Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной инженерии и компьютерной техники

Кафедра Информатики, прикладной математики

Дисциплина: Программирование интернет приложений

Лабораторная работа №3

Вариант 793

Группа №Р3217

Вариант №754

Работу выполнил:

Хлопков Д.

Фищенко В.

Преподаватель:

Николаев В.В.

Санкт-Петербург, 2017 г.

**Текст задания:**

Разработать приложение на базе JavaServer Faces Framework, которое осуществляет проверку попадания точки в заданную область на координатной плоскости.

Приложение должно включать в себя 2 facelets-шаблона - стартовую страницу и основную страницу приложения, а также набор управляемых бинов (managed beans), реализующих логику на стороне сервера.

**Стартовая страница должна содержать следующие элементы:**

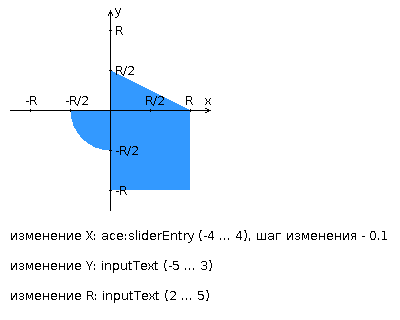
* "Шапку", содержащую ФИО студента, номер группы и номер варианта.
* Интерактивные часы, показывающие текущие дату и время, обновляющиеся раз в 12 секунд.
* Ссылку, позволяющую перейти на основную страницу приложения.

**Основная страница приложения должна содержать следующие элементы:**

* Набор компонентов для задания координат точки и радиуса области в соответствии с вариантом задания. Может потребоваться использование дополнительных библиотек компонентов - ICEfaces (префикс "ace") и PrimeFaces(префикс "p"). Если компонент допускает ввод заведомо некорректных данных (таких, например, как буквы в координатах точки или отрицательный радиус), то приложение должно осуществлять их валидацию.
* Динамически обновляемую картинку, изображающую область на координатной плоскости в соответствии с номером варианта и точки, координаты которых были заданы пользователем. Клик по картинке должен инициировать сценарий, осуществляющий определение координат новой точки и отправку их на сервер для проверки её попадания в область. Цвет точек должен зависить от факта попадания / непопадания в область. Смена радиуса также должна инициировать перерисовку картинки.
* Таблицу со списком результатов предыдущих проверок.
* Ссылку, позволяющую вернуться на стартовую страницу.

**Дополнительные требования к приложению:**

* Все результаты проверки должны сохраняться в базе данных под управлением СУБД Oracle.
* Для доступа к БД необходимо использовать JPA с ORM-провайдером на усмотрение студента.
* Для управления списком результатов должен использоваться Application-scoped Managed Bean.
* Конфигурация управляемых бинов должна быть задана с помощью аннотаций.
* Правила навигации между страницами приложения должны быть заданы в отдельном конфигурационном файле.



**Листинги кода:**

Страница main.xhtml (Основная страница)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. **<html** xmlns="http://www.w3.org/1999/xhtml"
5. xmlns:h="http://xmlns.jcp.org/jsf/html"
6. xmlns:f="http://xmlns.jcp.org/jsf/core"
7. xmlns:ace="http://www.icefaces.org/icefaces/components"**>**
8. **<h:head>**
9. **<title>**Lab3**</title>**
10. **<script** type="text/javascript" src="drawing.js" charset="utf-8"**></script>**
11. **</h:head>**
12. **<h:body** onload="drawDefault();"**>**
13. **<canvas** id="canv" width="300" height="300"**></canvas>**
15. **<h:form** id="form"**>**
16. Введите X:
18. **<ace:sliderEntry** id="xSlider" min="-400" max="400" stepPercent="2.5" value="#{checkAreaBean.x}"**>**
19. **<ace:ajax** event="slide" execute="@this" render="sliderValue"**/>**
20. **</ace:sliderEntry>**
21. **<h:outputText** id="sliderValue" value="#{checkAreaBean.x}"**/>**
22. **<br/>**
23. Введите Y:
24. **<h:inputText** id="yInput" value="#{checkAreaBean.y}"**>**
25. **<f:converter** converterId="converters.DoubleConverter"**/>**
26. **<f:validator** validatorId="validators.YValidator"**/>**
27. **</h:inputText>**
28. **<h:message** for="yInput" style="color:red"**/><br/>**
29. Введите R:
30. **<h:inputText** id="rInput" value="#{checkAreaBean.r}" onchange="drawArea(this)"**>**
31. **<f:converter** converterId="converters.DoubleConverter"**/>**
32. **<f:validator** validatorId="validators.RValidator"**/>**
33. **</h:inputText><h:message** for="rInput" style="color:red"**/><br/>**
34. **<h:commandButton** id="submitBtn" value="Отправить"**/><br/>**
35. **<h:outputText** value="#{checkAreaBean.output}" id="result"**/>**
36. **</h:form><br/>**
37. **<h:dataTable** value="#{resultsBean.resultList}" var="p"**>**
38. **<h:column>**
39. **<f:facet** name="header"**>**X**</f:facet>**
40. #{p.x}
41. **</h:column>**
42. **<h:column>**
43. **<f:facet** name="header"**>**Y**</f:facet>**
44. #{p.y}
45. **</h:column>**
46. **<h:column>**
47. **<f:facet** name="header"**>**R**</f:facet>**
48. #{p.r}
49. **</h:column>**
50. **<h:column>**
51. **<f:facet** name="header"**>**Попадает?**</f:facet>**
52. #{p.strike}
53. **</h:column>**
54. **</h:dataTable>**
55. **<h:link** outcome="index"**>**На стартовую страницу**</h:link>**
56. **</h:body>**
57. **</html>**

CheckAreaBean.java (Bean для проверки попадания в область)

1. **package** beans;
3. **import** database.PointEntity;
4. **import** database.PointUtil;
5. **import** org.hibernate.Session;
6. **import** org.hibernate.Transaction;
8. **import** javax.faces.bean.ManagedBean;
9. **import** javax.faces.bean.RequestScoped;
10. **import** javax.faces.context.FacesContext;
12. **import** database.DataAccess;
14. @ManagedBean
15. @RequestScoped
16. **public** **class** CheckAreaBean {
17. **private** Double x = 0.0;
18. **private** Double y;
19. **private** Double r;
20. **private** Boolean strike;
22. **public** Double getX() {
23. **return** **this**.x;
24. }
26. **public** **void** setX(Double x) {
27. **this**.x = x / 100;
28. }
30. **public** Double getY() {
31. **return** **this**.y;
32. }
34. **public** **void** setY(Double y) {
35. **this**.y = y;
36. }
38. **public** Double getR() {
39. **return** **this**.r;
40. }
42. **public** **void** setR(Double r) {
43. **this**.r = r;
44. }
46. **public** Boolean getStrike() {
47. **return** **this**.strike;
48. }
50. **public** **void** checkArea() {
51. **if** (!(x == **null** || y == **null** || r == **null**)) {
52. **if** (x < 0) {
53. **if** (y > 0) strike = **false**;
54. **else** {
55. **if** (x \* x + y \* y <= r \* r / 4) strike = **true**;
56. **else** strike = **false**;
57. }
58. } **else** {
59. **if** (y > 0) {
60. **if** (-x / 2 + r / 2 >= y) strike = **true**;
61. **else** strike = **false**;
62. } **else** {
63. **if** (x <= r && y >= -r) strike = **true**;
64. **else** strike = **false**;
65. }
66. }
67. *//Закомментить 2 строчки ниже, чтобы затестить без подключения к бд.*
68. DataAccess dataAccess = **new** DataAccess();
69. dataAccess.addNewPoint(x, y, r, strike);
70. FacesContext context = FacesContext.getCurrentInstance();
71. ResultsBean bean = (ResultsBean)context.getExternalContext().getApplicationMap().get("resultsBean");
72. bean.addToResultList(**this**);
73. }
74. }
76. **public** String getOutput() {
77. checkArea();
78. **if** (strike == **null**) **return** "";
79. String outputMessage = "Точка (" + **this**.x + "; " + **this**.y + ") ";
80. **if** (!strike) outputMessage += "не ";
81. outputMessage += "попадает в область радиусом " + **this**.r;
82. **return** outputMessage;
83. }
84. }

ResultsBean.java (Bean для хранения результатов)

1. **package** beans;
3. **import** org.hibernate.annotations.Check;
5. **import** javax.faces.bean.ApplicationScoped;
6. **import** javax.faces.bean.ManagedBean;
7. **import** java.util.ArrayList;
8. **import** java.util.List;
10. @ManagedBean
11. @ApplicationScoped
12. **public** **class** ResultsBean {
13. **private** List<CheckAreaBean> resultList;
14. **public** ResultsBean(){
15. resultList = **new** ArrayList<CheckAreaBean>();
16. }
17. **public** List<CheckAreaBean> getResultList() {
18. **return** resultList;
19. }
20. **public** **void** addToResultList(CheckAreaBean point){
21. **this**.resultList.add(point);
22. System.out.println("added");
23. }
24. }

PointEntity.java (Сущность для описания точки)

1. **package** database;
3. **import** org.hibernate.annotations.Type;
5. **import** javax.persistence.\*;
7. @Entity
8. @Table(name = "POINT", schema = "s223375")
9. **public** **class** PointEntity {
11. @Basic
12. @Column(name = "X", nullable = **false**)
13. **private** **double** x;
15. @Basic
16. @Column(name = "Y", nullable = **false**)
17. **private** **double** y;
19. @Basic
20. @Column(name = "R", nullable = **false**)
21. **private** **double** r;
23. @Basic
24. @Type(type = "true\_false")
25. @Column(name = "STRIKE", nullable = **false**   )
26. **private** **boolean** strike;
28. @Id
29. @SequenceGenerator(name = "SEQ", sequenceName = "ID\_SEQUENCE", schema = "s223375", allocationSize=1)
30. @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ")
31. @Column(name = "ID")
32. **private** **long** id;

35. **public** **double** getX() {
36. **return** x;
37. }
39. **public** **void** setX(**double** x) {
40. **this**.x = x;
41. }

44. **public** **double** getY() {
45. **return** y;
46. }
48. **public** **void** setY(**double** y) {
49. **this**.y = y;
50. }

53. **public** **double** getR() {
54. **return** r;
55. }
57. **public** **void** setR(**double** r) {
58. **this**.r = r;
59. }

62. **public** **boolean** getStrike() {
63. **return** strike;
64. }
66. **public** **void** setStrike(**boolean** strike) {
67. **this**.strike = strike;
68. }
70. }

DataAccess.java (класс содержащий методы доступа к таблице point)

1. **package** database;
3. **import** org.hibernate.Session;
4. **import** org.hibernate.SessionException;
5. **import** org.hibernate.Transaction;
7. **import** java.util.List;
9. **public** **class** DataAccess {
10. **private** Session session;
11. **public** DataAccess(){
12. **try** {
13. session = PointUtil.getSessionFactory().openSession();
14. }
15. **catch** (Throwable ex){
16. System.err.println("Opening session failed" + ex.getMessage());
17. **throw** **new** SessionException(ex.getMessage());
18. }
19. }
20. **public** **void** closeSession(){
21. session.close();
22. }
23. **public** **void** addNewPoint(**double** x, **double** y, **double** r, **boolean** strike){
24. Transaction tx = session.beginTransaction();
25. PointEntity point = **new** PointEntity();
26. point.setStrike(strike);
27. point.setR(r);
28. point.setY(y);
29. point.setX(x);
30. session.save(point);
31. tx.commit();
32. }
33. **public** List<PointEntity> getListOfPoints(){
34. **return** (List<PointEntity>)session.createQuery("from PointEntity order by id").list();
35. }
36. **public** **void** deleteAllFromTable(){
37. session.createQuery("delete from PointEntity");
38. }
39. }

Создание Таблицы и Sequence в oracle

1. **CREATE** **TABLE** POINT (
2. ID **NUMBER** **PRIMARY** **KEY**,
3. X **FLOAT** **NOT** **NULL**,
4. Y **FLOAT** **NOT** **NULL**,
5. R **FLOAT** **NOT** **NULL**,
6. STRIKE **CHAR**(1) **NOT** **NULL**,
7. **CONSTRAINT** BOOL\_CNSTR (**CHECK** STRIKE **IN** ('T', 'F')));
8. **CREATE** **SEQUENCE** ID\_SEQUENCE
9. **START** **WITH** 1
10. MINVALUE 1
11. **INCREMENT** **BY** 1;

**Вывод:**

В ходе выполнения лабораторной работы была изучена технология JSF, структура приложения основанная на этой технологии и такие ее элементы, как: Managed Beans, валидаторы, конвертеры. Также был изучен доступ к базе данных oracle из java приложений при помощи ORM-провайдера Hibernate.ы