Практическая работа №8

Время выполнения	@December 1, 2025 → December 12, 2025
Статус	Не начато

🕶️ Задание: «Сеть кибер-детекторов»

🎮 Сюжет

Мегаполис охвачен вирусными ИИ. Для победы над ними вы разрабатываете всё более сложную сеть кибер-детекторов. Сначала это простые независимые дроны, но затем они объединяются в синхронизированные группы, подключаются к Центральному Узлу и начинают работать как распределённая вычислительная система.

📝 Этапы выполнения

🔷 Этап 1. **Ручные потоки: первые дроны**

- Сгенерируйте массив чисел (100 000 элементов).
- Разбейте его на `N` равных сегментов (например, 20).
- Создайте отдельный поток (`Thread` / `Runnable`) для каждого сегмента.
- Потоки считают сумму сегмента и выводят лог:

```
[Детектор-#1] Сектор 0–4999 просканирован. Сумма = 123456
```

🔷 Этап 2. **Синхронизация: общий отчёт**

- Добавьте общую переменную для накопления глобальной суммы.
- Сделайте корректный доступ с помощью:
  - `synchronized` блока **ИЛИ**
  - `AtomicLong`.
- После завершения всех потоков выведите:

```
[Центральный ИИ] Общая сумма данных = XXXXX
```

🔷 Этап 3. **ExecutorService: подключение к Центральному Узлу**

- Переделайте систему: детекторы работают как задачи `Callable<Result>`.
- Запускайте их через `ExecutorService`.
- Используйте `Future.get()`, чтобы собрать частичные результаты.
- Сложите их в главном потоке.

🔷 Этап 4. **Аномалии**

- Каждый поток считает не только сумму, но и количество чисел выше **9000** (аномалии).
- Верните результат через объект:

```
class Result {
    long sum;
```

```
    int anomalies;
}
```

- Выведите отчёт:

```
[Центральный ИИ] Общая сумма = XXXXX
[Центральный ИИ] Найдено аномалий = YY
```

🔷 Этап 5. **Обработка по мере готовности (CompletionService)**

- Используйте `ExecutorCompletionService<Result>`, чтобы результаты обрабатывались **сразу после выполнения задачи**, а не в конце.
- Логи должны появляться динамически, например:

```
[Центральный ИИ] Получен отчёт от Детектора-#3: сумма = 11234, аномалий = 2
```

🔷 Этап 6. **Координация фаз (CyclicBarrier)**

- Используйте `CyclicBarrier`, чтобы все детекторы завершили сканирование **первого уровня** (например, 5000 элементов).
- Только после этого они переходят ко второму уровню анализа.
- Лог:

```
[ИИ-координатор] Все детекторы завершили фазу 1. Начинаем фазу 2.
```

🔷 Этап 7. **Сравнение производительности**

- Добавьте замеры времени для разных реализаций:
  - Ручные потоки,
  - ExecutorService,
  - CompletionService.
- Выведите итоговую таблицу:

```
[ИИ-Аналитик] Время выполнения:
- Ручные потоки: 350 мс
- ExecutorService: 210 мс
- CompletionService: 190 мс
```

📌 Пример выполнения программы

```
=== Этап 1: Ручные потоки ===
[Детектор-1] Сектор 0–4999 просканирован. Сумма = 24754270
[Детектор-10] Сектор 45000–49999 просканирован. Сумма = 25012690
[Детектор-9] Сектор 40000–44999 просканирован. Сумма = 25384111
[Детектор-7] Сектор 30000–34999 просканирован. Сумма = 24700888
[Детектор-8] Сектор 35000–39999 просканирован. Сумма = 25054760
[Детектор-6] Сектор 25000–29999 просканирован. Сумма = 25214104
[Детектор-5] Сектор 20000–24999 просканирован. Сумма = 25143900
[Детектор-4] Сектор 15000–19999 просканирован. Сумма = 24951264
[Детектор-3] Сектор 10000–14999 просканирован. Сумма = 25302383
[Детектор-2] Сектор 5000–9999 просканирован. Сумма = 25286743
[Центральный ИИ] Общая сумма данных = 250805113
⏱️ Время (ручные потоки): 17 мс
```

```
=== Этап 2-3: ExecutorService ===
[Детектор-1] Сектор 0–4999 завершён. Сумма = 24754270, Аномалий = 493
[Детектор-9] Сектор 40000–44999 завершён. Сумма = 25384111, Аномалий = 524
[Детектор-8] Сектор 35000–39999 завершён. Сумма = 25054760, Аномалий = 493
[Детектор-7] Сектор 30000–34999 завершён. Сумма = 24700888, Аномалий = 479
[Детектор-6] Сектор 25000–29999 завершён. Сумма = 25214104, Аномалий = 506
[Детектор-5] Сектор 20000–24999 завершён. Сумма = 25143900, Аномалий = 507
[Детектор-3] Сектор 10000–14999 завершён. Сумма = 25302383, Аномалий = 478
[Детектор-4] Сектор 15000–19999 завершён. Сумма = 24951264, Аномалий = 479
[Детектор-2] Сектор 5000–9999 завершён. Сумма = 25286743, Аномалий = 535
[Детектор-10] Сектор 45000–49999 завершён. Сумма = 25012690, Аномалий = 530
[Центральный ИИ] Общая сумма = 250805113
[Центральный ИИ] Найдено аномалий = 5024
⏱ Время (ExecutorService): 7 мс

=== Этап 5: CompletionService ===
[Центральный ИИ] Получен отчёт: сумма = 25302383, аномалий = 478
[Центральный ИИ] Получен отчёт: сумма = 24700888, аномалий = 479
[Центральный ИИ] Получен отчёт: сумма = 25012690, аномалий = 530
[Центральный ИИ] Получен отчёт: сумма = 25143900, аномалий = 507
[Центральный ИИ] Получен отчёт: сумма = 25214104, аномалий = 506
[Центральный ИИ] Получен отчёт: сумма = 24951264, аномалий = 479
[Центральный ИИ] Получен отчёт: сумма = 25384111, аномалий = 524
[Центральный ИИ] Получен отчёт: сумма = 24754270, аномалий = 493
[Центральный ИИ] Получен отчёт: сумма = 25054760, аномалий = 493
[Центральный ИИ] Получен отчёт: сумма = 25286743, аномалий = 535
[Центральный ИИ] Итоговая сумма = 250805113
[Центральный ИИ] Общие аномалии = 5024
⏱ Время (CompletionService): 277 мс

=== Этап 6: CyclicBarrier ===
[Детектор-2] Фаза 1 завершена. Сумма = 12693724
[Детектор-5] Фаза 1 завершена. Сумма = 12516893
[Детектор-4] Фаза 1 завершена. Сумма = 12446494
[Детектор-3] Фаза 1 завершена. Сумма = 12741109
[Детектор-1] Фаза 1 завершена. Сумма = 12363565
[Детектор-10] Фаза 1 завершена. Сумма = 12463260
[Детектор-9] Фаза 1 завершена. Сумма = 12788502
[Детектор-8] Фаза 1 завершена. Сумма = 12518594
[Детектор-7] Фаза 1 завершена. Сумма = 12416160
[Детектор-6] Фаза 1 завершена. Сумма = 12599588
[ИИ-координатор] Все детекторы завершили фазу. Переход к следующей.
[Детектор-6] Фаза 2 завершена. Сумма = 12614516
[Детектор-3] Фаза 2 завершена. Сумма = 12561274
[Детектор-5] Фаза 2 завершена. Сумма = 12627007
[Детектор-4] Фаза 2 завершена. Сумма = 12504770
[Детектор-2] Фаза 2 завершена. Сумма = 12593019
[Детектор-7] Фаза 2 завершена. Сумма = 12284728
[Детектор-8] Фаза 2 завершена. Сумма = 12536166
[Детектор-9] Фаза 2 завершена. Сумма = 12595609
[Детектор-10] Фаза 2 завершена. Сумма = 12549430
[Детектор-1] Фаза 2 завершена. Сумма = 12390705
```

🎯 Дополнительные миссии

🔷 1. **Сканирование в реальном времени**

- Вместо статического массива поток должен каждые 100 мс получать новые данные (например, генерировать случайные числа) и пересчитывать аномалии.

- Реализовать остановку сканирования по команде оператора (например, через ввод в консоль).

🔷 2. **Детекторы-специалисты**

- Сделать два вида потоков:
  - Одни считают суммы,
  - Другие ищут аномалии.
- Они работают параллельно над одним и тем же сегментом.

🔷 3. **Приоритетные детекторы**

- Использовать `Thread.setPriority()` или `PriorityBlockingQueue` для имитации детекторов разных «уровней доступа».
- Проверить, как это влияет на порядок выполнения задач.

🔷 4. **Многослойное сканирование**

- Каждое число массива сначала проверяется на «аномалию» (>9000),
- потом на «коррупцию данных» (например, делимость на 13),
- потом на «вирусный след» (палиндромное число).
- Потоки должны возвращать несколько метрик в `Result`.

📚 Контрольные вопросы

1. В чём разница между процессами и потоками?
2. Какие есть способы создания потоков в Java? Чем `Thread` отличается от `Runnable`?
3. Для чего нужен ключевое слово `synchronized`? Какие его недостатки?
4. Чем `AtomicLong` отличается от обычной переменной `long` при работе в многопоточности?
5. Что такое `ExecutorService`? Какие преимущества он даёт по сравнению с ручным созданием потоков?
6. Чем задачи `Runnable` отличаются от `Callable`?
7. Зачем нужен метод `Future.get()`? Что произойдёт, если его не вызывать?
8. В чём отличие работы с `ExecutorService` и `CompletionService`?
9. Почему при использовании `CompletionService` результаты могут приходить в разном порядке?
10. Как можно реализовать тайм-аут для задач и зачем он нужен?
11. Для чего используется `CyclicBarrier`? Чем он отличается от `CountDownLatch`?
12. Что произойдёт, если один поток «зависнет» и не дойдёт до барьера?
13. В каких сценариях `CyclicBarrier` удобнее, чем другие средства синхронизации?
14. Почему ExecutorService работает быстрее, чем ручные потоки?
15. Как влияет количество потоков на производительность программы?
16. Что может произойти, если потоков будет слишком много?