**Інтеграційні програмні системи**

Практична робота №4
**Автоматизовані тести**

Завдання присвячене створенню автоматичних тестів у вашому проекті. Та складається з двох частин. Для створення тестів вам необхідно буде використати один із фреймворків. Вибір останнього ніяк не обмежується. Для Java вас може зацікавити JUnit з бібліотеками для написання тверджень у тестах (AssertJ або Hamcrest) та створення моків (Mockito). Для JavaScript одним із популярних фреймворків є Mocha.

Наведені далі приклади написані на Java, але концепції та підходи є спільними для будь-яких мов чи середовищ.

**Завдання 1. Пряме тестування коду.**

Складність тестів напряму залежить від орагнізації та складності основного коду. Найпростіше тестувати роботу функції. Наприклад,

```
assertThat(idFromUri("http://kpi-integration.appspot.com/download/
pratice-4"))
    .isEqualTo("practice-4");
assertThat(idFromUri("http://kpi-integration.appspot.com/download/
pratice-3?format=pdf"))
    .isEqualTo("practice-3");
assertThat(idFromUri("http://kpi-integration.appspot.com/download/
pratice-2#part2"))
    .isEqualTo("practice-2");
```

Іншим прикладом простого випадку є тестування поведінки класа, який не має залежностей.

```
Calculator calc = new Calculator();
assertThat(calc.result()).isZero();
calc.add(5).mul(2);
assertThat(calc.result()).isEqualTo(10);
calc.reset();
assertThat(calc.result()).isZero();
```

Першим завданням є написання подібного тесту у вашому проекті - тесту, у якому за допомогою таких тверджень ви напряму тестуєте свій код.

У вашому репозиторії мають бути коміти від усіх членів команди, де кожен додає принаймні один такий тест.
За це завдання команда отримує 3 бали.

**Завдання 2. Тестування поведінки з моками.**

У випадку, коли у модуля, який ви тестуєте є залежності на інші частини вашої системи, тестування може ускалднюватися. І для створення юніт-тестів стають у нагоді моки. Уявимо, ситуацію, що модуль Computer користується модулем Battery для отримання даних про заряд батареї та модулем PowerManager для реалізації операцій вимкнення чи перезавантаження. І ми хочемо протестувати, що модуль Computer при нульовому заряді батареї приймає рішення про вимкнення.

```
Battery battery = mock(Battery.class);
PowerManager power = mock(PowerManager.class);
```

```
Computer computer = new Computer(battery, power);
doReturn(0).when(battery).getLevel();
computer.updateState();
assertThat(computer.isOn()).isFalse();
verify(power).shutdown();
```

У цьому тесті за допомогою підходу Dependency Injection ми підмінили реалізацію модулів, які використовує Computer. Далі ми сконфігурували повернення значення 0 методом getLevel(), спровокували оновлення стану та проверифікували, що було викликано метод shutdown().

Справжні реалізації Battery та PowerManager можуть комунікувати з реальною апаратурою. Тому їхнє використання в тестах може бути дуже ускладеним (уявіть собі тест, у результаті якого вимикається ваш комп'ютер). Тому ці реалізації замінено моками.

Ваше завдання полягає у написанні подібних тестів у своєму проекті. Це може бути тестування логіки взаємодії з сервером, замінюючи реалізацію комунікації з сервером на мок. Або тестування бізнес логіки якогось сервісу, який користується моком DAO.

Коміти з даним типом тестів мають бути в усіх членів команди.
Завдання оцінюється в 4 бали.

**Завдання 3 (+1 бал)**
Ваш README файл має містити інструкції, як запустити тести.

**Не забувайте**, що коли у вас є проблеми з реалізацією, то можна задати питання у нашій групі. І викладач, і інші студенти, будуть раді вам допомогти.
https://groups.google.com/forum/#!forum/kpi-integration-2015