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 №12

Выполнил: Бугаева А. Л.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc183379324)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc183379325)

[ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ 6](#_Toc183379326)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 7](#_Toc183379327)

[КОД ПРОГРАММЫ 8](#_Toc183379328)

[ВЫВОД 13](#_Toc183379329)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать систему взаимодействующих классов: класс, описывающий множество точек на двухмерной плоскости; класс, определяющий, такую точку, от которой сумма расстояние к другим точкам наименьшая; класс, определяющий три разные точки, которые составляют треугольник наибольшего периметра. Классы определения, должны срабатывать при каждом изменении в описании множества точек.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

Цель данной лабораторной работы – разработка и внедрение системы для обработки событий, связанных с добавлением и удалением точек в набор, с дальнейшим вычислением расстояний и периметров треугольников, образующихся из этих точек. В рамках лабораторной работы используются аннотации для квалификации событий в CDI (Contexts and Dependency Injection), а также обработчики для этих событий, что позволяет гибко реагировать на изменения в наборе точек.

В начале работы был создан класс Point, который представляет собой точку на плоскости с координатами xxx и yyy. В нем реализован метод distanceTo, позволяющий вычислять расстояние между текущей точкой и другой точкой на основе формулы Евклида. Этот метод использует стандартные математические функции Math.sqrt и Math.pow для вычисления расстояния. Также переопределены методы toString, чтобы красиво выводить точку в виде строки, и equals, чтобы сравнивать две точки на равенство.

Следующим шагом стало создание класса PointSet, который управляет набором точек. Этот класс инкапсулирует коллекцию точек и предоставляет методы для добавления и удаления точек. Важной частью является то, что добавление и удаление точек сопровождается генерацией событий с помощью CDI. Для этого используются аннотации @AddedPoint и @RemovedPoint, которые квалифицируют события добавления и удаления точек соответственно. Когда точка добавляется или удаляется, соответствующие события публикуются, и другие компоненты системы могут реагировать на эти изменения. Также в классе реализован метод для вывода списка всех точек в наборе.

Важным аспектом лабораторной работы является реализация обработки этих событий в классах PointDistanceCalculator и TrianglePerimeterCalculator. Класс PointDistanceCalculator реагирует на события добавления и удаления точек и вычисляет точку с минимальной суммой расстояний до остальных точек в наборе. Это достигается путем перебора всех точек и подсчета суммарного расстояния для каждой точки до остальных, после чего выбирается точка с минимальной суммой.

Аналогичным образом работает класс TrianglePerimeterCalculator, который обрабатывает события добавления и удаления точек, но в этом случае вычисляется треугольник с максимальным периметром, образованный из точек. Для этого перебираются все возможные треугольники, образуемые из трех точек, и для каждого треугольника вычисляется его периметр. Треугольник с максимальным периметром выводится в консоль.

В самой главной программе (Laboratory3) создается и инициализируется контейнер CDI, с помощью которого инжектируются необходимые компоненты. Создаются несколько точек, которые затем добавляются в PointSet, и в процессе их добавления и удаления происходят вычисления, связанные с минимальными расстояниями и максимальными периметрами треугольников. Все изменения в наборе точек приводят к вызову соответствующих обработчиков событий, и результаты вычислений выводятся на экран.

В конце работы производится выключение контейнера CDI, что завершает работу приложения. Этот этап необходим для корректного завершения работы с CDI-контейнером и освобождения всех ресурсов.

# **ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ**

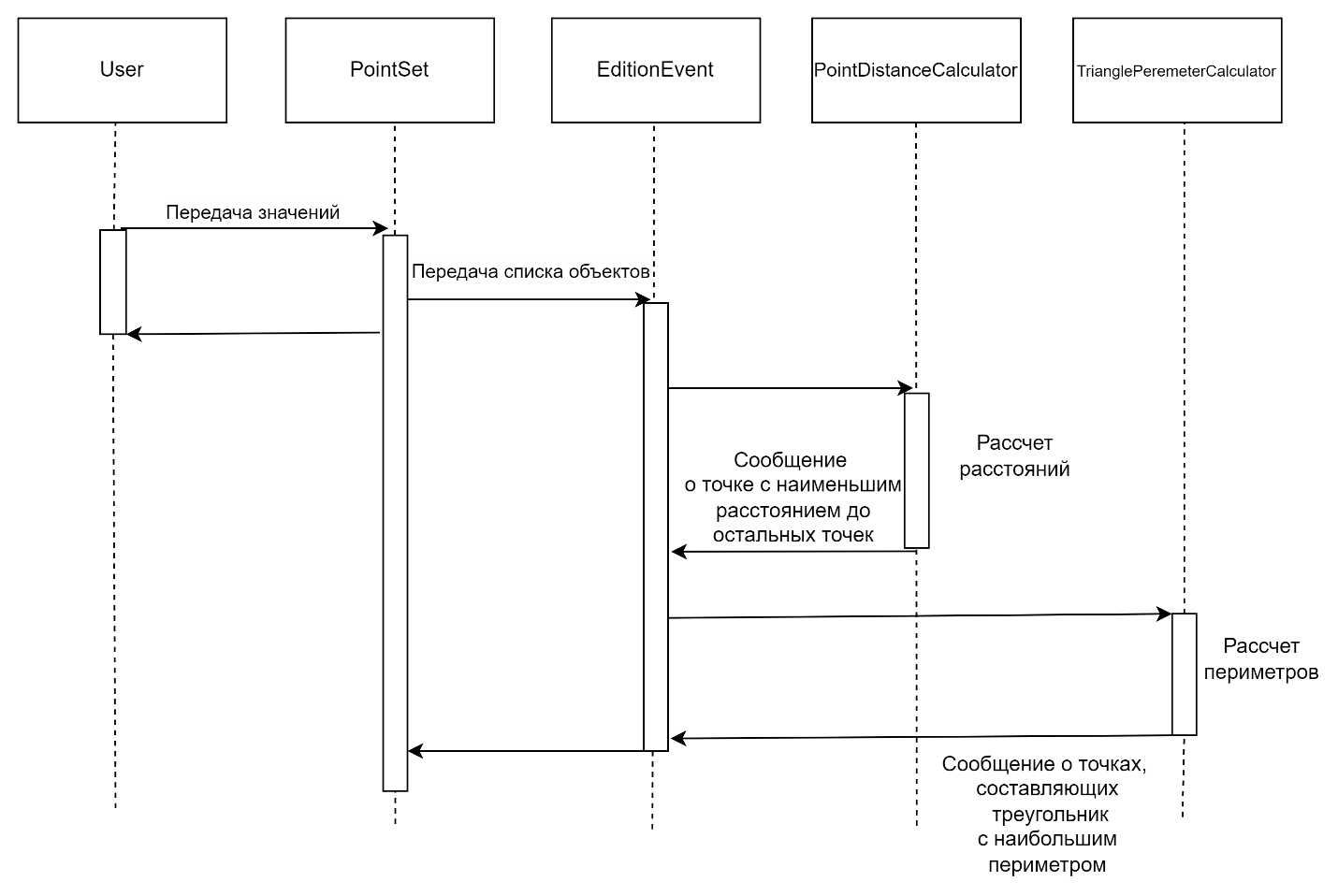


Рисунок 1 – Диаграмма последовательности

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

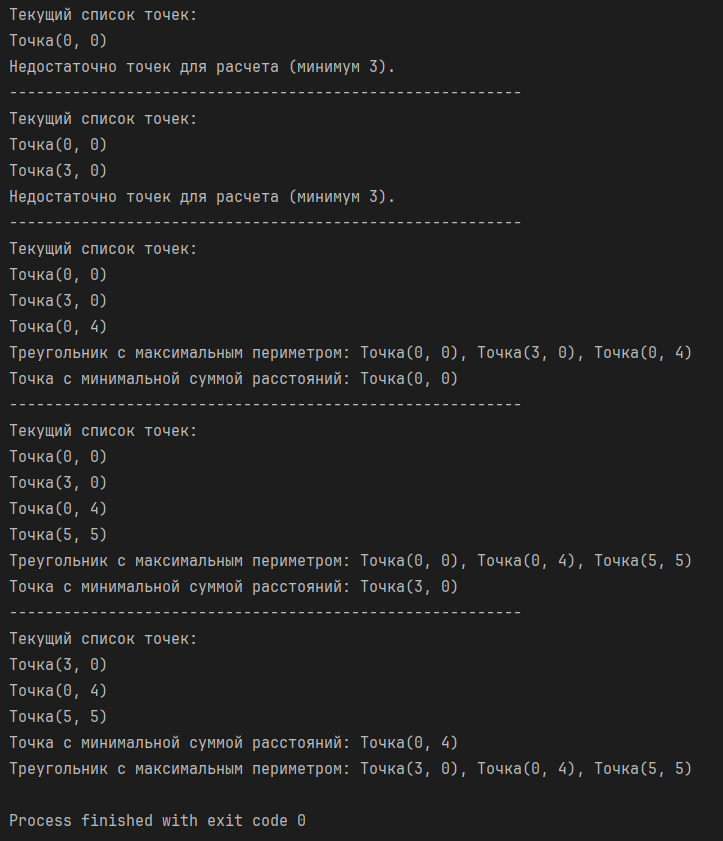


Рисунок 2 – Результат работы программы

# **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 AddedPoint {  
}

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 RemovedPoint {  
}

package Lab3\_Event.Util;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class Point {  
 private int x;  
 private int y;  
  
 // Метод для вычисления расстояния между двумя точками  
 public double distanceTo(Point other) {  
 return Math.sqrt(Math.pow(this.x - other.x, 2) + Math.pow(this.y - other.y, 2));  
 }  
  
 @Override  
 public String toString() {  
 return "Точка(" + x + ", " + y + ")";  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Point point = (Point) o;  
 return Double.compare(point.x, x) == 0 && Double.compare(point.y, y) == 0;  
 }  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.AddedPoint;  
import Lab3\_Event.Annotation.RemovedPoint;  
  
import javax.enterprise.event.Observes;  
import java.util.List;  
  
public class PointDistanceCalculator {  
  
 // Обработка события добавления точки  
 public void calculateDistanceOnAdd(@Observes @AddedPoint List<Point> pointList) {  
 if (pointList.size() < 3) {  
 System.out.println("Недостаточно точек для расчета (минимум 3).");  
 return;  
 }  
 Point minDistancePoint = findPointWithMinDistance(pointList);  
 System.out.println("Точка с минимальной суммой расстояний: " + minDistancePoint);  
 }  
  
 // Обработка события удаления точки  
 public void calculateDistanceOnRemove(@Observes @RemovedPoint List<Point> pointList) {  
 if (pointList.size() < 3) {  
 System.out.println("Недостаточно точек для расчета (минимум 3).");  
 return;  
 }  
 Point minDistancePoint = findPointWithMinDistance(pointList);  
 System.out.println("Точка с минимальной суммой расстояний: " + minDistancePoint);  
 }  
  
 // Метод для нахождения точки с минимальной суммой расстояний  
 private Point findPointWithMinDistance(List<Point> pointList) {  
 double minDistanceSum = Double.MAX\_VALUE;  
 Point minPoint = null;  
  
 for (int i = 0; i < pointList.size(); i++) {  
 Point p1 = pointList.get(i);  
 double distanceSum = 0;  
 for (int j = 0; j < pointList.size(); j++) {  
 if (i != j) {  
 distanceSum += p1.distanceTo(pointList.get(j));  
 }  
 }  
 if (distanceSum < minDistanceSum) {  
 minDistanceSum = distanceSum;  
 minPoint = p1;  
 }  
 }  
 return minPoint;  
 }  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.AddedPoint;  
import Lab3\_Event.Annotation.RemovedPoint;  
import lombok.Getter;  
  
import javax.enterprise.event.Event;  
import javax.inject.Inject;  
import java.util.ArrayList;  
import java.util.List;  
  
public class PointSet {  
  
 @Getter  
 private List<Point> pointList = new ArrayList<>();  
  
 @Inject  
 @AddedPoint  
 private Event<List<Point>> pointAddedEvent;  
  
 @Inject  
 @RemovedPoint  
 private Event<List<Point>> pointRemovedEvent;  
  
 public void addPoint(Point point) {  
 pointList.add(point);  
 printPoints();  
 pointAddedEvent.fire(new ArrayList<>(pointList)); // Генерация события добавления точки  
 }  
  
 public void removePoint(Point point) {  
 if (pointList.contains(point)) {  
 pointList.remove(point);  
 printPoints();  
 pointRemovedEvent.fire(new ArrayList<>(pointList)); // Генерация события удаления точки  
 } else {  
 System.out.println("Точка не найдена: " + point);  
 }  
 }  
  
 private void printPoints() {  
 System.out.println("Текущий список точек:");  
 for (Point point : pointList) {  
 System.out.println(point);  
 }  
 }  
  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.AddedPoint;  
import Lab3\_Event.Annotation.RemovedPoint;  
  
import javax.enterprise.event.Observes;  
import java.util.List;  
  
public class TrianglePerimeterCalculator {  
  
 // Обработка события добавления точки  
 public void calculatePerimeterOnAdd(@Observes @AddedPoint List<Point> pointList) {  
 if (pointList.size() >= 3) {  
 Point[] triangle = findTriangleWithMaxPerimeter(pointList);  
 System.out.println("Треугольник с максимальным периметром: " + triangle[0] + ", " + triangle[1] + ", " + triangle[2]);  
 }  
 }  
  
 // Обработка события удаления точки  
 public void calculatePerimeterOnRemove(@Observes @RemovedPoint List<Point> pointList) {  
 if (pointList.size() >= 3) {  
 Point[] triangle = findTriangleWithMaxPerimeter(pointList);  
 System.out.println("Треугольник с максимальным периметром: " + triangle[0] + ", " + triangle[1] + ", " + triangle[2]);  
 }  
 }  
  
 // Метод для нахождения треугольника с максимальным периметром  
 private Point[] findTriangleWithMaxPerimeter(List<Point> pointList) {  
 double maxPerimeter = Double.MIN\_VALUE;  
 Point[] maxTriangle = new Point[3];  
  
 for (int i = 0; i < pointList.size(); i++) {  
 for (int j = i + 1; j < pointList.size(); j++) {  
 for (int k = j + 1; k < pointList.size(); k++) {  
 Point p1 = pointList.get(i);  
 Point p2 = pointList.get(j);  
 Point p3 = pointList.get(k);  
 double perimeter = p1.distanceTo(p2) + p2.distanceTo(p3) + p3.distanceTo(p1);  
 if (perimeter > maxPerimeter) {  
 maxPerimeter = perimeter;  
 maxTriangle[0] = p1;  
 maxTriangle[1] = p2;  
 maxTriangle[2] = p3;  
 }  
 }  
 }  
 }  
 return maxTriangle;  
 }  
}

package Lab3\_Event;  
  
import Lab3\_Event.Util.Point;  
import Lab3\_Event.Util.PointSet;  
import Lab3\_Event.Util.PointDistanceCalculator;  
import Lab3\_Event.Util.TrianglePerimeterCalculator;  
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) {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
 // Получаем нужные бины через контейнер CDI  
 PointSet pointSet = container.select(PointSet.class).get();  
  
 // Создаем точки  
 Point p1 = new Point(0, 0);  
 Point p2 = new Point(3, 0);  
 Point p3 = new Point(0, 4);  
 Point p4 = new Point(5, 5);  
  
 // Добавляем точки в PointSet и проверяем результаты  
 pointSet.addPoint(p1);  
 System.out.println("---------------------------------------------------------");  
 pointSet.addPoint(p2);  
 System.out.println("---------------------------------------------------------");  
 pointSet.addPoint(p3);  
 System.out.println("---------------------------------------------------------");  
 pointSet.addPoint(p4);  
 System.out.println("---------------------------------------------------------");  
 pointSet.removePoint(p1); // Удаляем точку p1  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была разработана система для обработки событий добавления и удаления точек в набор с использованием CDI (Contexts and Dependency Injection). Реализованные компоненты позволили динамически реагировать на изменения в наборе точек, а также вычислять важные геометрические параметры, такие как точка с минимальной суммой расстояний до остальных точек и треугольник с максимальным периметром.

В ходе работы было освоено использование CDI для обработки событий, что позволяет эффективно разделять логику и реакцию на изменения в данных. Аннотации, такие как @AddedPoint и @RemovedPoint, обеспечивают квалификацию событий и позволяют их правильно обрабатывать в соответствующих классах. Также был продемонстрирован принцип работы с коллекциями и вычислениями, связанными с геометрией, через использование методов для нахождения минимальных расстояний и периметров.

В итоге, лабораторная работа продемонстрировала возможности CDI для создания гибкой и расширяемой архитектуры, где обработка событий и вычисления могут быть легко масштабированы и адаптированы под новые требования.