Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №3 з дисципліни «Технології паралельних обчислень»

Виконав:

Гончаренко Дмитро ІТ-01

Захищено з оцінкою \_\_\_\_\_\_\_ Дата захисту \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірила: проф. Стеценко І. В.

Завдання:

1. Реалізуйте програмний код, даний у лістингу, та протестуйте його при

різних значеннях параметрів. Модифікуйте програму, використовуючи

методи управління потоками, так, щоб її робота була завжди коректною.

Запропонуйте три різних варіанти управління. 30 балів.

2. Реалізуйте приклад Producer-Consumer application (див.

https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html ).

Модифікуйте масив даних цієї програми, які читаються, у масив чисел

заданого розміру (100, 1000 або 5000) та протестуйте програму. Зробіть

висновок про правильність роботи програми. 20 балів.

3. Реалізуйте роботу електронного журналу групи, в якому зберігаються

оцінки з однієї дисципліни трьох груп студентів. Кожного тижня лектор і

його 3 асистенти виставляють оцінки з дисципліни за 100-бальною

шкалою. 40 балів.

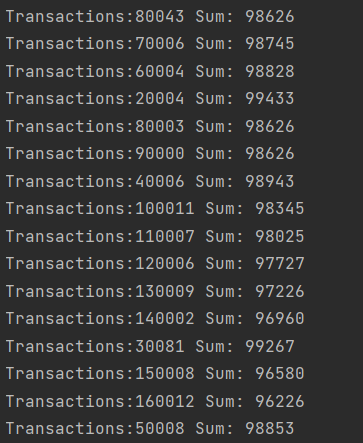
4. Зробіть висновки про використання методів управління потоками в java.

10 балів.

Хід роботи

1. Початковий лістинг представляє простий банківський сценарій з використанням багатопоточності. Основна мета програми - виконання переказу коштів між рахунками у банку. Програма створює NACCOUNTS рахунків з початковим балансом INITIAL\_BALANCE. Клас Bank містить метод transfer(), який відповідає за переказ коштів між рахунками. Кожен раз, коли виконується переказ, збільшується лічильник ntransacts, який відстежує кількість здійснених транзакцій. При досягненні кратного значення NTEST, викликається метод test(), який виводить повідомлення про кількість транзакцій та суму на рахунках. Клас TransferThread є потоком, який виконує випадкові перекази між рахунками. У циклі run() виконується REPS ітерацій, під час кожної ітерації генеруються випадкові номери рахунків та сума переказу, і викликається метод transfer() класу Bank для здійснення переказу коштів.

Реалізація програмного коду з лістингу



Однак, у початковому лістингу відсутні механізми синхронізації, тому програма працює некоректно в умовах багатопоточності. Виникають проблеми зі зміною балансів на рахунках, дублювання транзакцій та неочікуване поведінка програми.

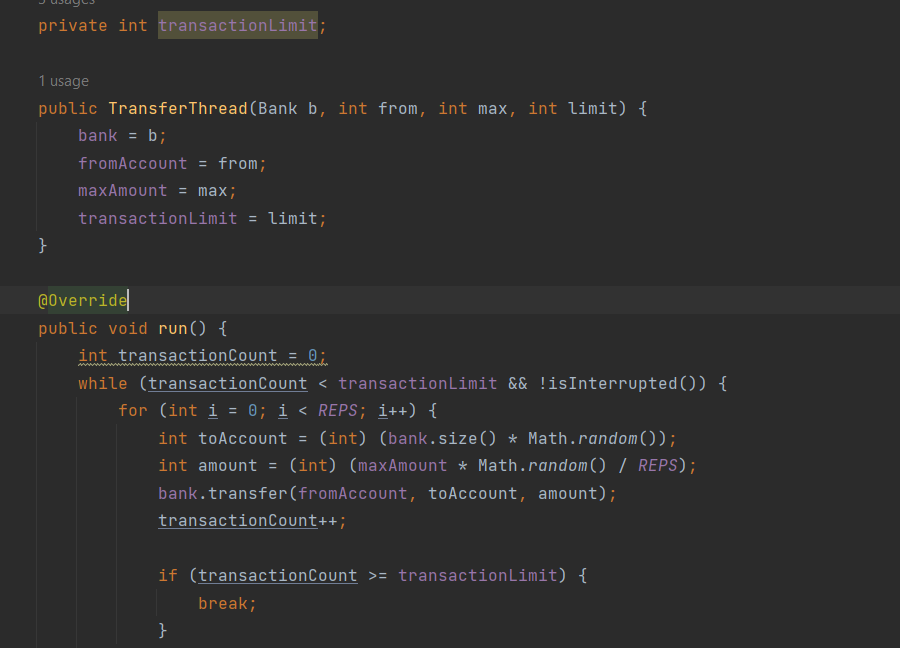
Підсумовуючи, ось проблеми, які треба вирішити:

* Транзакції відбуваються в неправильному порядку та не на однакових проміжках
* Загальна сума поступово зменшується
* Нескінчений цикл ітерацій

Почнемо з останії(нескінчений цикл ітерацій). Проблема з нескінченим циклом ітерацій виникає через відсутність умови зупинки циклу в потоці TransferThread.

Щоб вирішити цю проблему, я додав умову зупинки циклу в потоці TransferThread. Ця умова пов'язана з досягненням ліміту кількості транзакцій або зі станом переривання потоку.

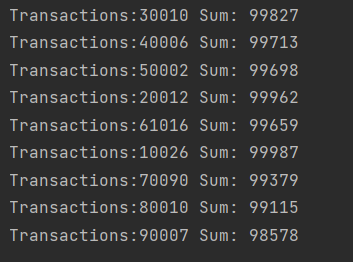
Оновлений код



У цьому оновленому коді я додав змінну transactionLimit, яка встановлює ліміт кількості транзакцій для кожного потоку. В циклі run() перевіряється, чи не досягнуто ліміту кількості транзакцій або чи не був потік перерваний. Якщо будь-яка з цих умов виконується, цикл завершується, а потік зупиняється.

За допомогою цих змін, цикл в потоці TransferThread буде зупинятися після досягнення ліміту кількості транзакцій або в разі переривання потоку. Це допоможе уникнути нескінченого виконання транзакцій та забезпечити правильну роботу програми.

Результат



Далі потрібно вирішити дві інші проблеми.

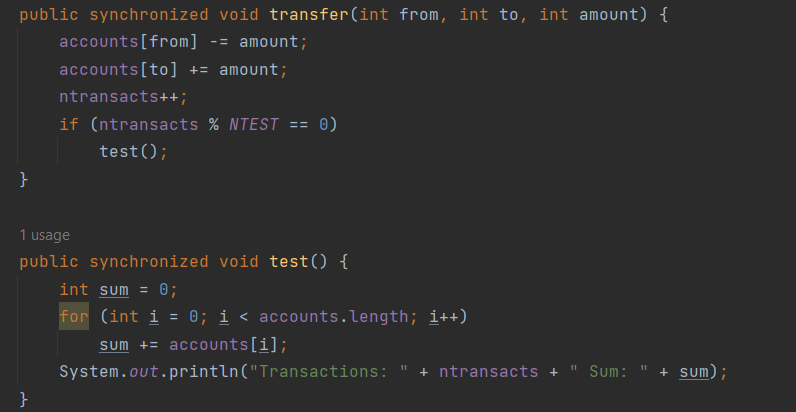
1 спосіб - використання ключового слова synchronized

Ключове слово synchronized в Java використовується для створення синхронізованих блоків коду або синхронізованих методів.

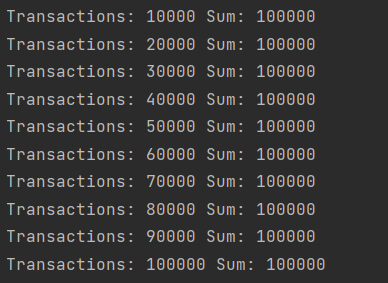
Коли метод оголошується як synchronized, тільки один потік може виконувати цей метод в будь-який момент часу. Інші потоки, які намагаються виконати цей метод, повинні зачекати, поки перший потік завершить виконання. Цей підхід дозволяє забезпечити взаємовиключення потоків та правильний порядок виконання коду. Коли потік виконує синхронізований метод, інші потоки мають зачекати, поки виконання поточного методу не завершиться.

У випадку, коли кілька потоків намагаються виконати різні синхронізовані методи на одному об'єкті, вони також будуть блоковані один від одного. Це допомагає уникнути проблем конкурентного доступу до спільних ресурсів та забезпечує коректну роботу програми у багатопотоковому середовищі.

Використання ключового слова synchronized для методів transfer() та test() допоможе забезпечити взаємовиключення потоків та правильний порядок виконання.

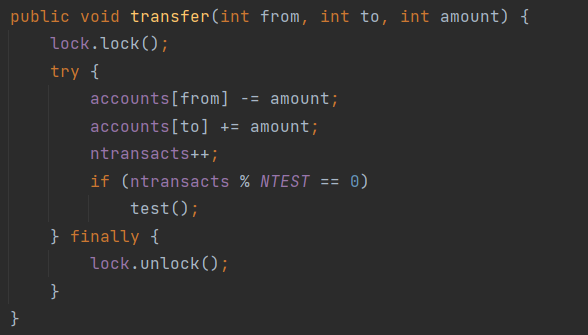


Результат

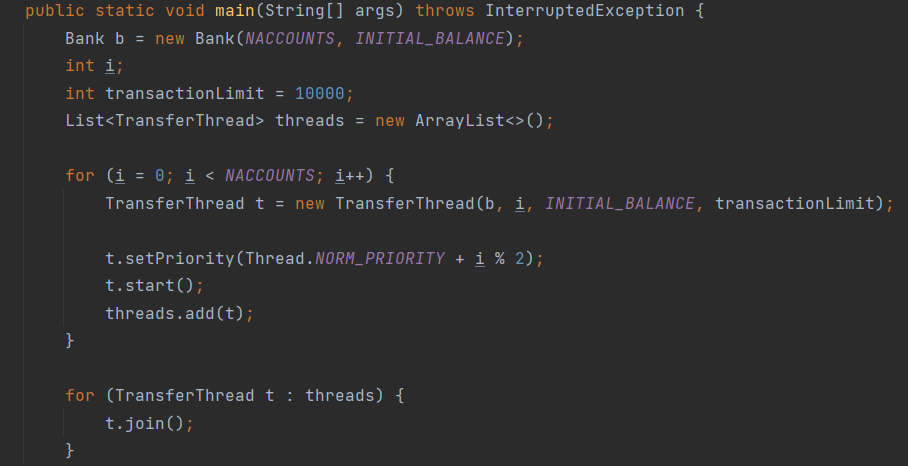


2 спосіб – Lock/ReentrantLock

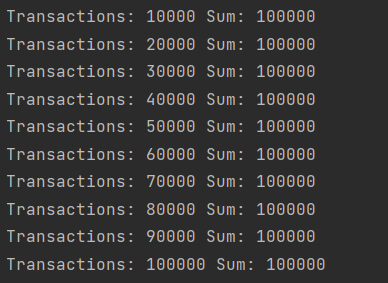
Використовується пакет java.util.concurrent.locks, який надає можливості для синхронізації доступу до ресурсів між потоками. Зокрема, використовується клас ReentrantLock, який є реалізацією інтерфейсу Lock. Lock використовується для створення блокувань, щоб гарантувати, що тільки один потік може отримати доступ до коду, захищеного блокуванням, в один момент часу. Це дозволяє забезпечити правильну синхронізацію між потоками та запобігти race conditions і помилкам у багатопотоковому середовищі. Блокування дозволяє потокам отримувати доступ до критичної секції коду послідовно, уникнувши некоректних конкурентних ситуацій та забезпечивши безпеку та правильність виконання операцій.



Також в main треба додати метод join(). Метод join() блокує виконання головного потоку до тих пір, поки кожен потік зі списку threads не завершить своє виконання. Це забезпечує, що всі потоки будуть виконані перед продовженням виконання головного потоку.



Таким чином, використання методу join() дозволяє дочекатись завершення всіх потоків перед тим, як програма завершиться повністю. Це важливо, щоб переконатись, що всі транзакції були оброблені перед виведенням результатів або подальшою обробкою даних.



3 спосіб – синхронізований блок

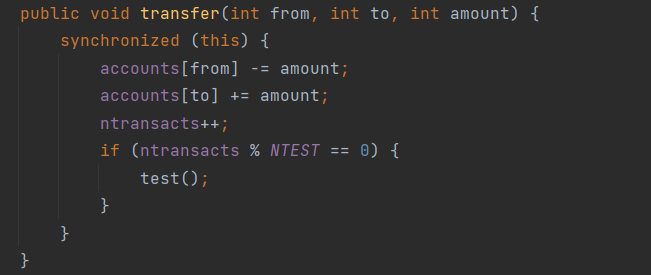
Різниця між синхронізованим методом і синхронізованим блоком в тому, яка частина коду забезпечується синхронізацією.

Синхронізований метод: У цьому випадку весь метод помічений ключовим словом synchronized. Це означає, що доступ до цього методу буде синхронізованим для всіх потоків, які викликають цей метод. Коли один потік викликає синхронізований метод, інші потоки повинні очікувати, доки перший потік не завершить виконання методу. Це забезпечує синхронізований доступ до ресурсів, які контролюються методом.

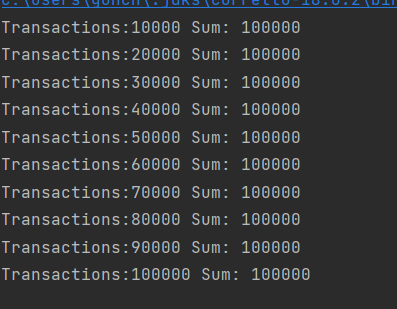
Синхронізований блок: У цьому випадку ви використовуєте ключове слово synchronized разом з об'єктом або ресурсом, який ви хочете синхронізувати. Всередині синхронізованого блоку ви вказуєте об'єкт або ресурс, який буде використовуватись для блокування доступу до коду всередині блоку. Тільки один потік може виконувати код всередині синхронізованого блоку одночасно, інші потоки повинні чекати, доки перший потік не вийде з блоку.

Отже, основна різниця полягає в тому, яка частина коду забезпечується синхронізацією. Синхронізований метод синхронізує весь метод, тоді як синхронізований блок дозволяє синхронізувати лише певну частину коду. Вибір між цими двома підходами залежить від потреб вашої програми та специфічності ситуації, яку ви намагаєтесь вирішити.

Оскільки метод transfer маніпулює даними банку (зменшує баланс одного рахунку і збільшує баланс іншого), важливо забезпечити атомарність операцій і унікальний доступ до ресурсів для кожного потоку, який викликає цей метод.



Результат



**Завдання 2.**

Програма Producer-Consumer є одним з найпоширеніших способів координації роботи потоків. В ній є два типи потоків: виробник (Producer) і споживач (Consumer). Виробник виробляє дані (повідомлення), а споживач споживає ці дані. Обмін даними між потоками відбувається через спільний об'єкт (Drop).

Основна ідея полягає в тому, що виробник не може виробляти нові дані, якщо старі дані не були спожиті споживачем, і навпаки, споживач не може споживати нові дані, якщо виробник не виробив їх. Ця синхронізація забезпечується за допомогою використання "охоронних блоків" (guarded blocks) і методів wait() та notify().

Клас Drop - спільний об'єкт, що забезпечує комунікацію між класами Producer і Consumer.

Клас Consumer - отримує повідомлення з об'єкта Drop і виводить їх на екран.

Клас Producer - створює повідомлення і передає їх у об'єкт Drop для споживання.

Спочатку реалізуємо початковий приклад з лістингу

