**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Навчально-науковий інститут комп’ютерних**

**інформаційних технологій**



Лабораторна робота №5

з дисципліни «Об’єктно орієнтоване програмування»

«ДОСЛІДЖЕННЯ МОДУЛЬНОГО ТЕСТУВАННЯ»

Виконав студент:

групи ПІ-223Б

Карпов І.Б.

Перевірив викладач:

Дишлевий О.П.

Київ 2022

**Мета:** навчитись створювати модульні тести для розроблюваного програмного забезпечення.

**Завдання**

1. Спроектувати та розробити функціонал згідно з завданням. Виділити рівні представлення (PL), бізнес0логіки (BLL) та доступу до даних (DAL). PL можуть бути Console App, WinForms, WPF, DAL – потоки вводу-виводу чи серіалізація (один із способів). Контроль даних реалізувати на рівні бізнес логіки.
2. Написати модульні тести до бізнес-логіки, використовуючи певний фреймворк (MSTest, Nunit, Sunit тощо), які перевіряють коректність роботи розробленого коду. Кожен тест повинен відповідати принципу Triple A. За необхідності винести функціонал у окремі методи.
3. Дослідити ступінь покриття модульними тестами вихідного коду колекції, використовуючи засіб Code Coverage (чи будь-який інший).
4. \* Додаткове завдання – реалізувати зв’язок між рівнями за допомогою ін’єкції залежностей (Dependency Injection) для написання модульних тестів через «заглушки» (moch, stub).
5. Реалізація:

* для отримання оцінки «**задовільно**» достатньо реалізувати функціонал лише для класу **Student** (з відповідними операціями), та написати модульні тести. Покриття повинно бути 100%;
* для отримання оцінки «**добре**» розширити роботу реалізацією завдання з колонки «**Додаткові уміння**» та написати відповідні модульні тести. Покриття повинно бути 100%;
* для отримання оцінки «**відмінно**» реалізувати все з завдання відповідного варіанту у табл. Покриття функціоналу тестами для операцій з колонки «**Додаткові сутності та операції**» може бути не 100% - обов’язково повинні бути лише тести, які перевіряють деякі специфічні операції, що не перевірялися в класу **Student**.

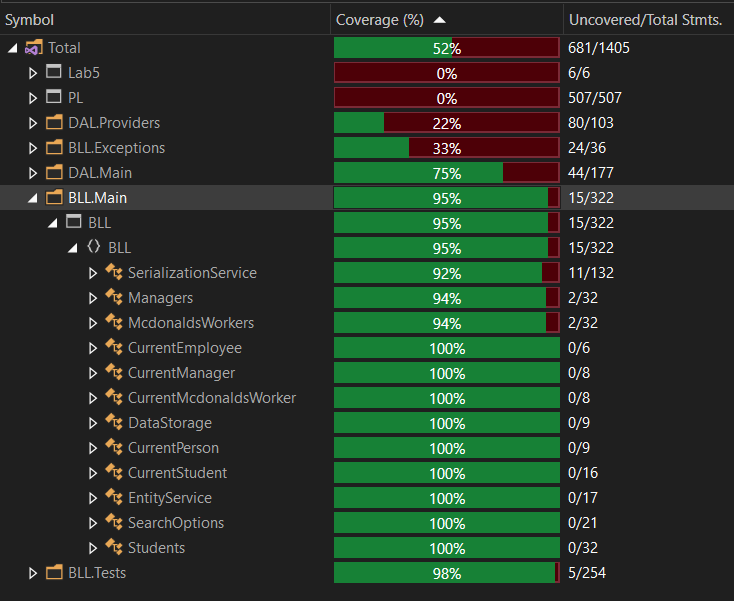
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Елемент класу Student** | **Операції зі студентами** | **Додаткові сутності та операції** | **Додаткові вміння** |
| **5** | Прізвище, Курс, Студентський квиток, Середній бал, Країна, Номер залікової книжки | Обчислити кількість студентів 3-го курсу, які проживають в Україні. Отримати їх дані з файлу.  Відрахування за неуспішність | McdonaldsWorjer, Manager  Готувати бургери з різних інгредієнтів | Грати в шахи |

**Хід роботи**

Код програми та приклад роботи зберігається на GitHub за цим посиланням:  
[https://github.com/IvanKarpov-1/course2-term1-OOP-lab](https://github.com/IvanKarpov-1/course2-term1-OOP-lab3)5

Для виконання роботи була взята друга частина третьої лабораторної роботи, оскільки завдання однакові, з додаванням нового проекту BLL.Tests де і були написані модульні тести до рівня BLL.

Покриття модульних тестів становить майже 100% на весь рівень BLL:



**Відповіді на контрольні запитання**

1. **Що таке модульне тестування?**

Модульне тестування (юніт-тестування) – процес, який дозволяє перевіряти на коректність окремі модулі вихідного коду програми.

1. **Як використовуються модульні тести?**

Ідея складається з того, щоб робити тести для кожної нетривіальної функції або методу. Це дозволяє швидко перевірити, чи не привела чергова зміна коду до регресії, а також полегшує виявлення і відсторонення таких помилок.

1. **Назвіть вимоги до юніт-тестів.**

Unit тест повинен бути:

* Автоматизованим і повторюваним;
* Простим в реалізаціі;
* Після написання він повинен залишатися для подальшого використання;
* Хто завгодно в команді повинен мати можливість запустити Unit тест;
* Повинен запускатися одним натиском кнопки;
* Повинен виконуватися швидко;
* Повиннен бути частиною контроля версій;
* В окремому проекті.

1. **Наведіть переваги використання юніт-тестів у розробці ПЗ.**

Однією з переваг unit-тестування є можливість виявлення помилок на ранніх етапах розробки самим розробником. Unit-тестування дозволяє бути впевненим, що з-за виправлення одного бага у вас не з'явився інший. Unit-тести служать деяким чином документацією до коду і путівником по ньому. Наступний плюс полягає в тому, що застосування unit-тестів дає більш якісне відділення інтерфейсу від реалізації.

1. **Що таке рефакторинг?**

Перетворення програмного коду, зміна внутрішньої структури програмного забезпечення для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи.

1. **Які метрики та засоби використовуються для оцінювання ефективності застосування юніт-тестів у проекті?**

Для вимірювання успішності впровадження юніт-тестів у вашому проекті слід використовувати дві метрики:

* Кількість багів в нових релізах (в т.ч. і регресії);
* Покриття коду.

Перша показує, чи є у наших дій результат, або ми даремно витрачаємо час, який могли б витратити на фічі. Друга - як багато нам ще належить зробити.

1. **Що таке TDD? Назвіть переваги застосування TDD.**

TDD - (test-driven development) - техніка розробки програмного забезпечення, яка ґрунтується на повторенні дуже коротких циклів розробки: спочатку пишеться тест, що покриває бажану зміну, потім пишеться код, Який дозволити пройти тест, и під кінець проводитися рефакторинг нового коду до відповідних стандартів. Правильно реалізоване TDD може підвищити якість вашого коду, Зменшити Кількість помилок, підвищити Впевненість в коді, скоротити годину для пошуку помилок, поліпшити дизайн коду.

Переваги:

* Оскільки ви пишете невеликі тести за раз, це змушує ваш код бути більш модульним
* Розробка на основі тестування допомагає також розробити гарну архітектуру
* Спрощує підтримку коду і рефакторинг
* Оскільки розробка на основі тестування допомагає вам писати модульні тести перед написанням кінцевого коду, рефакторинг коду стає простіше і швидше.

1. **Що таке принцип «Triple A»? Поясніть сутність його використання.**

Arrange-Act-Assert (AAA) - стандартний шаблон для написання тесту. Його суть в тому, що тест складається з трьох частин. Arrange (Встановлюємо) - виробляємо налаштування вхідних даних для тесту. Act (Діємо) - виконуємо дію, результати якого тестуємо. Assert (Перевіряємо) - перевіряємо результати виконання.

1. **Як використовуються в юніт-тестах клас Assert?**

Клас Assert використовується для перевірки отриманих результатів тестування.

1. **Як допомагає DI при написанні модульних тестів?**

Dependency Injection (DI) допомагає протестувати частину коду без задіяння іншого коду шляхом підміни іншого коду заглушками.

1. **Для чого потрібні модульні тести? Поясніть принцип FIRST.**

Призначення модульного тесту — тестування єдиного програмного модулю.

FISRT – fast, independent, repeatable, self-validating, timely.

* Швидкість (Fast). Тести мають виконуватися швидко. Всі ми знаємо, що розробники люди, а люди ліниві, оскільки ці висловлювання є "транзитивними", то можна зробити висновок, що люди теж ліниві. А лінива людина не захоче запускати тести при кожній зміні коду, якщо вони довго виконуватимуться.
* Незалежність (Independent). Результати виконання одного тесту не повинні бути вхідними для іншого. Усі тести повинні виконуватися у довільному порядку, оскільки в іншому випадку при збої одного тесту каскадно накриється виконання цілої групи тестів.
* Повторюваність (Repeatable). Тести повинні давати однакові результати незалежно від середовища виконання. Результати не повинні залежати від того, чи виконуються вони на вашому локальному комп'ютері, на комп'ютері сусіда або на білд-сервері. Інакше знайти кінці з кінцями буде непросто.
* Очевидність (Self-Validating). Результатом виконання тесту має бути булеве значення. Тест або пройшов, або не пройшов і це має бути легко зрозумілим для будь-якого розробника. Не потрібно змушувати людей читати логи лише для того, щоб визначити пройшов тест успішно чи ні.
* Своєчасність (Timely). Тести мають створюватися своєчасно. Несвоєчасність написання тестів є головною причиною того, що вони відкладаються на потім, а це потім потім так і не настає. Навіть якщо ви і не писатимете тести перед кодом (хоча цей варіант вже довів свою життєздатність) їх потрібно писати як мінімум паралельно з кодом.

1. **Як правильно написати модульний тест?**

* Модульний тест повинен тестувати тільки одну дію;
* Якщо потрібно протестувати метод з умовними операторами, то створюємо окремі тести для кожної гілки оператора;
* Не потрібно створювати методи/класи спеціально для тестів;

1. **Чим модульні тести відрізняються від інтеграційних тестів?**

Модульне тестування означає окреме тестування окремих модулів програми (без будь-якої взаємодії із залежностями), щоб підтвердити, що код робить все правильно.

Інтеграційне тестування означає перевірку того, чи працюють різні модулі нормально, якщо їх об’єднати як групу.

Для модульного тестування наступними тестами можуть бути:

* Довжина поля - поля імені користувача та пароля.
* Значення поля введення повинні бути дійсними.
* Кнопка входу вмикається лише після введення дійсних значень (Формат і по довжині) в обидва поля.

Для інтеграційного тестування наступними тестами можуть бути:

* Користувач бачить привітальне повідомлення після введення дійсних значень та натискання кнопки входу.
* Користувач повинен перейти на сторінку привітання або домашню сторінку після дійсного введення та натискання кнопки Вхід.

1. **Що таке Mock та Stub? З якою ціллю вони використовуються в юніт-тестах?**

Mocks - об'єкти, які налаштовуються (наприклад, специфічно кожному тесту) і дозволяють задати очікування у вигляді свого роду специфікації викликів, які ми плануємо отримати. Перевірки відповідності очікуванням проводяться через виклики до об'єкта Mock.

Stubs - забезпечують жорстко зашиту відповідь на виклики під час тестування. Застосовуються для заміни об'єктів, які забезпечують SUT вхідними даними. Також вони можуть зберігати в собі інформацію про виклик (наприклад, параметри або кількість цих викликів) - такі іноді називають своїм терміном Test Spy. Такий "запис" дозволяє оцінити роботу SUT, якщо стан самого SUT не змінюється.

Моки і стаби використовуються для того, щоб позбутися залежності коду методів, що тестуються, від інших модулів програми.