Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Комп’ютерний практикум №1**

з дисципліни «Технології паралельних обчислень»

Тема: «Розробка потоків та дослідження пріоритету запуску потоків»

|  |  |
| --- | --- |
| **Виконав:**  студент групи ІТ-04  Коновальчук Андрій Володимирович  Дата здачі \_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_ | **Перевірила**:  асистент кафедри ІПІ  Дифучина Олександра Юріївна |

Київ 2023

**Хід роботи:**

***Завдання 1:*** Реалізуйте програму імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці (див. презентацію «Створення та запуск потоків в java» та приклад). Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм.

*Опис пункту:*

Використовуючи приклад програми імітації руху більярдних кульок (“By Cay S. Horstman”) створюємо рішення з паралельним рухом сірих кульок по “полотні” вікна, де кульки генеруються у випадкових точках цього полотна (подальші скріншоти результатів містять імплементацію пунктів 1-4). Рух кульок в кожному відкритому потоці реєструється в консолі.

*Приклад роботи:*

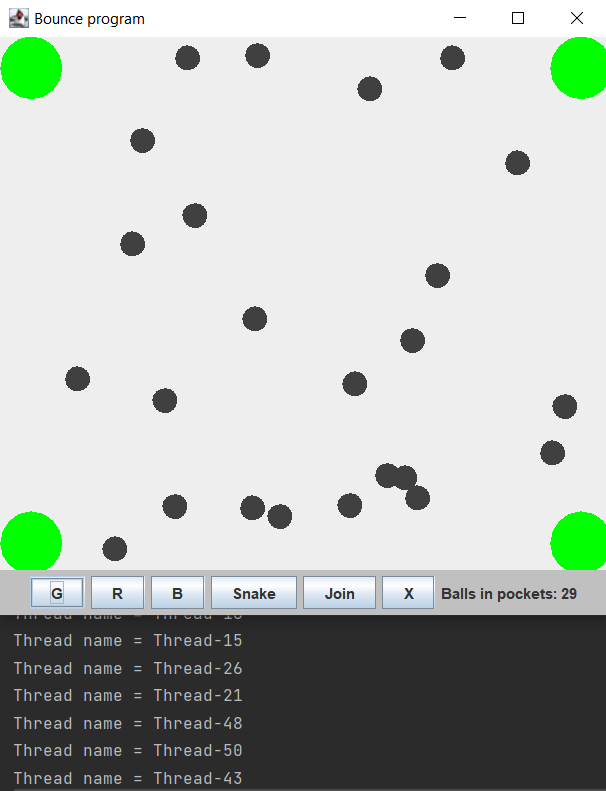


Рисунок 1 – Рух сірих кульок, потоки

*Спостереження та висновки:*

При відсутності “луз”, котрі стирають кульки з полотна, та великій кількості ітерацій (часу життя кульок), чим більша кількість кульок, тим повільніший їх рух.

Це спричинено великою кількістю відкритих потоків, котрі використовують та розподіляють між собою ресурси процесора та пам’яті. Чим більше кульок, тим більше ресурсів витрачається.

Переваги потоків та потокової архітектури:

1. Потоки можна використовувати для звільнення основного потоку. Дозволяють правильно і максимально ефективно використовувати ресурси багатоядерних процесорів.
2. Потоки можна використовувати для розбиття завдання на менші блоки, які можна виконувати одночасно (забезпечення паралельності).

***Завдання 2:*** Модифікуйте програму так, щоб при потраплянні в «лузу» кульки зникали, а відповідний потік завершував свою роботу. Кількість кульок, яка потрапила в «лузу», має динамічно відображатись у текстовому полі інтерфейсу програми.

*Опис пункту:*

До попереднього рішення додаємо клас “лузи” зі своїми параметрами. Лузи для зручності та достатньої тривалості вимальовуємо по кутках полотна. Для реєстрації попадання використовується формула відстані між двома точками і радіуси кульок. Завершення роботи потоку через переривання (“return”) методу “run” класу “Thread”. Лічильник зроблений через клас “Score” і “прослуховування подій” – попадання у лузу.

*Приклад роботи:*

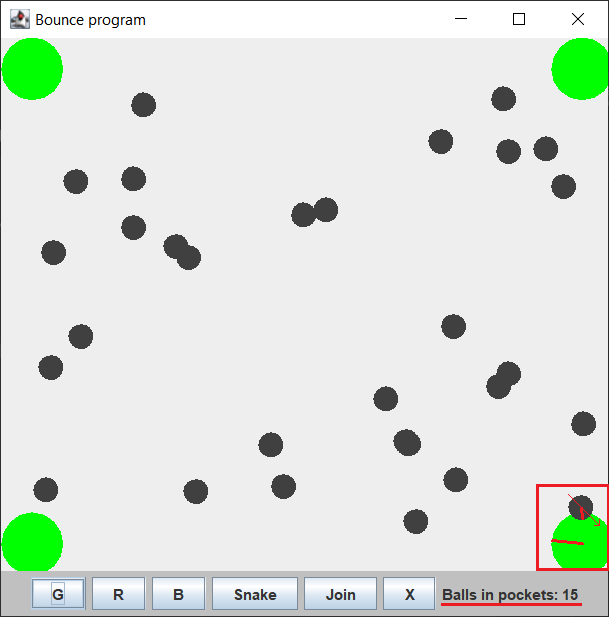


Рисунок 2 – Лузи та лічильник

***Завдання 3:*** Виконайте дослідження параметру priority потоку. Для цього модифікуйте програму «Більярдна кулька» так, щоб кульки червоного кольору створювались з вищим пріоритетом потоку, в якому вони виконують рух, ніж кульки синього кольору. Спостерігайте рух червоних та синіх кульок при збільшенні загальної кількості кульок. Проведіть такий експеримент. Створіть багато кульок синього кольору (з низьким пріоритетом) і одну червоного кольору, які починають рух в одному й тому ж самому місці більярдного стола, в одному й тому ж самому напрямку та з однаковою швидкістю. Спостерігайте рух кульки з більшим пріоритетом. Повторіть експеримент кілька разів, значно збільшуючи кожного разу кількість кульок синього кольору. Зробіть висновки про вплив пріоритету потоку на його роботу в залежності від загальної кількості потоків.

*Опис пункту:*

Створимо конструктор з можливістю передавати параметр кольору в клас “Ball”, а також метод-обгортку для зміни пріоритету потоку після створення нової кульки і відповідні кнопки для відповідних кульок в інтерфейсі програми.

Для експерименту створимо генерацію 800 кульок синього кольору (малий пріоритет) та 1-єї кульки червоного кольору (високий пріоритет). Запускатись кульки будуть з однієї точки (де буде найбільша кількість відскакувань), червона кулька генерується останньою.

*Приклад роботи:*

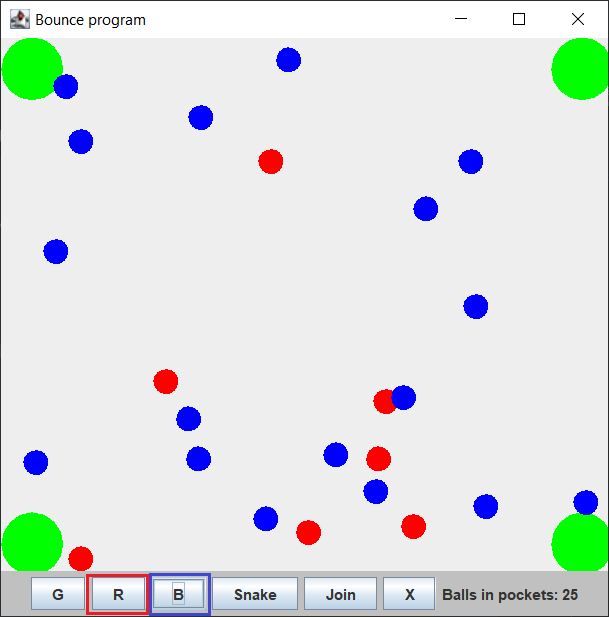


Рисунок 3 – Червоні та сині кульки

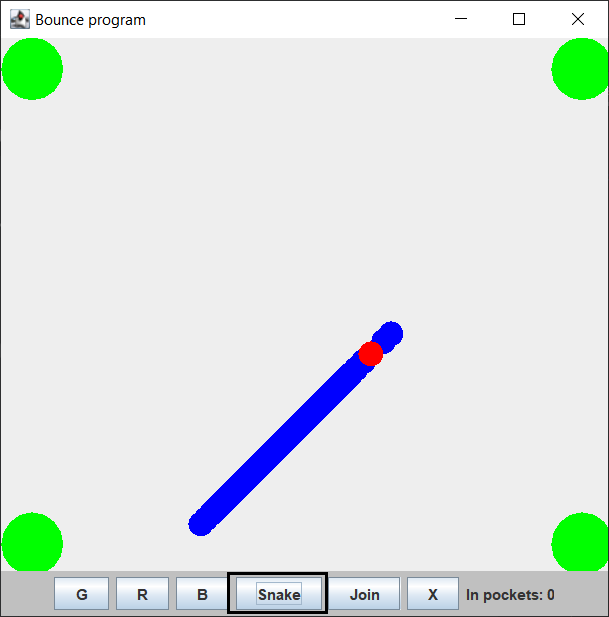


Рисунок 4 – “Змійка” з кульок

*Спостереження та висновки:*

Через хаотичний рух кульок, видимої різниці у роботі потоків з вищим пріоритетом ми не бачимо.

При експерименті зі “змійкою” бачимо як червона кулька із хвоста повільно і без зупинок рухається до голови, попри те, що задана швидкість кульок однакова.

Пояснити це можна тим, що через найвищий пріоритет у червоної кульки, вона завжди обробляється однією з перших. Чим більша кількість кульок, тим помітніший це результат, через велику кількість потоків з низьким пріоритетом.

***Завдання 4:*** Побудуйте ілюстрацію для методу join() класу Thread з використанням руху більярдних кульок різного кольору. Поясніть результат, який спостерігається

*Опис пункту:*

Кульки будуть генеруватися почергово синього та червоного кольорів. Запускатимуться з однієї точки, зі зменшеним числом ітерацій та однаковим пріоритетом. Основою буде використання методу “join()” класу “Thread”.

*Приклад роботи:*

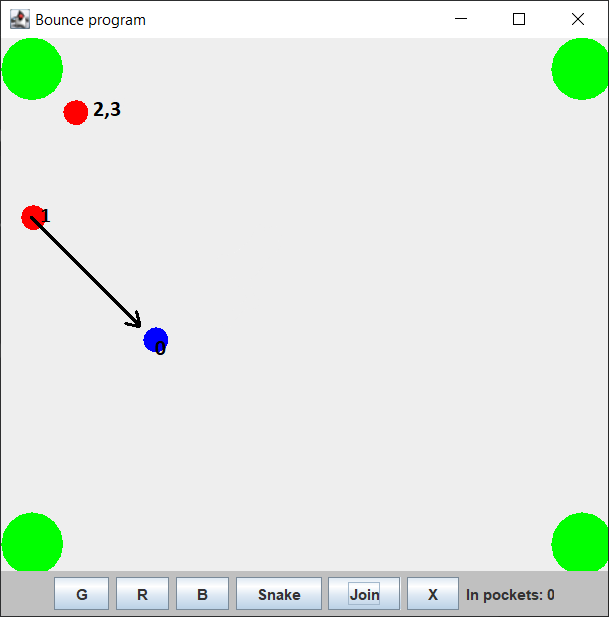


Рисунок 5 – Експеримент join

*Спостереження та висновки:*

Усі 4 кульки спершу вимальовуються, потім о одній летять в одному напрямку і зупиняються в одній точці. Кулька одного одного кольору запускається зразу після зупинки кульки іншого кольору.

Це тому що коли ми викликаємо метод join() для потоку, визваний потік переходить у стан очікування. Він залишається в стані очікування, доки не завершиться потік, на який посилається.

***Завдання 5:*** Створіть два потоки, один з яких виводить на консоль символ ‘-‘, а інший – символ ‘|’. Запустіть потоки в основній програмі так, щоб вони виводили свої символи в рядок. Виведіть на консоль 100 таких рядків. Поясніть виведений результат. Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів.

*Опис пункту:*

Для виконання цієї задачі необхідно розробити два класи: клас символьного потоку і клас контрольованого потоку. В першій частині завдання ми просто створюємо та запускаємо два потоки. В другій частині ми створюємо спільний для двох потоків об’єкт, котрий передаємо у конструктор. Використовуючи синхронний метод або блок, а також спільний об’єкт, вимальовуємо потрібний нам символ. Потім, використовуючи методи “wait()” та “notify()”, відповідно переводимо у очікування один потік та сповіщаємо інший про продовження роботи.

*Приклад роботи:*

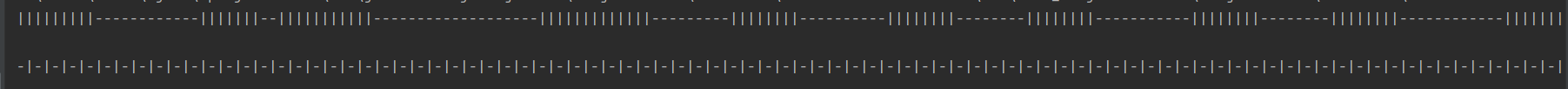


Рисунок 6 – Неконтрольований та контрольований потік

*Приклад коду:*



Рисунок 7 – Клас контрольованого потоку

*Спостереження та висновки:*

Результат першої частини завдання – чергування групок різної кількості символів “-” та “|”. Результат другої частини завдання – почерговий вивід символів “-” та “|”.

Пояснюється це тим, що якщо не контролювати роботу потоків, то на потоки навіть однакового пріоритету буде виділятися до різному ресурси процесора. Контролюючи потоки, ми сповіщаємо їх про роботу інших потоків, і тому отримуємо бажаний результат.

***Завдання 6:*** Створіть клас Counter з методами increment() та decrement(), які збільшують та зменшують значення лічильника відповідно. Створіть два потоки, один з яких збільшує 100000 разів значення лічильника, а інший – зменшує 100000 разів значення лічильника. Запустіть потоки на одночасне виконання. Спостерігайте останнє значення лічильника. Поясніть результат. Використовуючи синхронізований доступ, добийтесь правильної роботи лічильника при одночасній роботі з ним двох і більше потоків. Опрацюйте використання таких способів синхронізації: синхронізований метод, синхронізований блок, блокування об’єкта. Порівняйте способи синхронізації.

*Опис пункту:*

Для виконання завдання створимо 3 набори методів increment() та decrement(): звичайний, із синхронним методом та блоком. Потім створюємо, запускаємо та join-имо їх як і до цього в практикумі.

Що до синхронізації блокуванням, то тут використовуємо звичайний набір increment() та decrement(). Проте синхронізація відбувається з використанням об’єкта класу ReentrantLock та його методів. Отже, операція відбувається наступним чином: виконання методу “lock” об’єкта класу ReentrantLock , потім методу “increment” об’єкта класу Counter і потім “unlock” класу ReentrantLock.

*Приклад роботи:*

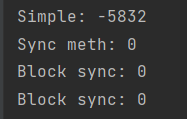


Рисунок 8 – Результати роботи різних методів

*Приклад коду:*

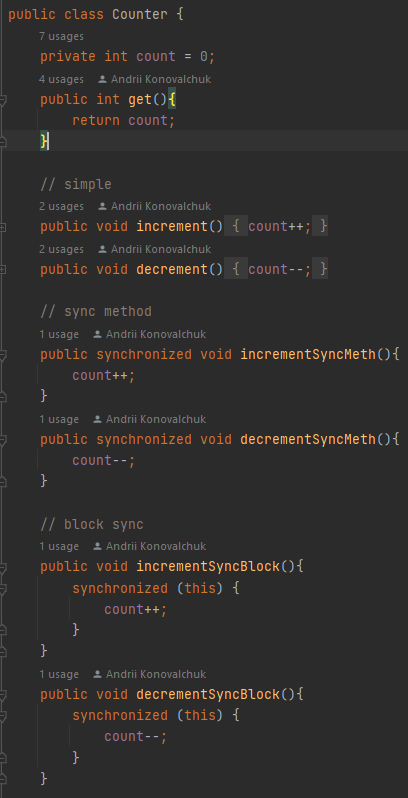


Рисунок 9 – клас Counter з методами increment() та decrement()



Рисунок 10 – Синхронізація блокуванням об’єкту

*Спостереження та висновки:*

Результат роботи звичайних методів increment() та decrement() дають незадовільний результат і кожного разу різний. Решта методів з використанням синхронізації дають задовільний результат – 0.

Говорячи про перший результат, з аналогією на попередній пункт (“----|||-|||||--”), потоки опрацьовують задачу з різними ресурсами і швидкістю. Так як об’єкт у них один, і він обробляється одночасно двома потоками з неоднаковою швидкістю, то результат викривлюється, попри однакову кількість ітерацій. Використовуючи синхронізацію ми виправляємо цей дефект.

**Висновок:**

В ході виконання практикуму ми навчилися створювати, користуватися та керувати потоками в Java. Було виконано, продемонстровано, описано та пояснено 6 експериментів. Результатами експериментів є візуалізація паралелізму, а також вирішення поставлених проблем шляхом паралельного програмування.

У звіті наявні скріншоти з прикладами результатів роботи коду, а також опис роботи коду та пояснення результатів. Вперше використано такі класи, інтерфейси, методи та ключові слова: Thread, Runnable, ReentrantLock, sleep, join, notify, wait, lock, unlock, synchronized.

**Лістинг коду у вигляді GitHub-репозиторію:**

<https://github.com/m4cy43/parallel_programming>