Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

## Лабораторна робота № 1 з дисципліни «ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ»

Тема: «\_ Створення та запуск потоків: імітація руху більярдних кульок. Найпростіші засоби управління потоками.\_»

Виконав: Перевірив:

студент групи ІТ-02 Дифучин В. В.

Терешкович Максим.

Дата здачі \_\_\_\_\_\_\_\_\_\_\_\_

Захищено з балом \_\_\_\_\_\_\_\_\_\_\_

Київ 2023

**5 Завдання до комп’ютерного практикуму**

**5.1 Завдання до комп’ютерного практикуму 1 «Розробка потоків та дослідження пріоритету запуску потоків»**

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. **10 балів.**
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. **10 балів.**
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. **20 балів.**
4. Побудуйте ілюстрацію для методу join() класу Thread з використанням руху більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**
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. **10 балів.** Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів. **15 балів.**
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. **10 балів.** 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. **15 балів.**

**Виконання.**

1. Реалізуйте програму імітації руху більярдних кульок, в якій рух кожноїкульки 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. 10 балів.

При збільшенні кількості кульок програма працює так, що кількість кульок може бути більше ніж фізичних ядер у процесору, тому кульки можуть рухатися повільніше, так як потоки чередуються та цим самим сповільнюють програму.

Потокова архітектура програм - це підхід до проектування програм, в якому програма розділяється на окремі функціональні блоки, називані потоками, які взаємодіють між собою, передаючи дані через канали.

Кожен потік виконує конкретну задачу і може бути запущений паралельно з іншими потоками. Канали використовуються для обміну даними між потоками. Як правило, ці дані передаються у вигляді потоку байтів або повідомлень.

Така архітектура дозволяє ефективно використовувати багатоядерні процесори та забезпечує підвищення продуктивності програми за рахунок паралельної обробки даних. Вона широко використовується в програмуванні мережевих додатків, систем реального часу, баз даних та інших високопродуктивних застосувань.

*Ефективність*: Потокова архітектура дозволяє виконувати операції над даними в реальному часі. Це дозволяє програмі забезпечувати швидку і ефективну обробку даних.

*Паралелізм*: У потоковій архітектурі можна одночасно виконувати декілька операцій над різними частинами даних. Це забезпечує можливість використання паралельних обчислень і прискорює роботу програми.

*Масштабованість*: Потокова архітектура дозволяє легко масштабувати програму для роботи з більшими об'ємами даних. Це забезпечує можливість роботи з великими обсягами даних, що відповідає вимогам сучасних додатків.

*Надійність*: У потоковій архітектурі програми, як правило, зменшується кількість багів і помилок. Це пов'язано з тим, що кожна операція виконується окремо і перевіряється на валідність даних.

*Гнучкість*: Потокова архітектура дозволяє легко змінювати послідовність операцій для відповідності вимогам різних проектів. Це забезпечує можливість швидко адаптувати програму до нових вимог користувачів.

*Модульність*: Потокова архітектура програми дозволяє використовувати різні модулі для виконання різних операцій. Це забезпечує можливість розподілу функцій програми між різними командами розробників, що полегшує розробку та підтримку програмного забезпечення.

*Підтримка стрімкого потоку даних*: Потокова архітектура програм дозволяє обробляти дані в режимі реального часу, що робить її ідеальною для задач, пов'язаних з обробкою потоку даних, таких як аналіз великих даних, розпізнавання мови, обробка відео та звуку.

*Розширюваність*: У потоковій архітектурі додавання нових операцій для обробки даних не потребує повної переробки програми. Це забезпечує можливість легко додавати нові функції та функціонал до програми.

*Відсутність блокування*: У потоковій архітектурі програми кожен потік виконується незалежно від інших, що дозволяє уникнути блокування та збоїв, пов'язаних з одночасним доступом до даних.

*Висока продуктивність*: Потокова архітектура дозволяє використовувати різні види оптимізацій, що забезпечують високу продуктивність програми. Наприклад, можна використовувати кешування, оптимізувати пам'ять, використовувати мультипроцесорні системи та інші техніки для підвищення продуктивності.

1. 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. **10 балів.**

A picture containing chart

Description automatically generated

A picture containing text

Description automatically generated

1. 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. **20 балів.**

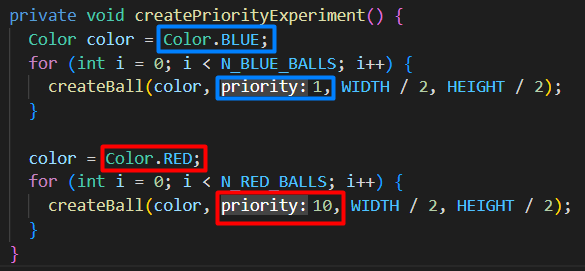
Text, whiteboard

Description automatically generated

Text

Description automatically generated with medium confidence

Ми спостерігаємо, що червона кулька випередила синю, хоча була створена пізніше. Це можливо завдяки вищому пріоритету червоної кульки, який призводить до пріоритетного опрацювання цього потоку та повільнішого виконання менш пріоритетних потоків. Ми провели експеримент на 100 синіх кульках та зафіксували це явище на моєму процесорі AMD Ryzen9 4000 series.

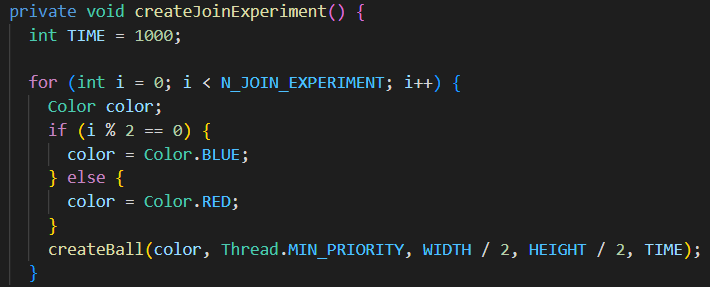


1. Побудуйте ілюстрацію для методу join() класу Thread з використанням руху більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**

Метод join() в мові Java, який викликається потоком A на потоці B, дозволяє потоку A зачекати, доки потік B не завершить своє виконання, перед тим як продовжити свою виконавчу діяльність. Протягом цього часу потік A буде зупинений і не буде виконуватись, поки потік B не закінчить своє виконання або не відбудеться переривання потоку. Якщо потік B вже завершив своє виконання, то виконання потоку A продовжиться нормально.

Метод join() може бути викликаний з параметром, який вказує на тайм-аут очікування. Якщо потік B не завершив своє виконання протягом вказаного тайм-ауту, то потік A продовжить свою виконавчу діяльність без очікування завершення потоку B.

Основне використання методу join() полягає в забезпеченні правильного порядку виконання потоків у програмі та уникненні неочікуваних помилок. Наприклад, якщо потрібно, щоб потік A почав виконуватись тільки після того, як потік B завершить своє виконання, то можна викликати метод join() на потоці B з потоку A.



1. 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. 10 балів. Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів. 15 балів.

Text

Description automatically generated

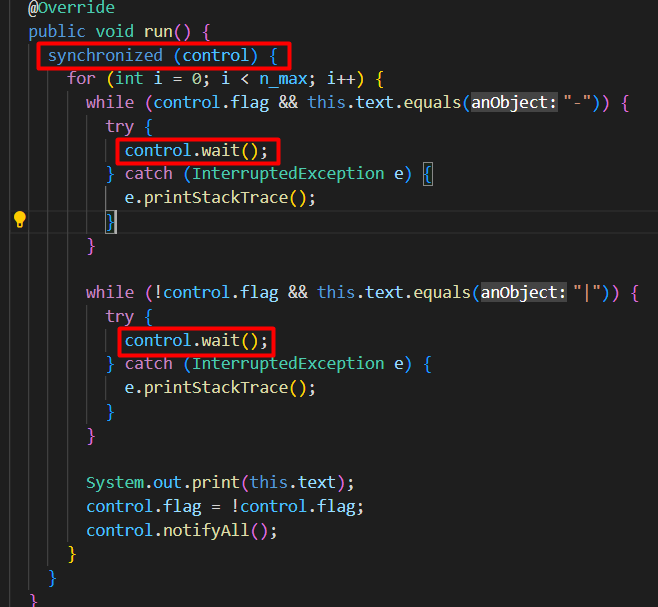
A picture containing calendar

Description automatically generated

Навіть якщо потоки виконуються паралельно, результат виконання програми може залежати від порядку виконання операцій, які конкурують за ресурси. Це називається змагальна умова (race condition) в Java.

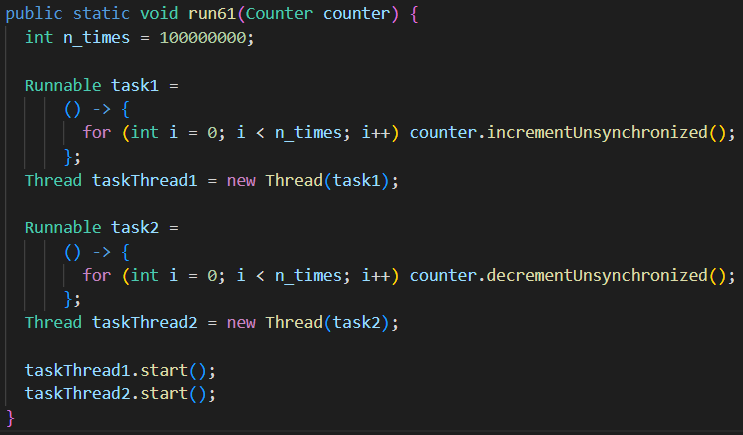
Змагальна умова може виникнути, коли декілька потоків намагаються звернутися до одного і того ж ресурсу, наприклад, спільного об'єкту або файла, і змінювати його стан. Якщо потоки не синхронізовані правильно, то вони можуть переписувати один одного і програма може поводитись неочікувано.

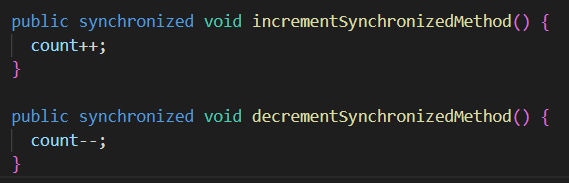
Щоб уникнути змагальної умови в Java, можна використовувати механізми синхронізації, такі як synchronized блоки або volatile змінні, щоб гарантувати, що тільки один потік може звертатися до ресурсу в один момент часу.

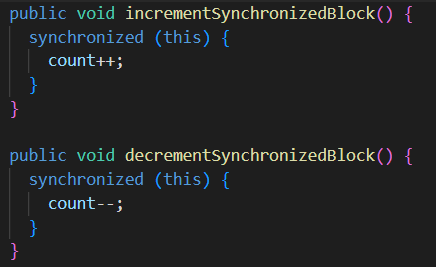
Щоб досягти синхронного виконання, можна використовувати метод "wait()" та "notify()", а також ключове слово "synchronized". Загалом, синхронізація дозволяє керувати порядком виконання потоків і забезпечити правильність результатів програми.

В мові Java ключове слово synchronized застосовується для забезпечення взаємовиключення між потоками, які одночасно працюють зі спільними змінними або ресурсами. Це ключове слово може бути застосовано до методу або блоку коду. Якщо до методу, то весь код методу буде виконуватись тільки в одному потоці за раз, таким чином, потік, який виконує метод, забезпечує виконання його коду без можливості одночасного доступу до нього з боку інших потоків. Якщо до блоку коду, то виконання цього блоку коду буде також забезпечено тільки в одному потоці за раз, тобто потік, який виконує блок коду, блокує доступ до нього інших потоків, щоб запобігти можливості виникнення конфлікту за ресурси.

1. 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. 10 балів.









Щоб забезпечити правильну роботу лічильника при одночасній роботі з ним двох і більше потоків, можна використовувати різні способи синхронізації, такі як синхронізований метод, синхронізований блок і блокування об'єкта.

У випадку синхронізованого методу, коли потік викликає синхронізований метод, він автоматично отримує блокування на об'єкті, що містить цей метод. Інші потоки не зможуть виконувати синхронізований метод до того моменту, поки блокування не буде знято.

У синхронізованому блоку коду блок коду захищений від одночасного виконання більше ніж одним потоком. Такі блоки використовуються для забезпечення взаємодії між потоками в многопотокових програмах та запобігання race condition.

Блокування об’єкту виконується за допомогою ключового слова synchronized. При виконанні закривається доступ до тих змінних які в ньому є. Закривається доступ до тих змінних які є в локері для всіх інших потоків крім того, що виконується зараз.

Кожен спосіб синхронізації має свої переваги та недоліки. Синхронізований метод зручний, якщо всі методи, які працюють з об'єктом, повинні бути виконані в одному потоці. Синхронізований блок коду зручний для тих випадків, коли не всі методи об'єкта потребують синхронізації. Блокування об'єкту можна використовувати, якщо необхідно синхронізувати доступ до кількох змінних одночасно.