Отчет по лабораторной работе №3 (Реализация параллельного движения на перекрестке)

Студент: Гладков Игорь
Группа: 5130201/20102

Описание

В данной работе необходимо создать многопоточное приложение, моделирующее работу интеллектуального светофора на перекрестке. Приложение должно эмулировать движение машин, подъезжающих к перекрестку, с учетом их направления и синхронизации.

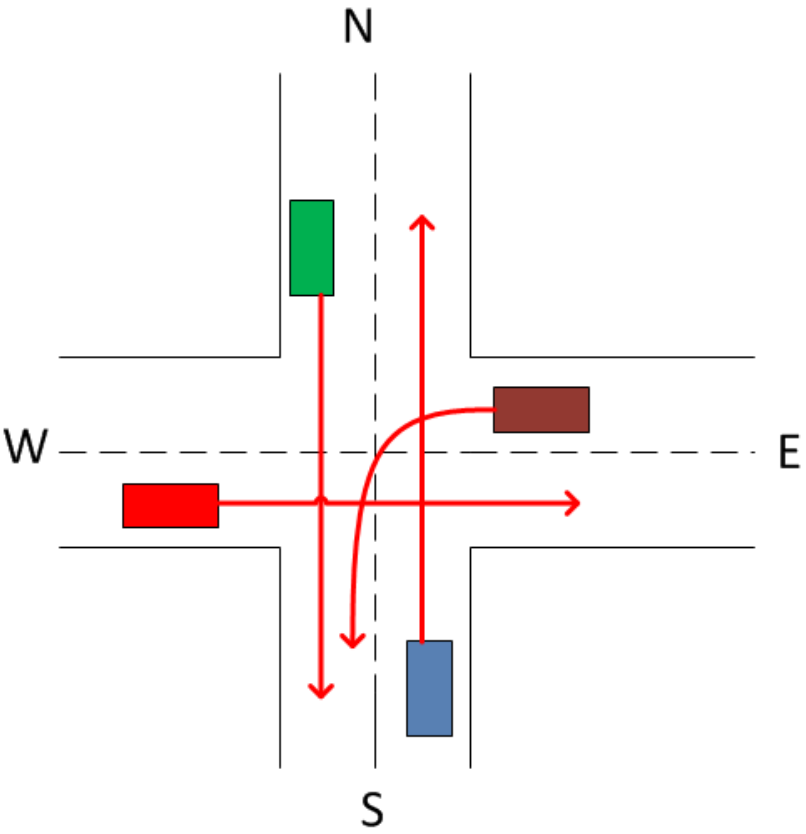
Светофор контролирует движение машин, обеспечивая параллельное движение автомобилей, чьи траектории не пересекаются. Если траектории автомобилей пересекаются, они двигаются поочередно.

Модель

Перекресток состоит из четырёх направлений:
- north(север)
- west (запад)
- south (юг)
- east (восток)

Каждое направление может принимать автомобили, которые встают в очередь перед светофором. Приложение отслеживает возможные пересечения траекторий машин и синхронизирует их движение.

Машины могут двигаться в одном из следующих направлений относительно себя:
- Straight (прямо)
- Left (налево)
- Right (направо)



Светофор реализует синхронизацию потоков с использованием механизма CyclicBarrier, который обеспечивает, что все потоки будут готовы к выполнению перед началом движения. Каждый поток (автомобиль) может двигаться только тогда,

когда это безопасно и не нарушает синхронизации с другими машинами. Например, автомобили, движущиеся с направлений north-straight(зеленая машина) и south-straight(синяя машина), могут двигаться одновременно, так как их пути не пересекаются, а west-straight(красная), east-left(коричневая) - поочередно.

Основные компоненты

- **Автомобили (потоки)**: Каждый поток представляет машину, которая подъезжает к перекрестку. У каждого потока есть два параметра: направление движения и сторона, с которой он подъехал.
- **Светофор**: Контролирует, какие автомобили могут двигаться одновременно, и синхронизирует потоки с помощью CyclicBarrier.
- MultiValueMap: Структура данных для хранения направлений движения машин и их возможных параллельных направлений.

Принципы работы

1. **Синхронизация потоков**: Каждый поток (машина) перед движением синхронизируется с другими потоками, чтобы избежать конфликтов на перекрестке.
2. **Параллельное движение**: Машины могут двигаться параллельно, если их траектории не пересекаются.
3. **Очередность движения**: Если траектории машин пересекаются, они едут поочередно.

Особенности реализации

Класс TrafficLight

Класс TrafficLight отвечает за управление потоками автомобилей и синхронизацию их работы на перекрёстке.

Поля:

- *ConcurrentMap<String, MultiValueMap<Integer, Integer>> threadData* : Потокобезопасная карта, которая хранит данные о возможных направлениях движения каждого потока.
- *int count_threads* : Количество потоков (автомобилей), участвующих в работе светофора.
- *List<String> activeThreads* : Список активных потоков, которые находятся в движении.
- *CyclicBarrier barrier* : Барьер синхронизации для управления параллельным выполнением потоков.

Методы:

- *static void changeBarrier(int count_threads)* : Меняет количество потоков и создаёт новый объект CyclicBarrier.
- *static void startCars(Thread t1, Thread t2, Thread t3, Thread t4)* : Запускает переданные потоки автомобилей.

Класс Car

Класс Car моделирует автомобиль, его начальное положение, направление движения и логику движения на перекрёстке.

Поля:

- *List<String> direction:* Статический список возможных направлений движения (straight, right, left).
- *List<String> from:* Статический список сторон света (south, north, west, east).
- *Integer fromIndex:* Индекс стороны света, откуда движется автомобиль.
- *Integer directionIndex:* Индекс направления движения автомобиля.
- *MultiValueMap<Integer, Integer> canRide:* Карта, описывающая, с каких сторон и в какие направления автомобиль может двигаться параллельно с другими.

Методы:

- Конструктор *Car(String from, String direction):* Инициализирует сторону движения (from) и направление движения (direction).
  Определяет индексы стороны и направления, вычисляет возможные пересекающиеся траектории с помощью метода canRide().

- *static void setDirection():* Инициализирует список направлений движения.
- *static void setFrom():* Инициализирует список сторон света.
- *static Integer leftFrom(Integer fromIndex):* Возвращает индекс стороны света, находящейся слева от текущей.
- *static Integer rightFrom(Integer fromIndex):* Возвращает индекс стороны света, находящейся справа от текущей.
- *static Integer reverseFrom(Integer fromIndex):* Возвращает индекс противоположной стороны света.
- *void canRide():* Определяет, с каких сторон и в какие направления автомобиль может двигаться параллельно с другими.
- *static int findIndex(List<String> array, String value):* Возвращает индекс заданного значения в списке.
- *boolean canRideParallel(List<String> activeThreads):* Проверяет возможность параллельного движения автомобиля с текущими активными потоками.
- Метод *run():* Добавляет данные текущего потока в threadData. Управляет движением потока в зависимости от состояния других потоков и барьера.

Результаты работы программы

При создании четырех машин с примера, а именно: *north-straight*, *south-straight*, *_west-straight и _east-left*, программа должна выполняться за три шага, так как всего два потока потока могут выполниться параллельно, остальные два будут выполняться поочередно.

Программа выдает корректный результат:

```
    Шаг выполнения: 1
south-straight
north-straight

    Шаг выполнения: 2
west-straight

    Шаг выполнения: 3
east-left
```

Тестирование

Для проверки корректности реализации класса MultiValueMap были созданы тесты:

- Тест *testDeadlocks* :
  Проверяет наличие дедлоков.
  - Создаются потоки с автомобилями, которые движутся из разных направлений и имеют различные направления движения.
  - Потоки запускаются через метод TrafficLight.startCars().
  - Устанавливается тайм-аут 5 секунд: если за это время все потоки завершаются, считается, что дедлоков нет.
  - Если потоки не завершаются за установленное время, тест завершается с ошибкой fail("Deadlock detected").
- Тест testRaceConditions
  Проверяет наличие условий гонки.
  - Потоки запускаются и ждут завершения методом join().
  - После завершения потоков их состояние проверяется: каждый поток должен быть в состоянии TERMINATED.
  - Если бы условия гонки были, это могло бы привести к некорректному завершению потоков или неконсистентным данным.

Результаты тестов

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.project.TrafficLightTest
```

```
        Шаг выполнения: 1
west-straight

        Шаг выполнения: 2
south-straight
north-straight

        Шаг выполнения: 3
east-left
east-left
north-straight

        Шаг выполнения: 1
west-straight
south-straight
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.141 s -- in
com.project.TrafficLightTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.784 s
[INFO] Finished at: 2025-01-24T20:24:51+03:00
[INFO] ------------------------------------------------------------------------
```

Заключение

В ходе выполнения лабораторной работы была разработана многопоточная программа, моделирующая работу интеллектуального светофора на перекрестке. Реализация обеспечила корректную синхронизацию потоков, предотвращение конфликтов и возможность параллельного движения машин с учётом их траекторий.

Исходный код

Ссылка на репозиторий: https://github.com/ukQueen/Java_Lab3

Файлы

example.png	11,4 КБ	2025-01-24	Игорь Гладков