Вкладка 1

Отчет о лабораторной работе №6

Выполнил:
Петров Евгений Станиславович
Группа: 6204-010302D

Вкладка 2

ВВЕДЕНИЕ

Цель: Разработать приложение, формирующее в одном потоке вычислений набор заданий для интегрирования, а во втором потоке – вычисляющее значения интегралов функций.

В ходе выполнения работы необходимо расширить библиотеку функций численным интегратором, реализовать последовательную и многопоточную генерацию и обработку заданий, обеспечить корректный обмен данными между потоками, корректную обработку прерываний, а также подготовить демонстрационный запуск реализованных решений.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1. *Метод трапеций*
   Метод трапеций основан на разбиении отрезка интегрирования на подотрезки длиной *h*. Площадь под графиком функции аппроксимируется суммой площадей трапеций и вычисляется по формуле:
   $\sum (f(x_i) + f(x_{i+1})) / 2 \cdot \Delta x$.
   Если длина последнего подотрезка меньше шага *h*, он интегрируется с использованием собственной ширины.
2. *Область определения функции*
   Численное интегрирование корректно только внутри области определения функции. Попытка интегрирования за пределами домена должна приводить к ошибке или исключению.
3. *Потоки и разделяемое состояние*
   Одновременный доступ нескольких потоков к общему объекту без синхронизации может приводить к гонкам данных, несогласованным чтениям и исключениям времени выполнения, таким как `NullPointerException`. Использование синхронизированных блоков или семафоров позволяет исключить одновременную запись и чтение разделяемых данных.
4. *Одноместный семафор*
   Одноместный семафор позволяет различать стадии «запись задания» и «чтение задания». Это предотвращает потерю данных и перемешивание параметров задания при передаче между потоками генерации и интегрирования.

МЕТОДИКА И РЕАЛИЗАЦИЯ

Среда выполнения и структура проекта

Язык реализации — Java. Рекомендуемая версия JDK — 17 и выше.

Структура проекта включает следующие элементы:

- пакет `functions` — базовые и табулированные функции, а также численный интегратор `Functions.integrate`;
- пакет `functions.basic` — элементарные функции (экспонента, логарифм, тригонометрические функции);
- пакет `functions.meta` — композиции и арифметические операции над функциями;
- пакет `threads` — классы обмена заданиями между потоками (`Task`, генераторы, интеграторы, семафор);
- класс `Main` — сценарии проверки заданий 1–4.

Реализация по этапам

1. *Численное интегрирование (Задание 1)*
   Реализован метод `Functions.integrate(Function f, double left, double right, double step)`, который проверяет корректность шага и принадлежность границ области определения функции. Интегрирование выполняется методом трапеций с учётом неполного последнего шага. В методе `Main.integrationCheck` вычисляется интеграл функции exp(x) на отрезке [0; 1] с подбором шага, обеспечивающего точность не хуже $1 \cdot 10^{-7}$.
2. *Последовательная версия (Задание 2)*
   Метод `Main.nonThread` создаёт объект `Task` с заданным количеством задач. Генерируются логарифмические функции со случайным основанием от 1 до 10, случайные границы интегрирования из диапазонов [0; 100] и [100; 200], а также шаг интегрирования из интервала (0; 1]. Для каждой задачи выводятся строки вида `Source ...` и `Result ...`, после чего интеграл вычисляется численным методом.
3. *Простая потоковая модель (Задание 3)*
   Поток `SimpleGenerator` реализует интерфейс `Runnable`, заполняет объект `Task` и выводит исходные данные задачи. Поток `SimpleIntegrator` считывает снимки состояния `Task`, ожидая появления корректных данных, и вычисляет результат

интегрирования.
Для предотвращения смешивания полей используется синхронизация внутри методов `Task.update` и `Task.snapshot`.

4. *Семафорная модель и обработка прерываний (Задание 4)*
Реализован одноместный семафор `OnePlaceSemaphore`, обеспечивающий корректное чередование операций записи и чтения.
Потоки `Generator` и `Integrator` наследуются от `Thread`, используют семафор для передачи задания, корректно обрабатывают вызов `interrupt()` и выполняют короткие паузы (`sleep(1)`) для наглядной демонстрации чередования выполнения.
Метод `Main.complicatedThreads` запускает пару потоков, ожидает 50 мс и затем завершает их с помощью прерывания.

РЕЗУЛЬТАТЫ

В результате выполнения лабораторной работы:

- реализован численный интегратор методом трапеций с проверкой области определения функции;
- последовательный сценарий корректно генерирует и решает не менее 100 логарифмических интегралов;
- потоковая версия с синхронизацией устраняет `NullPointerException` и несогласованные состояния данных;
- семафорная модель предотвращает потерю заданий и корректно обрабатывает прерывания потоков;
- продемонстрирована сходимость метода трапеций на примере интеграла exp(x) на отрезке [0; 1] с точностью лучше $1 \cdot 10^{-7}$.

РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Ниже приведены результаты работы всех четырёх режимов программы. Для заданий с большим объёмом вывода консольный лог обрезан до первых и последних пяти задач.

Задание 1. Проверка численного интегрирования

Результаты проверки метода трапеций на примере интегрирования функции exp(x) на отрезке [0; 1]:

```
exp(x) integral on [0;1] with step 0.01:

1,71829615 (theoretical 1,71828183)


Step to reach 1e-7 precision:

step = 0,00048828125

result = 1,71828186
```

Полученное значение интеграла сходится к теоретическому значению e − 1 с заданной точностью.

Задание 2. Последовательный режим (nonThread)

Первые пять задач:

```
Source 24,0821 103,7523 0,626389

Result 24,0821 103,7523 0,626389 178,042844


Source 91,1349 188,6409 0,936709

Result 91,1349 188,6409 0,936709 284,126986


Source 23,3924 164,8174 0,582542

Result 23,3924 164,8174 0,582542 275,293712


Source 99,5109 197,3187 0,874799
```

```
Result 99,5109 197,3187 0,874799 472,458983

Source 95,0781 159,8511 0,012046
Result 95,0781 159,8511 0,012046 153,096414
```

Последние пять задач:

```
Source 47,5859 105,5713 0,945200
Result 47,5859 105,5713 0,945200 182,421549

Source 81,2912 168,6619 0,501637
Result 81,2912 168,6619 0,501637 457,964239
```

(Всего последовательно обработано более 100 заданий.)

Задание 3. Потоки с синхронизацией (simpleThreads)

Первые пять задач (фрагмент):

```
Source 73,9530 102,9702 0,413366
Result 73,9530 102,9702 0,413366 61,190203

Source 4,7426 123,8633 0,082372
Result 4,7426 123,8633 0,082372 272,190486

Source 21,5804 162,4639 0,059915
```

```
Result 21,5804 162,4639 0,059915 328,698462


Source 30,6724 125,7831 0,168532
Result 30,6724 125,7831 0,168532 245,871234


Source 19,3749 187,0195 0,244430
Result 19,3749 187,0195 0,244430 660,321366
```

Последние пять задач:

```
Source 3,1317 118,8800 0,080785
Result 3,1317 118,8800 0,080785 214,201123


Source 44,0173 120,9497 0,747688
Result 44,0173 120,9497 0,747688 308,286281
```

Синхронизация обеспечивает корректное чередование генерации и интегрирования без потери данных и исключений.

Задание 4. Потоки с семафором и обработкой прерываний (complicatedThreads)

Первые пять задач:

```
[complicatedThreads] start


Source 25,6037 187,7877 0,064304
```

```
Result 25,6037 187,7877 0,064304 849,520697

Source 5,0275 157,9924 0,899985
Result 5,0275 157,9924 0,899985 402,245390

Source 98,8261 133,7850 0,713591
Result 98,8261 133,7850 0,713591 76,236109

Source 99,9249 115,4032 0,520255
Result 99,9249 115,4032 0,520255 64,157584

Source 90,1316 132,2397 0,635284
Result 90,1316 132,2397 0,635284 129,144446
```

Последние пять задач (фрагмент завершения):

```
Source 74,4086 178,9048 0,500603
Result 74,4086 178,9048 0,500603 275,422690

Source 75,0900 131,2637 0,916755
User program finished
Result 75,0900 131,2637 0,916755 163,166527

Source 78,5043 105,5026 0,323269
```

```
Result 78,5043 105,5026 0,323269 109,244595


[complicatedThreads] finished (threads interrupted and
joined)
```

ИТОГ

Представленные результаты подтверждают корректную работу:

- численного интегратора;
- последовательного алгоритма;
- многопоточной модели с синхронизацией;
- многопоточной модели с семафором и обработкой прерываний.

Формат вывода соответствует требованиям задания, а обрезка логов позволяет наглядно представить работу программы без избыточного объёма данных.

ВЫВОДЫ

Поставленные цели лабораторной работы достигнуты. Реализован численный интегратор и исследованы три модели взаимодействия потоков: без синхронизации, с использованием синхронизации и с применением семафора. Использование синхронизации и семафоров позволило устранить гонки данных и потери заданий. Метод трапеций продемонстрировал ожидаемую сходимость при уменьшении шага интегрирования.

ТЕСТИРОВАНИЕ И ПРОВЕРКА

Функциональная проверка выполняется с помощью сценариев `integrationCheck`, `nonThread`, `simpleThreads` и `complicatedThreads`, реализованных в классе `Main`.

В коде предусмотрены проверки граничных условий: положительность шага интегрирования, корректность границ области определения функции и защита от передачи `null`-функций.

Рекомендуется после установки JDK выполнить компиляцию и запуск проекта и убедиться, что консольный вывод содержит ожидаемые пары строк `Source` и `Result`.

КЛЮЧЕВЫЕ КЛАССЫ

- `functions.Functions.integrate` — численное интегрирование методом трапеций;
- `threads.Task` — контейнер параметров задания;
- `threads.SimpleGenerator`, `threads.SimpleIntegrator` — базовая потоковая модель;
- `threads.OnePlaceSemaphore`, `threads.Generator`, `threads.Integrator` — семафорная модель с корректным завершением по прерыванию;
- `Main` — точка входа и сценарии демонстрации заданий 1–4.

СПИСОК ЛИТЕРАТУРЫ

1. Документация Oracle по многопоточности в Java. Java Concurrency Tutorial.