МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №3  
по дисциплине «Корпоративные информационные системы»

Тема: «События»

Вариант №15

Выполнил: Дронов А. А.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc183377068)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc183377069)

[ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ 6](#_Toc183377070)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 7](#_Toc183377071)

[КОД ПРОГРАММЫ 8](#_Toc183377072)

[ВЫВОД 15](#_Toc183377073)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать систему взаимодействующих классов: класс, хранения графа с указанием расстояния для каждого ребра и операций перемещения объектов по этому графу; класс, рассчитывающий скорость перемещения каждого объекта; класс, рассчитывающий среднюю скорость перемещения всех объектов. Классы расчёта, должны срабатывать при каждом добавлении операции перемещения. В расчёте скоростей должны учитываться времена задержки объектов в вершинах графа.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В лабораторной работе реализован граф, моделирующий перемещение объектов по узлам с использованием событийного механизма CDI. Основная цель — отслеживание перемещений объектов, расчет их скорости и средней скорости по графу с использованием событий, передаваемых через аннотации и наблюдатели.

Класс Graph является основным компонентом, управляющим узлами, ребрами, объектами и их операциями. Узлы (Node) представляют точки графа, а ребра (Edge) — соединения между ними с указанным расстоянием. Объекты, перемещающиеся по графу, представлены классом MovingObject, а операции их перемещения — MovingObjectOperation. Операции включают начальный и конечный узлы, задержку в часах и расстояние между узлами.

В классе Graph реализован метод addOperation, который добавляет операцию перемещения для объекта. Если объект впервые появляется в графе, ему устанавливается начальное местоположение. Для каждой операции проверяется наличие соответствующего ребра, после чего объект перемещается. Операция добавляется в список операций для объекта, а событие с обновленным списком операций отправляется через CDI.

Механизм обработки событий представлен двумя классами: AverageSpeedCalculator и ObjectSpeedCalculator. Первый рассчитывает среднюю скорость всех объектов, исходя из общего пройденного пути и затраченного времени. Второй определяет индивидуальные скорости для каждого объекта, основываясь на их операциях перемещения. Оба класса используют аннотацию @AddedOperation для получения событий, связанных с добавлением операций в граф.

Программа начинается с инициализации CDI-контейнера через библиотеку Weld. В граф добавляются узлы и ребра, создающие топологию сети. Затем создаются объекты MovingObject и операции их перемещения. Все операции добавляются в граф, что инициирует выполнение событийного механизма. После выполнения всех операций граф печатает свое состояние, включая текущие местоположения объектов и их операции.

# **ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ**

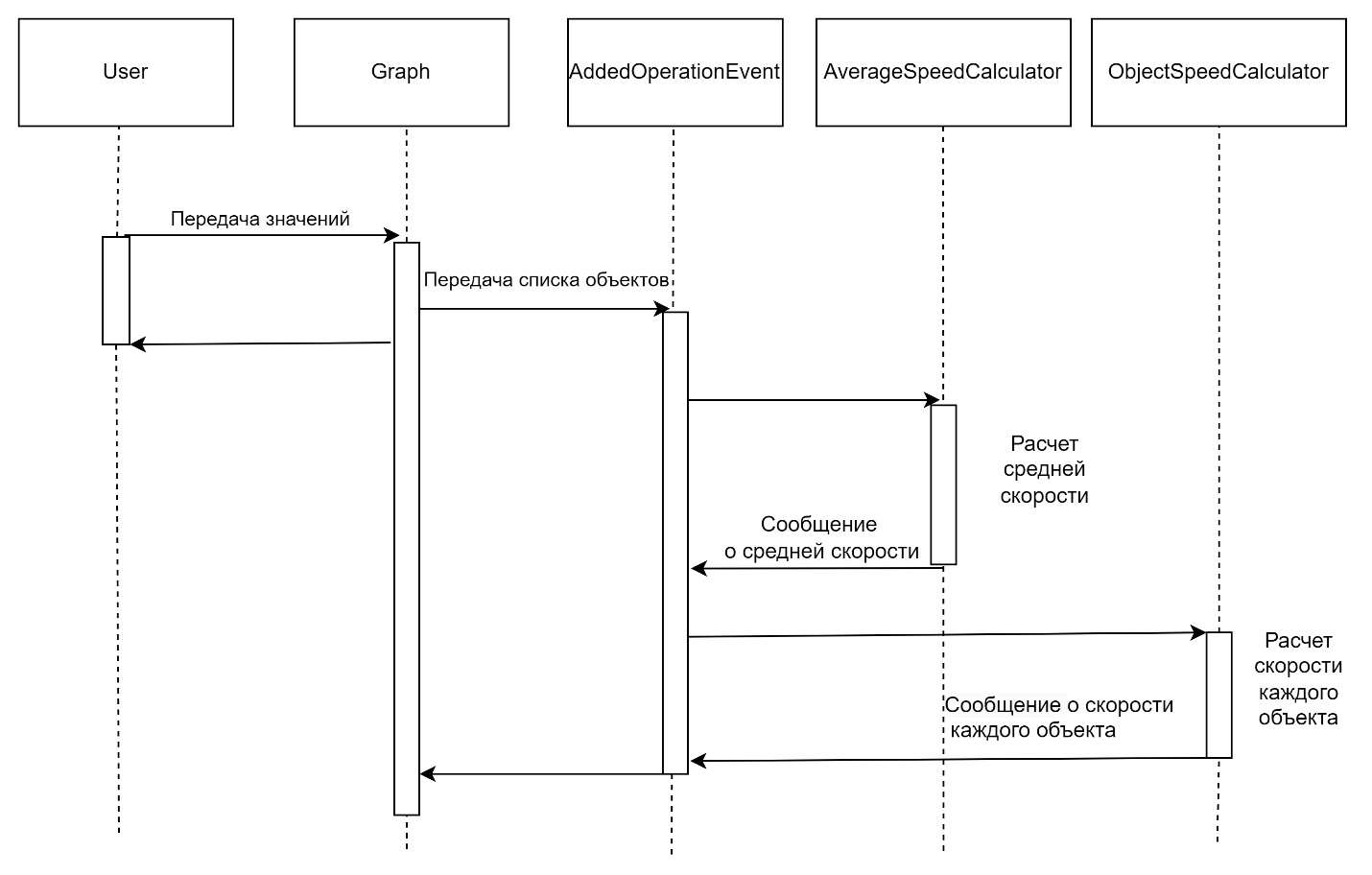


Рисунок 1 – Диаграмма последовательности

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

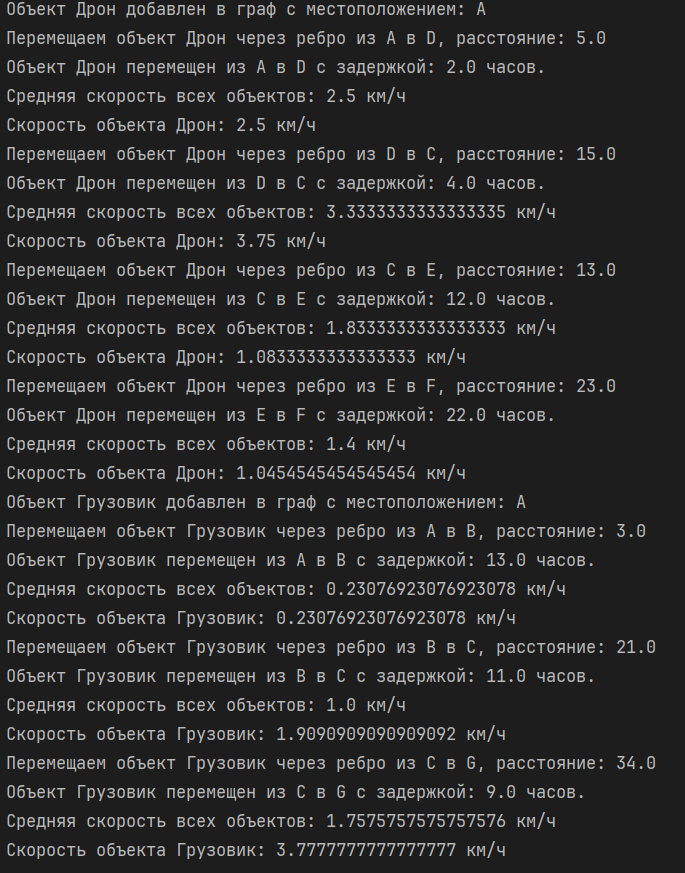


Рисунок 2 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface AddedOperation {  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Edge {  
 private Node from; // Начальный узел  
 private Node to; // Конечный узел  
 private double distance; // Расстояние между узлами  
}

package Lab3\_Event.Model;  
  
import Lab3\_Event.Annotation.AddedOperation;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.enterprise.event.Event;  
import javax.inject.Inject;  
import java.util.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Graph {  
 // Список узлов графа  
 private List<Node> nodes = new ArrayList<>(); // Инициализируем список узлов  
 // Список рёбер графа  
 private List<Edge> edges = new ArrayList<>(); // Инициализируем список рёбер  
 // Местоположения объектов в графе  
 private Map<MovingObject, Node> objectLocations = new HashMap<>();  
 // Операции для каждого объекта  
 private Map<MovingObject, List<MovingObjectOperation>> objectOperations = new HashMap<>();  
  
 @Inject  
 @AddedOperation  
 private Event<List<MovingObjectOperation>> operationEvent;  
  
 // Добавить узел в граф  
 public void addNode(Node node) {  
 if (!nodes.contains(node)) {  
 nodes.add(node);  
 }  
 }  
  
 // Добавить ребро в граф  
 public void addEdge(Node from, Node to, double distance) {  
 // Добавляем ребро от from к to  
 Edge edge1 = new Edge(from, to, distance);  
 edges.add(edge1);  
  
 // Добавляем ребро от to к from (для неориентированного графа)  
 Edge edge2 = new Edge(to, from, distance);  
 edges.add(edge2);  
 }  
  
 // Найти ребро на основе узлов from и to  
 public Edge findEdge(Node from, Node to) {  
 // Ищем ребро в обеих направлениях  
 return edges.stream()  
 .filter(edge -> (edge.getFrom().equals(from) && edge.getTo().equals(to)) ||  
 (edge.getFrom().equals(to) && edge.getTo().equals(from)))  
 .findFirst()  
 .orElseThrow(() -> new IllegalStateException("Ребро между узлами "  
 + from.getName() + " и "  
 + to.getName() + " не найдено."));  
 }  
  
 // Добавить операцию перемещения для объекта  
 public void addOperation(MovingObjectOperation operation) {  
 MovingObject object = operation.getMovingObject();  
 Node from = operation.getFrom();  
 Node to = operation.getTo();  
  
 // Проверка существования ребра и получение расстояния  
 Edge edge = findEdge(from, to);  
 operation.setDistance(edge.getDistance());  
  
 // Установить начальное местоположение объекта, если его нет в objectLocations  
 if (!objectLocations.containsKey(object)) {  
 objectLocations.put(object, from);  
 System.out.println("Объект " + object.getName() + " добавлен в граф с местоположением: " + from.getName());  
 }  
  
 // Добавить операцию в objectOperations  
 objectOperations.computeIfAbsent(object, k -> new ArrayList<>()).add(operation);  
  
 // Выполнить операции для объекта  
 executeLastOperation(object);  
  
 // Отправить событие с обновлённым списком операций  
 List<MovingObjectOperation> operations = objectOperations.get(object);  
 operationEvent.fire(operations); // Отправляем событие  
 }  
  
 // Переместить объект по операции  
 private void moveObject(MovingObjectOperation operation) {  
 MovingObject object = operation.getMovingObject();  
 Node from = operation.getFrom();  
 Node to = operation.getTo();  
  
 // Проверка текущего местоположения объекта  
 if (!objectLocations.get(object).equals(from)) {  
 throw new IllegalStateException("Объект не находится в ожидаемом месте.");  
 }  
  
 // Проверка существования ребра  
 Edge edge = findEdge(from, to);  
  
 // Лог перемещения объекта  
 System.out.println("Перемещаем объект " + object.getName() +  
 " через ребро из " + edge.getFrom().getName() +  
 " в " + edge.getTo().getName() + ", расстояние: " + edge.getDistance());  
  
 // Переместить объект  
 objectLocations.put(object, to);  
 }  
  
 // Выполнить только последнюю операцию для объекта  
 public void executeLastOperation(MovingObject object) {  
 if (!objectOperations.containsKey(object)) {  
 System.out.println("Для объекта не найдено операций: " + object.getName());  
 return;  
 }  
  
 List<MovingObjectOperation> operations = objectOperations.get(object);  
  
 if (operations.isEmpty()) {  
 System.out.println("Нет операций для объекта " + object.getName());  
 return;  
 }  
  
 // Получаем последнюю операцию  
 MovingObjectOperation lastOperation = operations.get(operations.size() - 1);  
 moveObject(lastOperation);  
 System.out.println("Объект " + object.getName() +  
 " перемещен из " + lastOperation.getFrom().getName() +  
 " в " + lastOperation.getTo().getName() +  
 " с задержкой: " + lastOperation.getDelayInHours() + " часов.");  
 }  
  
 // Получить текущее местоположение объекта  
 public Node getObjectLocation(MovingObject object) {  
 return objectLocations.get(object);  
 }  
  
 // Вывести состояние графа  
 public void printGraph() {  
 System.out.println();  
 System.out.println("ГРАФ");  
 System.out.println("Узлы:");  
 for (Node node : nodes) {  
 System.out.println("- " + node);  
 }  
 System.out.println("\nРёбра:");  
 for (Edge edge : edges) {  
 System.out.println("- Из: " + edge.getFrom().getName() +  
 " В: " + edge.getTo().getName() +  
 " Расстояние: " + edge.getDistance());  
 }  
 System.out.println("\nМестоположения объектов:");  
 for (Map.Entry<MovingObject, Node> entry : objectLocations.entrySet()) {  
 System.out.println("- Объект: " + entry.getKey().getName() +  
 " Местоположение: " + entry.getValue().getName());  
 }  
 System.out.println("\nОперации объектов:");  
 for (Map.Entry<MovingObject, List<MovingObjectOperation>> entry : objectOperations.entrySet()) {  
 System.out.println("- Объект: " + entry.getKey().getName());  
 for (MovingObjectOperation operation : entry.getValue()) {  
 System.out.println(" - Из: " + operation.getFrom().getName() +  
 " В: " + operation.getTo().getName() +  
 " Задержка: " + operation.getDelayInHours() + " часов.");  
 }  
 }  
 }  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class MovingObject {  
 private Integer id; // Идентификатор объекта  
 private String name; // Наименование объекта  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class MovingObjectOperation {  
 private MovingObject movingObject; // Объект перемещения  
 private Node from; // Начальный узел графа  
 private Node to; // Конечный узел графа  
 private Double delayInHours; // Время задержки в часах  
 private Double distance; // Расстояние между узлами  
  
 public MovingObjectOperation(MovingObject movingObject, Node from, Node to, Double delayInHours) {  
 this.movingObject = movingObject;  
 this.from = from;  
 this.to = to;  
 this.delayInHours = delayInHours;  
 }  
}

package Lab3\_Event.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Node {  
 private Integer id; // Уникальный идентификатор узла  
 private String name; // Название узла  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.AddedOperation;  
import Lab3\_Event.Model.MovingObjectOperation;  
  
import javax.enterprise.event.Observes;  
import java.util.List;  
  
public class AverageSpeedCalculator {  
  
 // Метод для обработки события и расчёта средней скорости всех объектов  
 public void calculateAverageSpeed(@Observes @AddedOperation List<MovingObjectOperation> operations) {  
 double totalDistance = 0.0;  
 double totalTime = 0.0;  
  
 for (MovingObjectOperation operation : operations) {  
 double delayInHours = operation.getDelayInHours();  
 double distance = operation.getDistance(); // Используем расстояние, переданное в операции  
  
 totalDistance += distance;  
 totalTime += delayInHours;  
 }  
  
 if (totalTime > 0) {  
 double averageSpeed = totalDistance / totalTime; // Средняя скорость = общий путь / общее время  
 System.out.println("Средняя скорость всех объектов: " + averageSpeed + " км/ч");  
 } else {  
 System.out.println("Средняя скорость всех объектов не рассчитана: общее время равно нулю.");  
 }  
 }  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.AddedOperation;  
import Lab3\_Event.Model.MovingObjectOperation;  
  
import javax.enterprise.event.Observes;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class ObjectSpeedCalculator {  
  
 // Метод для обработки события и расчёта скорости каждого объекта  
 public void calculateSpeeds(@Observes @AddedOperation List<MovingObjectOperation> operations) {  
 Map<String, Double> objectSpeeds = new HashMap<>();  
  
 for (MovingObjectOperation operation : operations) {  
 String objectName = operation.getMovingObject().getName();  
 double delayInHours = operation.getDelayInHours();  
 double distance = operation.getDistance(); // Используем расстояние, переданное в операции  
  
 if (delayInHours > 0) {  
 double speed = distance / delayInHours; // Скорость = расстояние / время  
 objectSpeeds.put(objectName, speed);  
 }  
 }  
  
 // Печать рассчитанных скоростей  
 objectSpeeds.forEach((objectName, speed) ->  
 System.out.println("Скорость объекта " + objectName + ": " + speed + " км/ч")  
 );  
 }  
}

package Lab3\_Event;  
  
import Lab3\_Event.Model.\*;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
public class Laboratory3 {  
 protected static Weld weld;  
 protected static WeldContainer container;  
  
 public static void main(String[] args) throws Exception {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
 Graph graph = container.select(Graph.class).get();  
  
 Node nodeA = new Node(1, "A");  
 Node nodeB = new Node(2, "B");  
 Node nodeC = new Node(3, "C");  
 Node nodeD = new Node(4, "D");  
 Node nodeE = new Node(5, "E");  
 Node nodeF = new Node(6, "F");  
 Node nodeG = new Node(7, "G");  
  
  
 // Добавление узлов в граф  
 graph.addNode(nodeA);  
 graph.addNode(nodeB);  
 graph.addNode(nodeC);  
 graph.addNode(nodeD);  
 graph.addNode(nodeE);  
 graph.addNode(nodeF);  
 graph.addNode(nodeG);  
  
 // Инициализация рёбер  
 graph.addEdge(nodeA, nodeB, 3.0);  
 graph.addEdge(nodeA, nodeC, 6.0);  
 graph.addEdge(nodeA, nodeD, 5.0);  
 graph.addEdge(nodeB, nodeC, 21.0);  
 graph.addEdge(nodeD, nodeC, 15.0);  
 graph.addEdge(nodeC, nodeE, 13.0);  
 graph.addEdge(nodeC, nodeF, 1.0);  
 graph.addEdge(nodeC, nodeG, 34.0);  
 graph.addEdge(nodeE, nodeF, 23.0);  
  
 // Инициализация объектов  
 MovingObject drone = new MovingObject(1, "Дрон");  
 MovingObject truck = new MovingObject(2, "Грузовик");  
  
  
 MovingObjectOperation operation1 = new MovingObjectOperation(drone, nodeA, nodeD, 2.0);  
 MovingObjectOperation operation2 = new MovingObjectOperation(drone, nodeD, nodeC, 4.0);  
 MovingObjectOperation operation3 = new MovingObjectOperation(drone, nodeC, nodeE, 12.0);  
 MovingObjectOperation operation4 = new MovingObjectOperation(drone, nodeE, nodeF, 22.0);  
  
 MovingObjectOperation operation5 = new MovingObjectOperation(truck, nodeA, nodeB, 13.0);  
 MovingObjectOperation operation6 = new MovingObjectOperation(truck, nodeB, nodeC, 11.0);  
 MovingObjectOperation operation7 = new MovingObjectOperation(truck, nodeC, nodeG, 9.0);  
  
 // Добавление операций в граф  
 graph.addOperation(operation1);  
 graph.addOperation(operation2);  
 graph.addOperation(operation3);  
 graph.addOperation(operation4);  
 graph.addOperation(operation5);  
 graph.addOperation(operation6);  
 graph.addOperation(operation7);  
  
 graph.printGraph();  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
  
}

# **ВЫВОД**

Работа демонстрирует использование событийного механизма CDI для обработки операций в графе. Гибкость CDI позволяет реализовать расчет метрик (скорости и средней скорости) без модификации кода графа. Архитектура программы хорошо изолирует логику обработки данных, делает ее расширяемой и удобной для сопровождения. Такой подход идеально подходит для моделирования сложных процессов в графах, включая транспортные сети и логистику.