Лабораторна робота №6

Багатопоточні програми

Мета роботи: *навчитись створювати багатопотокові програми.*

Завдання.

Обчислити значення визначеного інтеграла відповідно до варіанту. Реалізацію програми виконувати таким чином:

1.Створити клас “Функція” (з єдиним методом “обчислити”) для реалізації підінтегральної функції.

2.Створити клас “Обчислювач інтегралів”, який може працювати у багатопотоковому режимі і має метод “обчислити” з параметрами: a, b - кінці інтервалу, n - кількість кроків та f - підінтегральна функція.

3.Для цих класів розробити модульні тести і виконати тестування

4.Створити віконну програму, яка буде дозволяти вводити кількість інтервалів розбиття відрізку інтегрування і кількість потоків виконання.

5.Як результати роботи програми вивести обчислене значення інтегралу і час, який знадобився для її виконання.

6.Виконати обчислення декілька разів для різних (від 1 до 20 кількостей потоків виконання) при малій (менше 10^3) та великій (більше 10^6) кількості інтервалів розбиття відрізка.

7.Зробити висновки

Примітка. Формули для обчислення визначеного інтеграла наближеними методами наведено нижче:

- *Метод лівих прямокутників* $\int_a^b f\,(x)dx \approx \sum_{i=0}^{n-1} f(x_i)h$
- *Метод правих прямокутників* $\int_a^b f\,(x)dx \approx \sum_{i=1}^{n} f(x_i)h$
- *Метод середніх прямокутників* $\int_a^b f\,(x)dx \approx \sum_{i=0}^{n-1} f(x_i + h/2)h$
- *Метод трапецій* $\int_a^b f\,(x)dx \approx \frac{f(x_0)+f(x_n)}{2}h + \sum_{i=1}^{n-1} f\,(x_i)h$
- *Метод Сімпсона* $\int_a^b f\,(x)dx \approx \frac{h}{3}\left(\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f\,(x_i) + 2\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right) + \frac{1}{2}f(x_n)\right)$

в усіх методах $h = \frac{b-a}{n}, x_i = a + i \cdot h$

Варіанти завдання

№ вар	Інтеграл	Метод
1	$\int_1^9 3\sqrt{t}dt$	Метод лівих прямокутників
2	$\int_1^4 \frac{1+t}{\sqrt{2t}}dt$	Метод трапецій
3	$\int_1^9 3\sqrt{x}(1+\sqrt{x})dx$	Метод Сімпсона
4	$\int_0^1 ln(t+1)dt$	Метод правих прямокутників
5	$\int_1^{\sqrt{2}} \sqrt{2-x^2}dx$	Метод середніх прямокутників
6	$\int_1^2 \frac{e^x-1}{e^x+1}dx$	Метод Сімпсона
7	$\int_{\pi/6}^{\pi/3} \frac{dt}{sin^2(2t)}$	Метод лівих прямокутників
8	$\int_0^1 e^t\sqrt{e^t-1}dt$	Метод правих прямокутників
9	$\int_0^{\pi/3} cos(4t)cos(2t)dt$	Метод трапецій
10	$\int_0^{\pi/2} \sin(2t)\cos(3t)\,dx$	Метод середніх прямокутників

Короткі теоретичні відомості

Клас Thread і інтерфейс Runnable

Є два способи створити клас, екземплярами якого будуть потоки виконання: успадкувати клас від java.lang.Thread або реалізувати інтерфейс java.lang.Runnable. Цей інтерфейс має декларацію єдиного методу public void run(), який забезпечує послідовність дій при роботі потоку. При цьому клас Thread вже реалізує інтерфейс Runnable, але з порожньою реалізацією методу run(). Так що при створенні екземпляра Thread створюється потік, який нічого не робить. Тому в нащадку треба перевизначити метод run(). У нім слід написати реалізацію алгоритмів, які повинні виконуватися в даному потоці. Відзначимо, що після виконання методу run() потік припиняє існування – "вмирає".

Розглянемо перший варіант, коли ми успадковуємо клас від класу Thread, перевизначивши метод run().

Об'єкт-потік створюється за допомогою конструктора. Є декілька перевантажених варіантів конструкторів, найпростіший з них – з порожнім списком параметрів. Наприклад, в класі Thread їх заголовки виглядають так:

public Thread() – конструктор за умовчанням. Підпроцес отримує ім'я "system".

public Thread(String name) - потік отримує ім'я, що міститься в рядку name.

У класі-нащадку можна викликати конструктор за замовчуванням (без параметрів), або задати свої конструктори, використовуючи виклики конструкторів батьківських класів за допомогою виклику super(список параметрів). Із-за відсутності спадкування конструкторів в Java доводиться в спадкоємцеві заново задавати конструктори з тією ж сигнатурою, що і в класі Thread. Це є простою, але утомливою роботою. Саме тому зазвичай віддають перевагу способу завдання класу з реалізацією інтерфейсу Runnable, про що буде розказано далі.

Створення та запуск потоку здійснюється наступним чином:

```
public class T1 extends Thread{
   public void run(){
      ...
   }
   ...
}
Thread thread1 = new T1();
thread1.start();
```

Другий варіант – використання класу, в якому реалізований інтерфейс java.lang.Runnable. Цей інтерфейс, як вже говорилося, має єдиний метод public void run(). Реалізувавши його в класі, можна створити потік за допомогою перевантаженого варіанту конструктора Thread:

```
public class R1 implements Runnable{
   public void run(){
      ...
   }
   ...
}
Thread thread1 = Thread( new R1() );
thread1.start();
```

Зазвичай в такий спосіб користуються набагато частіше, оскільки в класі, що розробляється, не доводиться займатися дублюванням конструкторів класу Thread. Крім того, цей спосіб можна застосовувати у разі, коли вже є клас, що належить ієрархії, в якій базовим класом не є Thread або його спадкоємець, і ми хочемо використовувати цей клас для роботи усередині потоку. В результаті від цього класу ми отримуємо метод run(), у якому реалізований потрібний алгоритм, і цей метод працює усередині потоку типа Thread, що забезпечує необхідну поведінку в багатопотоковому середовищі. Проте в даному випадку ускладнюється доступ до методів з класу Thread – потрібне приведення типа.

Поля та методи, які задані в класі Thread

В класі Thread маємо декілька полів даних та методів, про які треба знати для роботи з потоками.

Найважніші константи та методи класа Thread:

•MIN_PRIORITY – мінімально можливий пріоритет потоків. (як описано у довідковій системі jdk, дорівнює 1)

•NORM_PRIORITY - нормальний пріоритет потоків. Головний потік створюється з нормальним пріоритетом, а потім пріоритет може бути змінено. (дорівнює 5)

•MAX_PRIORITY – максимально можливий пріоритет потоків. (як описано у довідковій системі jdk, дорівнює 10)

•static int activeCount() - вертає число активних потоків застосунку.

•static Thread currentThread() – вертає посилання на поточний потік.

•static boolean interrupted() – вертає стан статуса переривання поточного потоку, після чого встановлює його у значення false.

Найважливіші методи об'єктів типу Thread:

•void run() – метод, який забезпечує послідовність дій під час життя потоку. У класі Thread задана його порожня реалізація, тому в класі потоку він має бути перевизначений. Після виконання методу run() поток вмирає.

•void start() – викликає виконання поточного потоку, у тому числі запуск його методу run() у потрібному контексті. Може бути викликаний всього один раз.

•void setDaemon(boolean on) – у випадкуе on==true встановлює потоку статус демона, інакше – статус користувальницького потоку.

•boolean isDaemon() - вертає true у випадку, коли поточний потік є демоном.

•void yield() – "поступитися правами" – викликає тимчасове призупинення потоку, з передаванням прав іншим потокам виконувати необхідні для них дії.

•long getId() – повертає унікальний ідентифікатор потоку. Унікальність відноситься лише до часу життя потоку - після його завершення (смерті) даний ідентифікатор може бути привласнений іншому створюваному потоку.

•String getName() – повертає ім'я потоку, яке йому було задано при створенні або методом setName.

•void setName(String name) – встановлює нове ім'я потоку.

•int getPriority() - вертає пріоритет потоку.

•void setPriority(int newPriority) – встановлює пріоритет потоку.

•void checkAccess() – здійснення перевірки з поточного потоку на дозволеність доступу до іншого потоку. Якщо потік, з якого йде виклик, має право на доступ, метод не робить нічого, інакше – збуджує виключення SecurityException.

•String toString() – вертає рядкове представлення об'єкта потока, в тому числі – його им'я, групу, пріоритет.

•void sleep(long millis) – викликає призупинення ("засинання") потоку на millis мілісекунд. При цьому всі блокування (монітори) потоку зберігаються. Перевантажений варіант sleep(long millis,int nanos) - параметр nanos вказує число наносекунд. Дострокове пробудження здійснюється методом interrupt() – зі збудженням виключення InterruptedException.

•void interrupt() – перериває "сон" потоку, спричинений викликами wait(...) або sleep(...), встановлюючи йому статус перерваного (статус переривання=true). При цьому збуджується перевіряєма виключна ситуація InterruptedException.

•boolean isInterrupted() - вертає поточний стан статусу переривання потоку без зміни значення статусу.

•void join() – "злиття". Переводить потік в режим вмирання — очікування завершення. Це очікування – виконання метода join() - може відбуватися достатньо довго, якщо відповідний потік на момент виклику метода join() блокований. Тобто якщо в ньому виконується синхронізований метод або він очікує завершення синхронізованого метода. Перевантажений варіант join(long millis) - очікувати завершення потока протягом millis милісекунд. Виклик join(0) еквівалентний виклику join(). Зазвичай використовується для завершення головним потоком роботи всіх дочірніх потоків ("злиття" їх з головним потоком).

•boolean isAlive() - вертає true у випадку, коли поточний потік живий (ще не вмер). Відмітимо, що навіть якщо потік завершився, від нього залишається об'єкт - “привид”, який відповідає на запит isAlive() значенням false – тобто повідомляє, що об'єкт вмер.

Крім того, слід знати, що в класі Object визначені ще деякі методи, які можуть бути корисними для написання багатопотокової програми:

•public final void **wait**() throws InterruptedException — переводить потік в режим очікування доти, поки інший потік не викликає метод notify() або notyfyAll() для цього потоку

•public final void **wait**(long timeout) throws InterruptedException – те ж саме, що і попередній, але виконання буде автоматично продовжено після timeout мілісекунд

•public final void **wait**(long timeout, int nanos) throws InterruptedException - те ж саме, що і попередній, але виконання буде автоматично продовжено після timeout мілісекунд та nanos наносекунд

•public final void **notify**() - пробуджує потік, що перебуває в стані очікування після виклику wait()

•public final void **notifyAll**() - пробуджує всі потоки, що перебувають в очікування поточного монітору.

Слід зазначити, що всі провідні розробники процесорів перейшли на багатоядерну технологію. При цьому в одному корпусі процесора розташовано декілька процесорних ядер, здатних незалежно виконувати обчислення, але вони мають доступ до однієї і тієї ж загальної пам'яті. У зв'язку з цим програмування в багатопотоковому середовищі визнане найбільш перспективною моделлю паралелізування програм і стає одним з найважливіших напрямів розвитку програмних технологій. Модель багатопоточності Java дозволяє вельми елегантно реалізувати переваги багатоядерних процесорних систем. У багатьох випадках програми Java, написані з використанням багатопоточності, ефективно розпаралелюються автоматично на рівні віртуальної машини - без зміни не лише вихідного, але навіть скомпільованого байт-коду додатків.

Але програмування в багатопотоковому середовищі є складним і відповідальним заняттям, що вимагає дуже високої кваліфікації. Багато алгоритмів, що здаються простими, природними і надійними, в багатопотоковому середовищі опиняються непрацездатними. Через що способи вирішення навіть найпростіших завдань стають незвичайними і заплутаними, не кажучи вже про проблеми, що виникають при вирішенні складних завдань. У зв'язку з цим рекомендується на початковому етапі не захоплюватися багатопотоковими застосуваннями, а лише ознайомитися з даною технологією.