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 20111

Работу выполнила:

Касьяненко В.М.

Группа:

P3220

Санкт-Петербург,

2024

Оглавление

[Текст задания 3](#_Toc165337547)

[Реализация 4](#_Toc165337548)

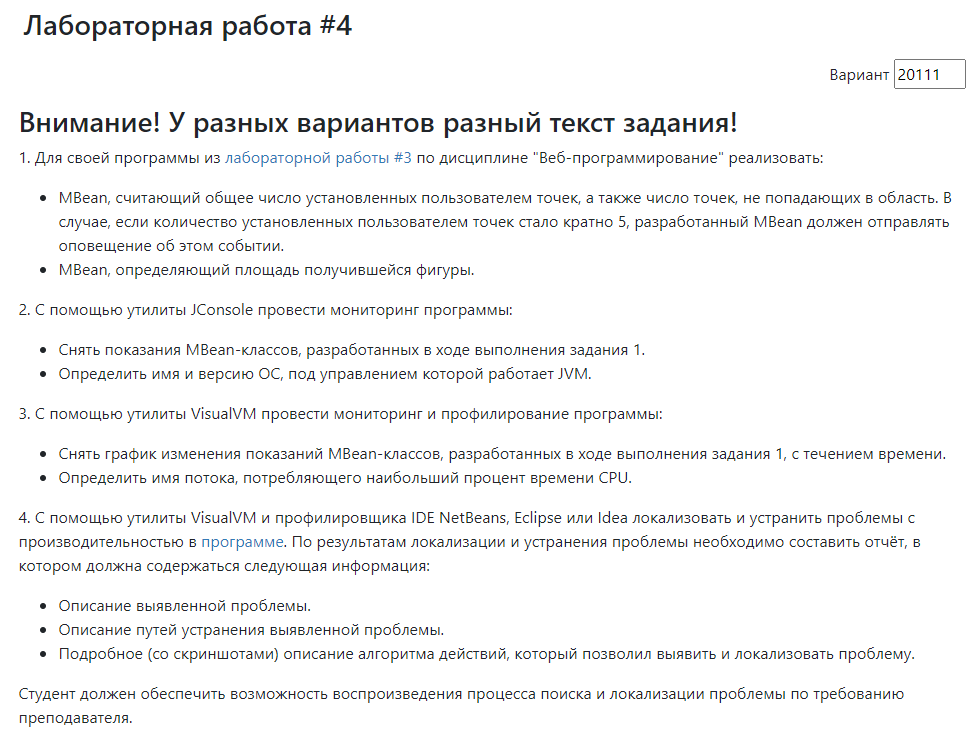
[Показания MBean классов из JConsole 8](#_Toc165337549)

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

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

[Вывод 13](#_Toc165337552)

## Текст задания



## Реализация

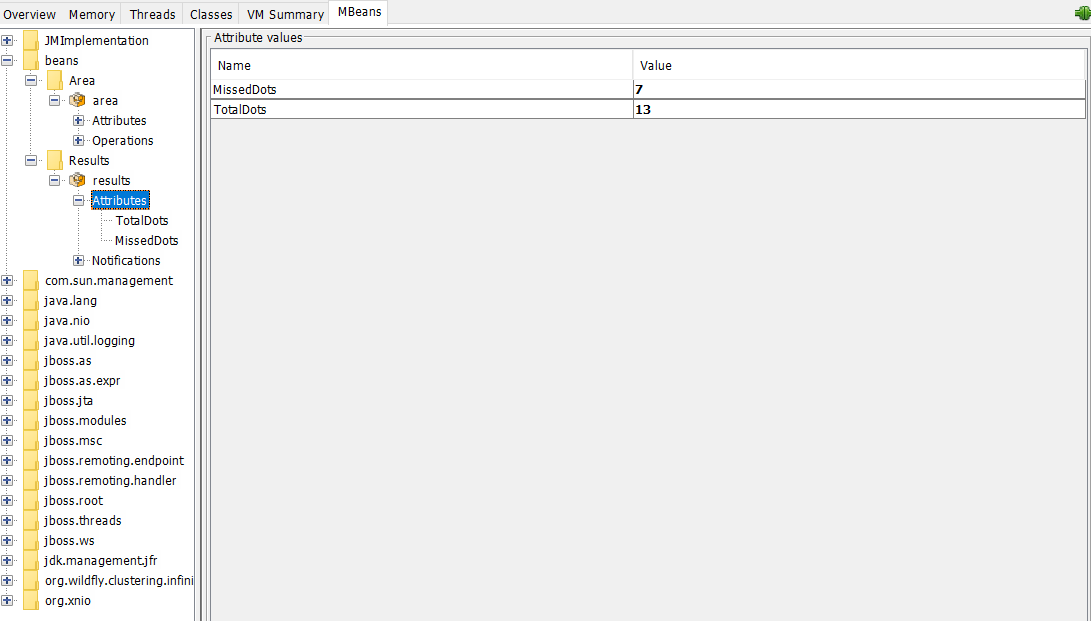
Results.java

package beans;  
  
import database.ResultDao;  
import entity.Result;  
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 lombok.Getter;  
import lombok.Setter;  
import util.MBeanRegistryUtil;  
  
import javax.management.AttributeChangeNotification;  
import javax.management.MBeanNotificationInfo;  
import javax.management.Notification;  
import javax.management.NotificationBroadcasterSupport;  
import java.io.Serializable;  
import java.util.List;  
  
@Getter  
@Setter  
@Named("results")  
@ApplicationScoped  
public class Results extends NotificationBroadcasterSupport implements Serializable, ResultsMBean {  
 private int sequenceNumber = 0;  
  
 @Inject  
 private ResultDao resultDao;  
  
 private Result currResult;  
 private List<Result> resultList;  
  
 public void init(@Observes @Initialized(ApplicationScoped.class) Object unused) {  
 currResult = new Result();  
 updateLocal();  
 MBeanRegistryUtil.*registerBean*(this, "results");  
 }  
  
 public void destroy(@Observes @Destroyed(ApplicationScoped.class) Object unused) {  
 MBeanRegistryUtil.*unregisterBean*(this);  
 }  
  
 private void updateLocal() {  
 resultList = resultDao.getAll();  
 }  
  
 public void addResult() {  
 Result copyResult = new Result(currResult);  
 resultDao.save(copyResult);  
 updateLocal();  
  
 if (resultList.size() % 5 == 0) {  
 Notification notification = new Notification(  
 "Total dots count is multiple of 5",  
 getClass().getSimpleName(),  
 sequenceNumber++,  
 "The total count of user-set dots is now multiple of 5!"  
 );  
 sendNotification(notification);  
 }  
 }  
  
 @Override  
 public long getTotalDots() {  
 return resultList.size();  
 }  
  
 @Override  
 public long getMissedDots() {  
 return resultList.size() - resultList.stream().filter(Result::getHit).count();  
 }  
  
 public void clearResults() {  
 resultDao.clear();  
 resultList = resultDao.getAll();  
 updateLocal();  
 }  
  
  
 @Override  
 public MBeanNotificationInfo[] getNotificationInfo() {  
 String[] types = new String[] { AttributeChangeNotification.*ATTRIBUTE\_CHANGE* };  
 String name = AttributeChangeNotification.class.getName();  
 String description = "Miss notification";  
 MBeanNotificationInfo info = new MBeanNotificationInfo(types, name, description);  
 return new MBeanNotificationInfo[] { info };  
 }  
}

Area.java

package beans;  
  
import database.ResultDao;  
import entity.Result;  
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 lombok.Getter;  
import lombok.Setter;  
import util.MBeanRegistryUtil;  
  
import java.io.Serializable;  
import java.util.List;  
  
@Getter  
@Setter  
@Named("area")  
@ApplicationScoped  
public class Area implements Serializable, AreaMBean {  
 private double area = 0;  
 @Inject  
 private ResultDao resultDao;  
  
 private Result currResult;  
 private List<Result> resultList;  
  
 public void init(@Observes @Initialized(ApplicationScoped.class) Object unused) {  
 currResult = new Result();  
 System.*out*.println(currResult);  
 updateLocal();  
 MBeanRegistryUtil.*registerBean*(this, "area");  
 }  
  
 public void destroy(@Observes @Destroyed(ApplicationScoped.class) Object unused) {  
 MBeanRegistryUtil.*unregisterBean*(this);  
 }  
  
 public void calculateArea(double r) {  
 area = ((Math.*PI* \* Math.*pow*(r / 2, 2) / 4) + ((r \* r/2) / 2) + (r \* r/2));  
 System.*out*.println(area);  
 System.*out*.println(r);  
 }  
  
 @Override  
 public double getArea() {  
 return area;  
 }  
  
 @Override  
 public void computeArea() {  
 System.*out*.println(currResult.getR());  
 calculateArea(currResult.getR());  
 }  
 private void updateLocal() {  
 resultList = resultDao.getAll();  
 }  
}

## Показания MBean классов из JConsole



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

Автоматически созданное описание

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

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Имя и версия ОС, под управлением которой работает JVM – Windows 10 10.0.

## График изменения показаний MBean-классов с течением времени

Изображение выглядит как текст, снимок экрана, число, диаграмма

Автоматически созданное описание

Изображение выглядит как линия, текст, График, диаграмма

Автоматически созданное описание

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

Автоматически созданное описание

Имя потока, потребляющего наибольший процент времени CPU – MSC service thread 1-1.

## Исследование программы на утечки памяти

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Установим максимальный размер кучи в 30Мб с помощью -Xmx30m и запустим программу. Из графика использования памяти видно, что размер кучи постоянно увеличивается, что свидетельствует о проблемах с использованием памяти в программе. Через некоторое количество времени получаем ошибку OutOfMemoryError.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

С помощью HeapDump найдем объекты, занимающие большую часть памяти.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Объекты \_errorMessages хранятся в ArrayList

Найдем строчку с добавлением объектов в этот список:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

В результате получается накопление \_errorMessages в списке, за счет чего и получается переполнение памяти.

В программе есть функция для очистки \_errorMessage, однако мы можем увидеть, что на самом деле она не используется

Изображение выглядит как текст, Шрифт, рукописный текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Решением будет очистка списка с \_errorMessage после выполнения очередного запроса.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Запустим программу. Теперь память не стремится к максимальному значению, как было раньше. Программа работает нормально и не выкидывает OutOfMemoryException.

Изображение выглядит как текст, линия, График, снимок экрана

Автоматически созданное описание

## Вывод

Во время выполнения лабораторной работы были изучены утилиты для мониторинга работы программы JConsole и VisualVM. А также был получен опыт по полученным данным определять утечки памяти и устранять их.