**Лабораторна робота №11-1.** Розроблення тестів для перевірки функціонування класів

**Мета:** Набуття навичок в розробці тестування класів.

**Завдання**

1. Для власного проекту розробити тести для всіх класів.
2. Для тестування створити програму, виокремивши класи, і в головній програмі - функції main створити об‘єкти, викликати всі методи і перевірити працездатність класів.
3. Для вирішення цього завдання в конструкторах і деструкторах надати повідомлення про виклик.
4. Оформити однією програмою, до якої включити опис класів. В першому рядку програми та заголовкового файлу повинні бути записаними в коментарі номер групи та прізвище, а також номер ЛР
5. Продовжити працювати над програмою своєї курсової з ООП, закінчити попередні завдання з урахуванням наданих зауважень (для тих, хто не здав). Результати надсилати разом зі сформованими файлами.

Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

Для заголовкового файлу

**<Прізвище англійською>.h**

Наприклад, Ivanov.h

Для cpp-файлу:

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

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

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

**Строк відсилки ЛР ІПЗ-31 15.05.2024**

**ІПЗ-32 16.05.05.2024**

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

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

**Теоретичні відомості**

Під час тестування ПЗ використовуються різні методи та підходи, включно з функціональним тестуванням, інтеграційним тестуванням, системним тестуванням та багатьма іншими. Кожен із них має свої особливості та цілі. У цій статті ми зосередимося на одному з таких методів тестування – юніт-тести. Вони є основою для перевірки невеликих, ізольованих частин коду – так званих “юнітів”. І в статті ми розглянемо, чому вони важливі та як їх можна написати для забезпечення високої якості програмного продукту.

**Unit тести** – це автоматичні тести, які перевіряють невеликі частини коду, такі як функції або методи, ізольовано від решти системи. Вони дають змогу розробникам переконатися, що кожна частина коду працює правильно і відповідає очікуваній поведінці. Вони пишуться мовою програмування і можуть бути запущені автоматично для швидкої перевірки коду при кожній його зміні.

Термін “юніт” у контексті теми позначає мінімально можливу частину коду, яку можна протестувати ізольовано. Це може бути функція, метод або навіть окремий рядок коду. Кожен “юніт” має бути протестований окремо від решти коду, щоб переконатися в його коректності. Такий підхід дає змогу виявити помилки та проблеми в коді на ранніх стадіях розробки.

Що таке unit test у розробці ПЗ і наскільки вони важливі?

Давайте уявимо собі, що у вас є шматочок коду, який ви щойно написали. Ви думаєте, що він працює правильно, але як ви можете бути впевнені? Ось де на допомогу приходять юніт-тести. Вони дозволяють вам перевірити, що ваш код виконує свої функції так, як задумано. Вони допомагають виявити помилки і дефекти в коді на ранніх етапах розробки. Це дає змогу заощадити час і зусилля, які в іншому разі могли б бути витрачені на пошук і виправлення проблем у коді, що вже склався. Юніт-тести допомагають запобігти появі нових помилок під час внесення змін до коду в майбутньому.

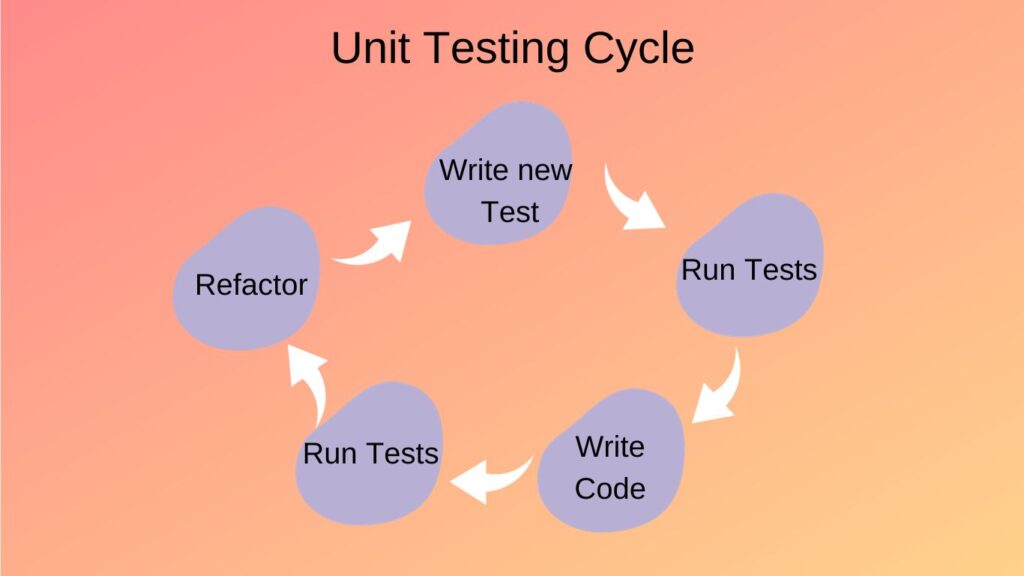
Коли ви їх пишете, то декомпозуєте ваш код на окремі “юніти” і перевіряєте кожен із них окремо. Це допомагає виявити проблеми в окремих компонентах коду і переконатися, що вони працюють правильно в ізоляції.

Юніт-тести також сприяють підвищенню надійності коду. Якщо у вас є хороший набір тестів, які перевіряють різні сценарії використання, ви можете бути впевнені, що ваш код буде працювати стабільно і передбачувано навіть у складних ситуаціях.

Крім того, юніт-тести слугують документацією до коду – це коли ви описуєте очікувану поведінку вашого коду в явному вигляді. Це робить ваш код зрозумілішим і полегшує його використання та підтримку для інших розробників, які можуть працювати з ним у майбутньому.

Написання юніт-тестів включає кілька кроків:

* Визначення тестованих “юнітів”. Виберіть функції, методи або класи, які ви хочете протестувати. Визначте, які частини вашого коду мають бути покриті тестами.



* Розробка тестових випадків. Визначте різні сценарії використання і стани, які мають бути протестовані. Напишіть тестові випадки для кожного сценарію, де ви перевіряєте очікувані результати та поведінку коду.
* Написання тестового коду. Напишіть код для кожного тестового випадку, який перевіряє очікуваний результат. Використовуйте спеціальні фреймворки та бібліотеки для тестування, щоб спростити процес написання та виконання тестів.
* Запуск тестів. Запустіть усі написані тести та перевірте їхні результати. Переконайтеся, що всі тести пройшли успішно і код працює правильно.
* Аналіз результатів і виправлення помилок. Вивчіть результати тестів і зверніть увагу на будь-які помилки або несподівану поведінку. Виправте проблеми у вашому коді та перезапустіть тести для перевірки виправлень.

**Як правильно писати юніт тести?**

Тут слід дотримуватися кількох принципів:

* **Напишіть тести одразу.** Краще писати тести разом із кодом або якомога раніше в процесі розробки. Це допомагає запобігти появі помилок і спростити виявлення проблем.
* **Будьте зрозумілі та специфічні.** Називайте ваші тестові випадки та функції таким чином, щоб було зрозуміло, що вони перевіряють. Зробіть кожен тестовий випадок максимально специфічним і фокусуйтеся на конкретних аспектах коду.
* **Використовуйте асерти.** Використовуйте твердження (assertions) для перевірки очікуваних результатів. Переконайтеся, що ваш код повертає очікувані значення і поводиться так, як ви припускаєте.
* **Підтримуйте свої тести.** Оновлюйте ваші тести під час внесення змін до коду. Переконайтеся, що вони все ще актуальні та продовжують перевіряти правильну поведінку вашого коду.
* **Запускайте тести регулярно**, щоб переконатися, що вони все ще працюють. Це допоможе вам швидко виявити будь-які проблеми у вашому коді.

Як бачите – це мистецтво, яке потребує практики та досвіду. Дотримуючись цих кроків і принципів, ви зможете створювати надійні та ефективні тести для вашого коду.

## Приклади

Давайте розглянемо кілька прикладів простих юніт-тестів для наочності;

1. **Приклад для функції додавання:**

Припустимо, у вас є функція add(a, b), яка додає два числа. Ви можете написати наступний юніт-тест для перевірки її роботи:

def test\_add():

    result = add(2, 3)

    assert result == 5, "Очікуваний результат: 5"

У цьому тесті ми викликаємо функцію add(2, 3) і порівнюємо її результат з очікуваним значенням 5. Якщо результат не збігається з очікуваним, тест завершиться з помилкою.

1. **Приклад для класу:**

Припустимо, у вас є клас Calculator, який має метод multiply(a, b), що виконує множення двох чисел. Ви можете написати такий юніт-тест для перевірки його роботи:

def test\_multiply():

    calculator = Calculator()

    result = calculator.multiply(2, 3)

    assert result == 6, "Очікуваний результат: 6"

У цьому тесті ми створюємо екземпляр класу Calculator, викликаємо його метод multiply(2, 3) і порівнюємо результат з очікуваним значенням 6.

Але виникає питання, як інтерпретувати результати? Вони зазвичай представлені у вигляді звітів або виведення на екран. Коли ви запускаєте тести, вони можуть показати таку інформацію:

* **Пройдені тести:** якщо все пройшло успішно, ви побачите повідомлення про проходження тесту або його статус “пройдено”. Це означає, що код, який було протестовано, працює так, як очікувалося.
* **Невдалі тести:** якщо ж не все гладко, ви побачите повідомлення про невдачу тесту або його статус “не пройдено”. Це означає, що було виявлено проблему в коді, яку потрібно виправити.
* **Зведення результатів:** наприкінці виконання всіх тестів ви можете побачити зведення результатів, яке показує загальну кількість пройдених і невдалих тестів. Це допомагає вам зрозуміти, наскільки добре ваш код пройшов тестування.