Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 9

з дисципліни «Мова програмування Java»

Тема: «Concurrency and ProducerConsumer»

Виконала:

Студентка групи ІА-31

Соколова Поліна

Київ – 2025

Мета роботи – набуття практичних навичок у використанні високорівневих методів паралельного виконання та взаємодії потоків, взаємодія потоків на основі умовних змінних, використання пулу потоків, створення додатків з використанням фреймворку Fork/Join.

Task 1. Parallel algorithm and shared state

Переказ коштів

У Вас є клас Bank з методом transfer () для переказу грошей з одного рахунку на інший в межах банку. Напишіть реалізація тіла методу transfer(), яка могла б працювати в багатопотоковому середовищі. Під час переказу грошей з рахунку на рахунок, дані рахунки повинні блокуватися. Подумайте, як при цьому уникнути deadlock-ів.

Дочекайтеся закінчення переказів і підрахуйте скільки грошей є всього в банку (сума грошей на всіх рахунках). Сума грошей в банку до перекладів і після, повинні збігатися.

Task 2. Producer-consumer task

Реалізувати потокобезпечний кільцевий буфер та виконання наступних вимог:

1) П’ять потоків генерують рядки в кільцевий буфер. Рядок має формат: «Потік No ... згенерував повідомлення ...»

2) Інші два потоки перекладають рядки з першого кільцевого у другий кільцевий буфер. У другий кільцевий буфер записується рядок наступного формату: «Потік No ... переклав повідомлення ...».

3) Основний потік програми вичитує та роздруковує 100 повідомлень із другого буфера.

Потоки, описані у вимогах 1)-2) повинні бути потоками-демонами, інакше програма не завершить своє виконання.

Хід роботи

Завдання 1

У реалізованій програмі кожен банківський рахунок представлений класом Account, який містить баланс та методи поповнення і зняття коштів. Доступ до балансу синхронізований через synchronized, щоб унеможливити його зміну з кількох потоків одночасно. Проблема може виникнути, коли один потік хоче переказати гроші з одного рахунку на інший, а інший потік у цей самий момент намагається зробити протилежну операцію. Якщо при цьому блокувати рахунки у довільному порядку, система може потрапити в стан deadlock, коли два потоки чекають один на одного, і жоден не може продовжити виконання. Щоб цього уникнути, у методі transfer() рахунки завжди блокуються в залежності від їхніх ідентифікаторів - спочатку блокується об’єкт з меншим id, а потім з більшим. Використання id є зручним, тому що це незмінне поле, яке чітко визначає порядок блокування для будь-якої пари рахунків, незалежно від того, який потік виконує операцію.

Після переказу потрібно переконатись, що зберігається загальна сума грошей у банку після масового паралельного виконання транзакцій. Для цього створюється 200 рахунків з випадковими початковими балансами, після чого обчислюється загальна сума коштів до початку експерименту. Далі запускається великий пул потоків, у якому кожен потік виконує випадкову транзакцію: обирає два рахунки, генерує довільну суму, перевіряє, чи достатньо на рахунку коштів, і виконує переказ через метод transfer(). Після завершення всіх потоків повторно підраховується сума грошей на всіх рахунках. Якщо реалізація коректна, кінцева сума повинна збігатися з початковою.

Результат виконання:

```
Початкова сума: 1072707
Кінцева сума: 1072707
Успішно. Гроші не втрачено
```

Завдання 2

У програмі реалізовано модель взаємодії потоків за схемою виробник-споживач з використанням двох кільцевих буферів. Перший буфер приймає необроблені повідомлення від п’яти потоків-генераторів. Кожен виробник працює у нескінченному циклі та формує рядок із зазначенням номера потоку і порядкового номера повідомлення. При спробі запису у переповнений буфер потік автоматично переходить у стан очікування за допомогою wait(), доки інший потік не звільнить місце.

Другий буфер використовується для зберігання вже оброблених рядків. Два потоки-перекладачі постійно вичитують повідомлення з першого буфера, додають власний форматований опис та записують результат у другий. Якщо перший буфер тимчасово порожній, перекладач переходить у режим очікування, доки не з’являться нові дані. Аналогічно, якщо другий буфер заповнений, запис блокується до звільнення місця. Синхронізація обох буферів забезпечена через монітор synchronized, механізми wait() та notifyAll(), що дозволяє уникнути активного циклічного чекання та забезпечити коректну взаємодію кількох потоків.

Основний потік програми виконує роль кінцевого споживача. Він читає 100 уже перекладених повідомлень із другого буфера й виводить на консоль. Як тільки основний потік завершує свою роботу, програма завершується повністю, оскільки всі допоміжні робочі потоки є потоками-демонами й припиняють виконання автоматично.

Виведення:

```
Producer 1: Потік № 1 згенерував повідомлення 25
Main: Потік № 2 переклав повідомлення  Потік № 1 згенерував повідомлення 20
Producer 5: Потік № 5 згенерував повідомлення 22
Translator 1: Потік № 1 переклав повідомлення  Потік № 4 згенерував повідомлення 20
Main: Потік № 1 переклав повідомлення  Потік № 4 згенерував повідомлення 20
Producer 2: Потік № 2 згенерував повідомлення 26
Translator 2: Потік № 2 переклав повідомлення  Потік № 2 згенерував повідомлення 21
Main: Потік № 2 переклав повідомлення  Потік № 2 згенерував повідомлення 21
Producer 3: Потік № 3 згенерував повідомлення 23
Main: Потік № 1 переклав повідомлення  Потік № 5 згенерував повідомлення 19
Translator 1: Потік № 1 переклав повідомлення  Потік № 5 згенерував повідомлення 19
Producer 1: Потік № 1 згенерував повідомлення 26
Main: Потік № 2 переклав повідомлення  Потік № 1 згенерував повідомлення 21
```

Висновок: у результаті виконання лабораторної роботи я реалізувала паралельний переказ коштів, забезпечивши правильну синхронізацію потоків і усунувши можливість deadlock-ів. Тестування з тисячами транзакцій підтвердило, що загальна сума грошей у системі зберігається, а отже, метод transfer() працює коректно в умовах багатопотокового виконання.
Також реалізовано багатопотокову систему з використанням потокобезпечного кільцевого буфера. Завдання дозволило закріпити на практиці синхронізацію потоків, організацію взаємодії через спільні структури даних і застосування механізму wait()/notifyAll() і продемонструвало, як налаштовується паралельна обробка даних між незалежними потоками.

Посилання на GitHub репозиторій:
https://github.com/sokolovapolina230/Java_labs/tree/main