAOException,

java.sql.SQLException

Видаляє об'єкт з бази даних

Parameters:

instance - об'єкт

Throws:

dao.exception.DAOException

java.sql.SQLException

• select

<T> java.util.List<T> select(java.lang.Class entityClass,

DAOFilter filter)

throws dao.exception.DAOException,

java.sql.SQLException

Повертає список , який містить таблицю з бази даних

Parameters:

entityClass - клас, що представляє таблицю

filter - фільтр

Returns:

список

Throws:

dao.exception.DAOException

java.sql.SQLException

• select <T> java.util.List<T> select(java.lang.String SQLString)

throws dao.exception.DAOException

не обов'язково реалізовувати

Parameters:

SQLString -

Returns:

Throws:

dao.exception.DAOException

‣‣

public class CRUDSingleton

extends java.lang.Object

Забезпечує єдиність CRUDImpl

Методи:

• getInstance

public static CRUDImpl getInstance()

Returns:

Повертає об'єкт

• setInstance

public static void setInstance(CRUDImpl instance)

Встановлює об'єкт

Parameters:

instance - об'єкт