Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Лабораторная работа по

Основам программной инженерии №4

Вариант 3087

Работу выполнил:

Алферов Г.А.

Группа:

P3207

Санкт-Петербург,

2025

Оглавление

[Текст задания 3](#_Toc165337547)

[Реализация 4](#_Toc165337548)

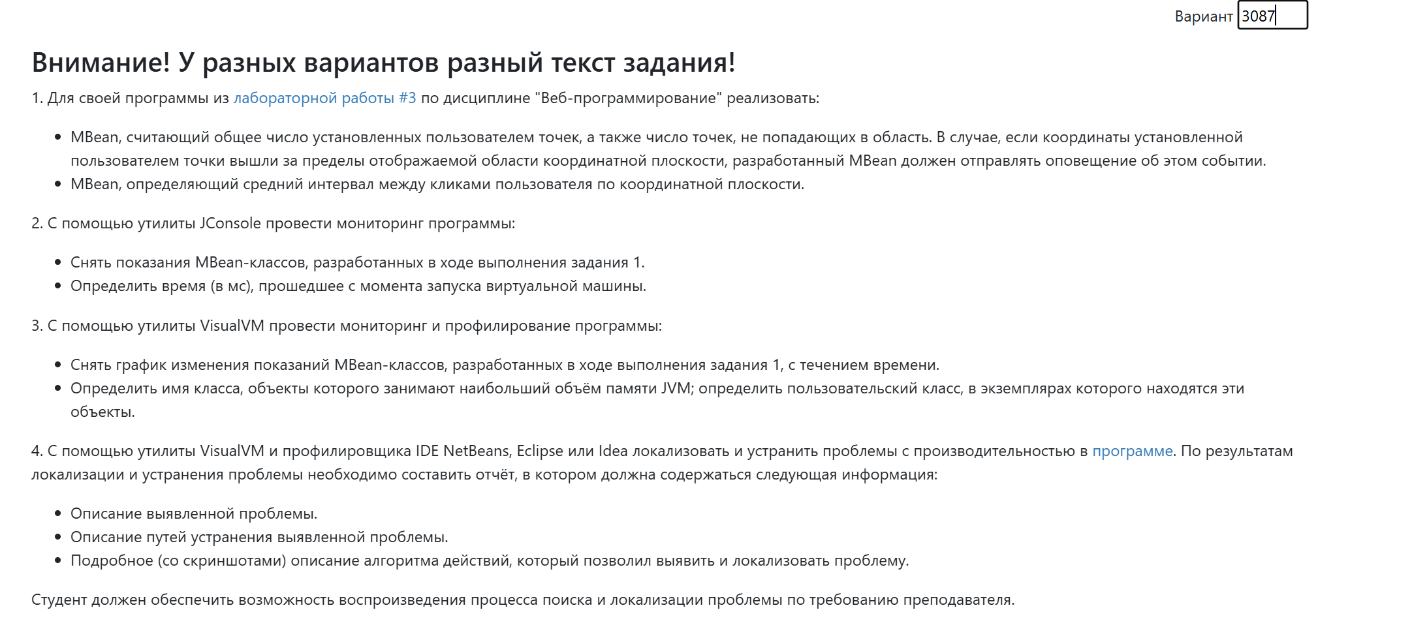
[Показания MBean классов из JConsole 8](#_Toc165337549)

[График изменения показаний MBean-классов с течением времени 10](#_Toc165337550)

[Исследование программы на утечки памяти 11](#_Toc165337551)

[Вывод 13](#_Toc165337552)

## Текст задания



## Реализация

**interface PointTrackerMBean**

package org.GleBlassUSA.beans;  
  
public interface PointTrackerMBean {  
 long getTotalPoints();  
 long getMissedPoints();  
}

**interface ClickIntervalMBean**

package org.GleBlassUSA.beans;  
  
public interface ClickIntervalMBean {  
 double getClickInterval();  
 long getClickCount();  
 void resetStat();  
}

**PointTracker**

package org.GleBlassUSA.beans;  
  
import jakarta.enterprise.context.ApplicationScoped;  
import jakarta.enterprise.context.Destroyed;  
import jakarta.enterprise.context.Initialized;  
import jakarta.enterprise.event.Observes;  
import jakarta.inject.Inject;  
import jakarta.inject.Named;  
import org.GleBlassUSA.dao.PointDao;  
import org.GleBlassUSA.models.Point;  
  
import javax.management.\*;  
import java.io.Serializable;  
  
@Named("pointTracker")  
@ApplicationScoped  
public class PointTracker extends NotificationBroadcasterSupport implements Serializable, PointTrackerMBean {  
  
 private int sequenceNumber = 0;  
  
 @Inject  
 private PointDao pointDao;  
  
 public void init(@Observes @Initialized(ApplicationScoped.class) Object unused) {  
 org.GleBlassUSA.beans.MBeanRegistryUtil.*registerBean*(this, "pointTracker");  
 }  
  
 public void destroy(@Observes @Destroyed(ApplicationScoped.class) Object unused) {  
 MBeanRegistryUtil.*unregisterBean*(this);  
 }  
  
 @Override  
 public long getTotalPoints() {  
 return pointDao.getPoints().size();  
 }  
  
 @Override  
 public long getMissedPoints() {  
 return pointDao.getPoints().stream()  
 .filter(p -> !inArea(p))  
 .count();  
 }  
  
 public void checkPoint(Point point) {  
 if (inArea(point)) {  
 sendOut(point);  
 }  
 }  
  
 private boolean inArea(Point point) {  
 return point.calculate();  
 }  
  
 private void sendOut(Point point) {  
 Notification notification = new Notification(  
 "Point out of bounds",  
 this.getClass().getName(),  
 sequenceNumber++,  
 System.*currentTimeMillis*(),  
 String.*format*("Point (%.2f, %.2f out of bounds:", point.getX(), point.getY())  
 );  
 sendNotification(notification);  
 }  
  
 @Override  
 public MBeanNotificationInfo[] getNotificationInfo() {  
 String[] types = new String[]{  
 AttributeChangeNotification.*ATTRIBUTE\_CHANGE* };  
 String name = AttributeChangeNotification.class.getName();  
 String description = "Point out of bounds notification";  
 return new MBeanNotificationInfo[]{  
 new MBeanNotificationInfo(types, name, description)  
 };  
 }  
}

**ClickInterval**

package org.GleBlassUSA.beans;  
  
import jakarta.enterprise.context.ApplicationScoped;  
import jakarta.enterprise.context.Destroyed;  
import jakarta.enterprise.context.Initialized;  
import jakarta.enterprise.event.Observes;  
import jakarta.inject.Named;  
  
import java.io.Serializable;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.concurrent.atomic.AtomicReference;  
  
@Named("ClickInterval")  
@ApplicationScoped  
public class ClickInterval implements Serializable, ClickIntervalMBean {  
  
 private final AtomicLong lastClickTime = new AtomicLong();  
 private final AtomicLong clickCount = new AtomicLong();  
 private final AtomicReference<Double> averageInterval = new AtomicReference<>(0.0);  
 private boolean isFirstClick = true;  
  
 public void init(@Observes @Initialized(ApplicationScoped.class) Object unused) {  
 org.GleBlassUSA.beans.MBeanRegistryUtil.*registerBean*(this, "ClickInterval");  
 }  
  
 public void destroy(@Observes @Destroyed(ApplicationScoped.class) Object unused) {  
 MBeanRegistryUtil.*unregisterBean*(this);  
 }  
  
 public void recordClick() {  
 long now = System.*currentTimeMillis*();  
 if (isFirstClick) {  
 lastClickTime.set(now);  
 isFirstClick = false;  
 return;  
 }  
 long past = lastClickTime.getAndSet(now);  
 long interval = now - past;  
 long count = clickCount.incrementAndGet();  
 averageInterval.updateAndGet(cur -> (cur \* (count - 1) + interval) / count);  
 }  
  
  
 @Override  
 public double getClickInterval() {  
 return averageInterval.get() / 1000;  
 }  
  
 @Override  
 public long getClickCount() {  
 return clickCount.get();  
 }  
  
 @Override  
 public void resetStat() {  
 lastClickTime.set(0);  
 clickCount.set(0);  
 averageInterval.set(0.0);  
 }  
}

**MBeanRegistryUtil**

package org.GleBlassUSA.beans;  
  
import jakarta.servlet.ServletContextListener;  
import lombok.experimental.UtilityClass;  
  
import javax.management.\*;  
import java.lang.management.ManagementFactory;  
import java.util.HashMap;  
import java.util.Map;  
  
@UtilityClass  
public class MBeanRegistryUtil implements ServletContextListener {  
 private final Map<Class<?>, ObjectName> *beans* = new HashMap<>();  
  
 public void registerBean(Object bean, String name) {  
 try {  
 String domain = bean.getClass().getPackageName();  
 String type = bean.getClass().getSimpleName();  
 ObjectName objectName = new ObjectName(String.*format*("%s:type=%s,name=%s", domain, type, name));  
 ManagementFactory.*getPlatformMBeanServer*().registerMBean(bean, objectName);  
 *beans*.put(bean.getClass(), objectName);  
 } catch (InstanceAlreadyExistsException | MBeanRegistrationException | NotCompliantMBeanException |  
 MalformedObjectNameException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public void unregisterBean(Object bean) {  
 if (!*beans*.containsKey(bean.getClass())) {  
 throw new IllegalArgumentException("Specified bean is not registered.");  
 }  
 try {  
 ObjectName objectName = *beans*.get(bean.getClass());  
 ManagementFactory.*getPlatformMBeanServer*().unregisterMBean(objectName);  
 } catch (InstanceNotFoundException | MBeanRegistrationException ex) {  
 ex.printStackTrace();  
 }  
 }  
}

**PointService**

package org.GleBlassUSA.beans;  
  
  
import jakarta.enterprise.context.ApplicationScoped;  
import jakarta.inject.Inject;  
import org.GleBlassUSA.dao.PointDao;  
import org.GleBlassUSA.models.Point;  
  
@ApplicationScoped  
public class PointService {  
 private final PointDao pointDao = PointDao.*getInstance*();  
 @Inject  
 private PointTracker pointTracker;  
  
 public void addPoint(Point point) {  
 pointDao.addPoint(point);  
 pointTracker.checkPoint(point);   
 }  
}

**Весь проект**

https://github.com/wrakelft/OPI\_ITMO/tree/main/lab4/src

## Показания MBean классов из JConsole

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

В ходе использования утилиты Jconsole можно сделать вывод что MBeans были успешно разработаны. Уведомления отправляются штатно и позволяют отслеживать события.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.



420 000 мс

## График изменения показаний MBean-классов с течением времени

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Контент, сгенерированный ИИ, может содержать ошибки.

Видно, что больше всего памяти занимают значения TreeMap$Entry

Больше всего памяти из пользовательских классов занимают объекты Point

Эта не прямая связь обусловлена тем, что во время работы приложения, развернутого на WildFly он автоматически создает TreeMap для своих внутренних процессов (хранение конфигов, кэширования, обработка запросов), а так как при работе приложения WildFly вызывает наши сервлеты и бины он и создает временные TreeMap для инъекций зависимостей или других нужд приложения.

## Исследование программы на утечки памяти

Изображение выглядит как текст, снимок экрана, линия, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Установим максимальный размер кучи в 25Мб с помощью -Xmx25m и запустим программу. Из графика памяти видно, что размер кучи постоянно увеличивается, что говорит о проблемах с использованием памяти в программе.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

С помощью HeapDump найдем объекты, занимающие большую часть памяти.

Изображение выглядит как текст, снимок экрана, Шрифт, число

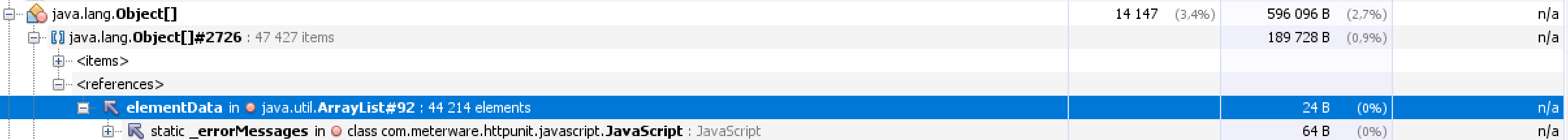
Контент, сгенерированный ИИ, может содержать ошибки.

Видно, что создаются экземпляры строк на каждый запрос этим можно объяснить и рост char[] и TreeMap$Entry, так как char[] хранит символы строк, String его обертка, а TreeMap$Entry связывает ключи String и другие наши объекты.

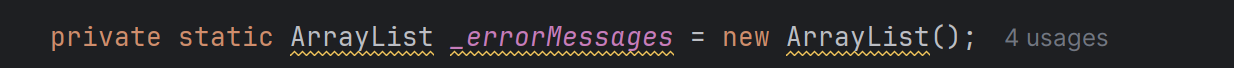
Изображение выглядит как текст, число, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Исследовав, видим повторяющиеся строки



Все они связаны с \_\_errorMessages

Объекты \_errorMessages хранятся в ArrayList

Добавление объектов в этот список:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Накопление \_errorMessages в списке, за счет чего и получается переполнение памяти.

В программе есть функция для очистки:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Просмотрев классы видим, что она не используется.

Решением будет очистка после выполнения запроса.

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Теперь память не стремится к максимальному значению. Программа не выкидывает OutOfMemoryError.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Вывод

Во время выполнения лабораторной работы я использовал утилиты для мониторинга работы программы JConsole и VisualVM. А также была произведена работа по поиску проблемы, ее локализации и устранению.