**Лабораторна робота №6.** Тестування класів. **Контрольна робота**.

**Мета:** Набуття навичок у формуванні плану демонстраційного тестування.

**Порядок виконання роботи**

1. Ознайомитися з теоретичним відомостями.
2. Лабораторна робота є частиною контрольної роботи .

*Для визначеної теми курсової роботи завдання виконуєте для класів, які розробляєте в курсовій. В* контрольній роботі *фіксуєте тему курсової. Теми курсових викладено на гіт-хаб.*

1. Розробити тест-план для демонстраційного тестування. Цей тест-план є частиною контрольної роботи.
2. За цим тест-планом в подальшому постійно перевіряти працездатність застосунку.
3. Результати оформити у вигляді звіту з виконання контрольної роботи.
4. Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

**<Номер групи> <Номер лабораторної><Прізвище англійською>**

Наприклад, 31-01Ivanov.doc.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР <Номер лабораторної>-<Прізвище>**

**Строк відсилки ЛР ІПЗ-31 11.12.2023**

**ІПЗ-32 14.12.2023**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ООП<Номер групи>-Запитання-<Прізвище >.**

Мінімальна оцінка за контрольну роботу з урахуванням включення до неї ЛР-29, максимальна – 46.

Оцінка знижується за несвоєчасну здачу роботи (-10)

**КОНТРОЛЬНА РОБОТА**

Відповідно до теми власної курсової роботи необхідно розробити

1. UML-діаграму класів, де визначаються і виділяються класи, які на цей момент студент бачить в своєму проекті (максимальна оцінка -18 балів)

Як мінімум 2 класи з визначенням членів класу та методів (як мінімум встановлення значення set\_... та надання значення get\_...). Під діаграмою надати пояснення щодо призначення методів у вигляді

|  |  |
| --- | --- |
| Назва методу | Призначення методу |
|  |  |
|  |  |

1. Надати схему меню застосунку (максимальна оцінка -14 балів)
2. Скласти тест – план демонстраційного тестування (максимальна оцінка -14 балів).

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Тестування програмного забезпечення — техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином. Тестування в рамках процесу розробки ПЗ важливе з таких причин:

1. Тестування дозволяє перевірити, чи правильно реалізовано усі вимоги до ПЗ, що розроблялось.

2. Тестування допомагає у виявленні дефектів / помилок та забезпечує їх розпізнавання / вирішення до етапу розгортання програмного забезпечення.

3. Тестування пом’якшує наслідки та ризики втрат, якщо програмний продукт все ж випустили по неправильних вимогах. Вимоги в такому випадку намагаються частково виправити, переоцінити та покращити програму.

4. Тестування також демонструє, що створене ПЗ працює відповідає також вимогам до продуктивності.

5. Тестування допомагає перевірити належну інтеграцію та взаємодію програми з навколишнім середовищем.

Базові поняття

*План Тестування* (Test Plan) – це документ, що описує весь обсяг робіт з тестування, починаючи з опису об’єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх дозволу .

*Тестовий випадок* (Test Case) – це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується або її частини

Існує цілий ряд видів тестування. Розглянемо одну з класифікацій тестування за ознакою позитивності сценаріїв.

**Позитивне тестування** – це тестування із застосуванням сценаріїв, які відповідають нормальній (штатній, очікуваній) поведінці системи. З його допомогою ми можемо визначити, що система робить те, для чого і була створена. Наприклад, множення на калькуляторі цифр 3 і 5. Таке тестування ще називають ***демонстративним***.

**Негативним** називають тестування, в рамках якого застосовуються сценарії, які відповідають позаштатній поведінці тестованої системи. Це можуть бути, наприклад, виняткові ситуації або невірні дані. На прикладі калькулятора, це множення числа 3 на грушу. Значення "груша" не є дійсним для калькулятора.

Перш за все негативне тестування направлено на перевірку стійкості системи до різних впливів, валідації невірних даних, обробки виняткових ситуацій. Сценарії позитивного тестування, в свою чергу, спрямовані на перевірку роботи системи з тими типами даних, для яких вона розроблялася.

Створення позитивних сценаріїв (тест-кейсів), як правило, передує створенню негативних тест-кейсів. Спочатку ми перевіряємо роботу системи, коли наш умовний користувач працює з системою "правильно", а потім приступаємо до перевірки відгуку системи на користувача, який допускає різні помилки (введення невірних даних, наприклад). І наша система повинна бути готова відповісти на невірний запит. Це і є мета негативного тестування.

|  | **Позитивне тестування** | **Негативне тестування** |
| --- | --- | --- |
| Опис | Перевірка наявності очікуваних функцій | Перевірка наявності непередбачуваних проблем |
| Мета | Виявлення успішних сценаріїв використання | Виявлення помилок, виключень і недоліків |
| Характеристики | Додаткові вимоги до функцій, властивостей та взаємодії | Негативні тестові сценарії та граничні умови |
| Тестові дані | Спеціально створені дані для перевірки очікуваних результатів | Дані, що відповідають негативним умовам |
| Очікуваний результат | Підтвердження, що програма працює, як очікувалося | Виявлення помилок, помилкових повідомлень та інших непередбачуваних проблем |
| Приклади | Може включати сценарії коректного введення даних та отримання очікуваних результатів | Негативне тестування може включати намагання ввести неправильні дані або викликати винятки |

Демонстраційне тестування призначене показати працездатність застосунку на тестових даних при виконанні тестового сценарію. Тестові дані – це вхідні і вихідні дані, підібрані так, щоб для заданих вхідних даних були відомі вихідні. Для вхідних даних програма (модуль), що перевіряється, видає свої вихідні дані. Якщо результат роботи програми не збігається з тестовими  даними, то це значить, що в програмі є помилка. Якщо тестові дані містять помилку, то результат демонстраційного тестування демонструє дії програми по обробленню такої ситуації (наприклад, виведення повідомлення про помилкові дані, виконання дій передбачених алгоритмом в цій ситуації).

Типова процедура тестування полягає у підготовці та виконанні тестових прикладів (або просто тестів). Кожен тестовий приклад перевіряє одну "ситуацію" в поведінці модуля і складається зі списку значень, переданих на вхід модуля, описи запуску і виконання переробки даних - тестового сценарію і списку значень, які очікуються на виході модуля в разі його коректної поведінки. Виконання починається з передачі модулю вхідних даних і запуску сценарію. Реальні вихідні дані,отримані від модуля в результаті виконання сценарію, зберігаються і порівнюються з очікуваними. У разі їх збігу тест вважається пройденим, в іншому випадку - НЕ пройденим. Кожний не пройдений тест вказує на дефект або в тестуємому модулі або в описі тесту.

Сукупність описів тестових прикладів становить тест-план, який визначає процедуру тестування програмного модуля. Тест-план задає не тільки самі тестові приклади, а й порядок їх слідування, який також може бути важливий. Приклад тест – плану наданий в табл.1.

Таблиця 1 – Приклад тест - плану

| Визначення мети | Послідовність дій | Очікуваний результат |
| --- | --- | --- |
| Перевірка інтерактивноговведення даних та запису їх до файлу | 1. Обрати функцію "Інтерактивневведення даних"  2. Обрати клас, до якого вводяться дані (наприклад, "Студент")  3. Ввести дані по студенту.  4. На запит "Дозаписати дані до файлу?" відповісти "Так"  5. На запит "Введення завершено" відповісти "Так". | Повернення відбувається в головне меню. При перегляді даних у файлі введена інформація збережена правильно. |
| Перевірка реакції на неправильне обрання позиції меню. | Першою обирається позиція меню "Сортування даних" | Видається повідомлення "Введіть /завантажте дані", відбувається повернення в головне меню. |

UML — уніфікована мова моделювання, яка використовується у парадигмі об'єктно-орієнтованого програмування. Вона є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення, використовує графічні позначення для створення абстрактної моделі системи. В такій діаграмі повинні знайти відображення всі структурні об'єкти програмного додатку, а також їх важливі методи та властивості, які відповідають за взаємодію класів між собою та безпосереднє виконання поставленої задачі.

В діаграмі класів визначаються перелік та опис призначення об'єктів, а також структури об'єктів (поля і методи). Приклад діаграми класів надається на рис.1. При виконанні КР командою на UML-діаграмі класів визначаються і виділяються класи, які реалізуються студентом – членом команди.

Інструменти по розробці UML-діаграми класів розглядались в ЛР№5.

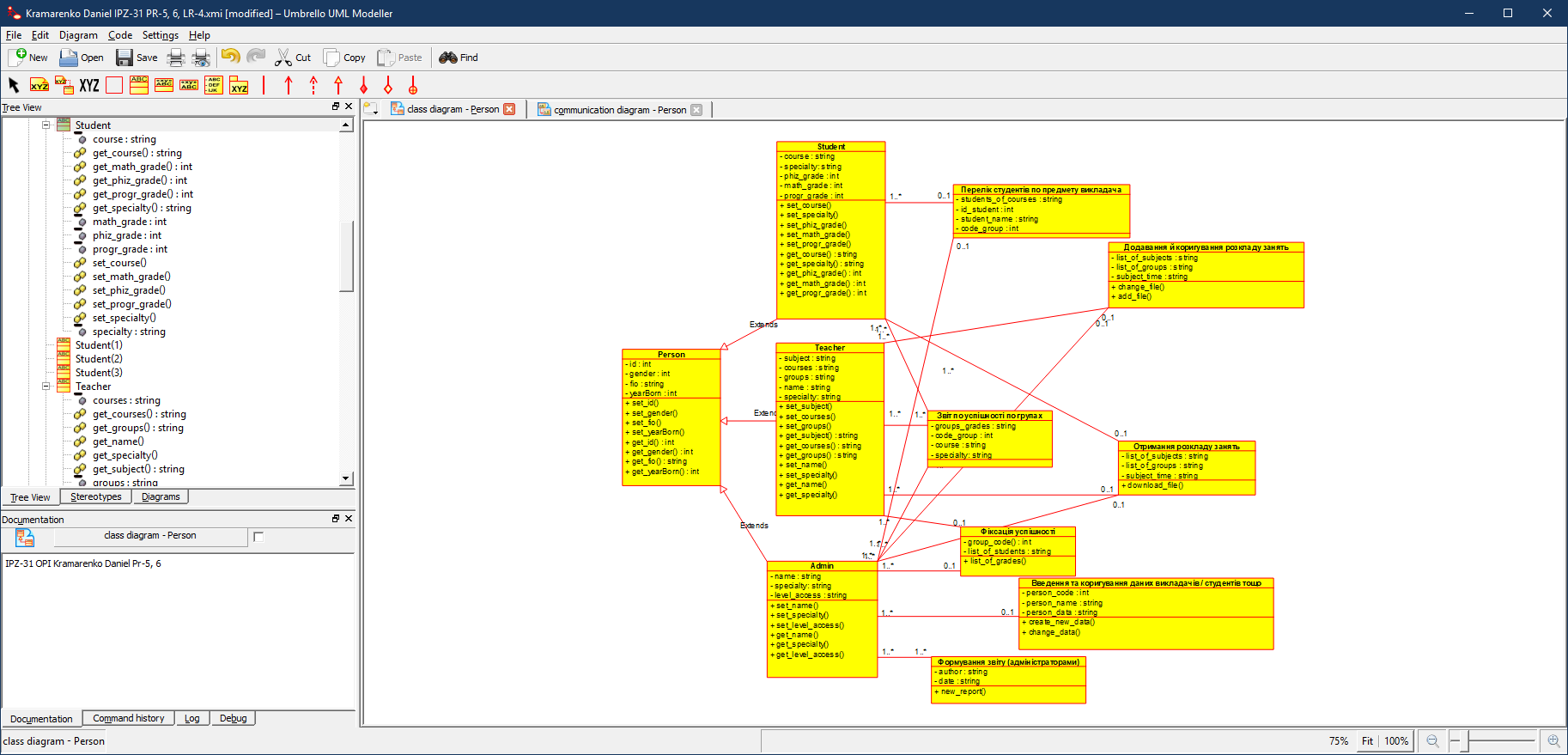


Рисунок – 1 Приклад UML-діаграми

Приклад схеми меню консольного застосунку надається на рис.2. До схеми потрібно надати короткі пояснення до кожної гілки меню.

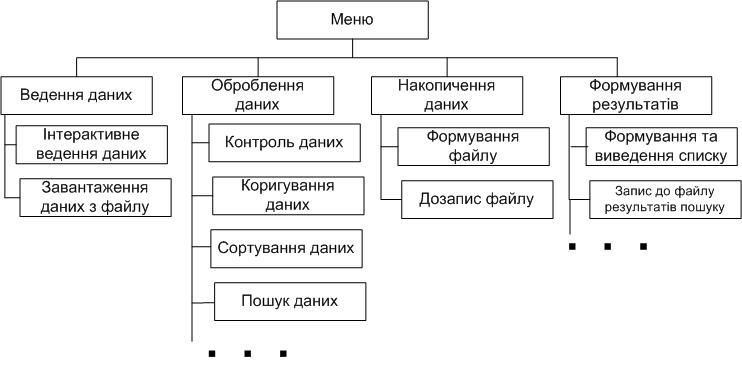


Рисунок – 2 Приклад ієрархічного меню застосунку