Государственный Университет Молдовы

Факультет Математики и Информатики

Департамент Информатики

“Tehnologii Java”

Лабораторная работа 2

Проверил: Латул Георгий

Выполнил: Чобану Артём

Кишинев 2021

Задание:

Создать приложение, работающее с БД посредством ГПИ(GUI). Пользователь должен иметь возможность в окне приложения

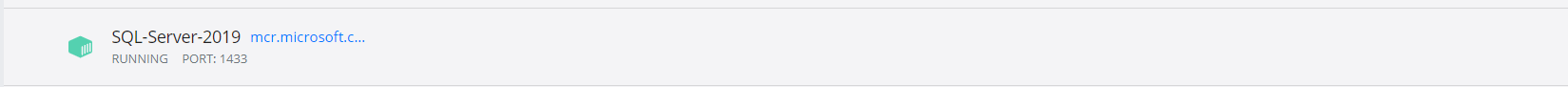
* вводить данные в БД (из полей GUI) и из текстового файла;
  + выводить данные из БД в окне;
  + выводить данные из БД по определённым критериям (минимум 2);
  + выводить отсортированные данные.

В БД должно быть минимум 3 таблицы. При проектировании графического пользовательского интерфейса использовать модель MVC (**Model/View/Controller, *модель – вид – контроллер***)

**БД** некого фитнесс клуба, которая состоит из таблиц: clients – информация о клиентах фитнесс клуба, gympass – информация о абонементах фитнесс клуба, managers – информация о менеджерах фитнесс клуба (к каждому клиенту приставляется менеджер).

Я запустил SQL Server 2019 в Docker контейнере:

docker run -e "ACCEPT\_EULA=Y" -e "SA\_PASSWORD=yourStrong(!)Password" -e "MSSQL\_PID=Express" -p 1433:1433 -d mcr.microsoft.com/mssql/server:2019-latest



Интерфейс таблицы(в данном случае менеджеров)

Graphical user interface, application

Description automatically generated

Добавление записи (По нажатию на кнопку **Add**):

Graphical user interface, application, Word

Description automatically generated

Результат:

Graphical user interface

Description automatically generated

Удаление записи (По нажатию на кнопку **Remove**):

Graphical user interface, application

Description automatically generated

Результат:

Graphical user interface

Description automatically generated

Страница Абонементов в спортзал:

Graphical user interface

Description automatically generated

Код программы

Класс Lab2, в котором находится метод main:

import Controllers.ClientController;  
import Controllers.GymPassController;  
import Controllers.ManagerController;  
import Mappers.ClientMapper;  
import Mappers.GymPassMapper;  
import Mappers.ManagerMapper;  
import Repositories.ClientRepository;  
import Repositories.GymPassRepository;  
import Repositories.ManagerRepository;  
import Views.EntityInfoTable;  
  
class Lab2  
{  
 private static ClientController clientController;  
 private static GymPassController gymPassController;  
 private static ManagerController managerController;  
  
 public static void main(String[] args)  
 {  
 var connectionString = "jdbc:sqlserver://localhost:1433;username=sa;password=yourStrong(!)Password";  
 Launch(connectionString);  
 }

private static void Launch(String connectionString)  
 {  
 var managerRepository = new ManagerRepository(connectionString);  
 var clientRepository = new ClientRepository(connectionString);  
 var gymPassRepository = new GymPassRepository(connectionString);  
  
 var clientMapper = new ClientMapper();  
 var gymPassMapper = new GymPassMapper();  
 var managerMapper = new ManagerMapper();  
  
 var clientInfoTable = new EntityInfoTable<>(  
 "Clients",  
 new String[]  
 {  
 "Id",  
 "Name",  
 "RegistrationDate",  
 "GymPassId",  
 "ManagerId"  
 },  
 clientMapper,  
 "Gym Passes",  
 "Managers");  
 var gymPassInfoTable = new EntityInfoTable<>(  
 "GymPasses",  
 new String[]  
 {  
 "Id",  
 "Price",  
 "MonthAmount"  
 },  
 gymPassMapper,  
 "Managers",  
 "Clients");  
 var managerInfoTable = new EntityInfoTable<>(  
 "Managers",  
 new String[]  
 {  
 "Id",  
 "Name"  
 },  
 managerMapper,  
 "Clients",  
 "GymPasses");  
  
  
 managerController = new ManagerController(managerRepository, managerInfoTable, managerMapper);  
 clientController = new ClientController(clientRepository, clientInfoTable, clientMapper);  
 gymPassController = new GymPassController(gymPassRepository, gymPassInfoTable, gymPassMapper);  
  
 clientInfoTable.SetTopButtonAction(e -> GymPassesButton\_Click());  
 clientInfoTable.SetBottomButtonAction(e -> ManagersButton\_Click());  
 clientInfoTable.SetAddButtonAction(e -> clientController.Add());  
 clientInfoTable.SetRemoveButtonAction(e -> clientController.Remove());

gymPassInfoTable.SetTopButtonAction(e -> ManagersButton\_Click());  
 gymPassInfoTable.SetBottomButtonAction(e -> ClientsButton\_Click());  
 gymPassInfoTable.SetAddButtonAction(e -> gymPassController.Add());  
 gymPassInfoTable.SetRemoveButtonAction(e -> gymPassController.Remove());  
  
 managerInfoTable.SetTopButtonAction(e -> ClientsButton\_Click());  
 managerInfoTable.SetBottomButtonAction(e -> GymPassesButton\_Click());  
 managerInfoTable.SetAddButtonAction(e -> managerController.Add());  
 managerInfoTable.SetRemoveButtonAction(e -> managerController.Remove());  
  
 managerController.LoadPage(0);  
 managerController.Show();  
 }  
  
 static void GymPassesButton\_Click()  
 {  
 managerController.Hide();  
 clientController.Hide();  
  
 gymPassController.Show();  
 }  
  
 static void ClientsButton\_Click()  
 {  
 managerController.Hide();  
 gymPassController.Hide();  
  
 clientController.Show();  
 }  
  
 static void ManagersButton\_Click()  
 {  
 gymPassController.Hide();  
 clientController.Hide();  
  
 managerController.Show();  
 }  
}

Класс базового репозитория, реализующий CRUD операции и посылающий запрос через jdbc:

package Repositories;  
  
import Connections.ConnectionFactory;  
import Entities.BaseEntity;  
import Mappers.SqlMapper;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.UUID;  
  
public class BaseRepository<TEntity extends BaseEntity>

implements Repository<TEntity>  
{  
 private final String tableName;  
 private final SqlMapper<TEntity> sqlMapper;  
  
 private final String connectionString;  
  
 protected final ConnectionFactory ConnectionFactory =

new ConnectionFactory();  
  
 private Connection connection;  
  
 public BaseRepository(  
 String connectionString,  
 String tableName,  
 SqlMapper<TEntity> sqlMapper)  
 {  
 this.connectionString = connectionString;  
 this.sqlMapper = sqlMapper;  
  
 this.tableName = tableName;  
 }  
  
  
 public void Create(TEntity entity)  
 {  
 StringBuilder sqlQuery = new StringBuilder(

String.format("insert into %s values(", tableName));  
  
 var fields = sqlMapper.GetFields(entity);  
  
 for (var field : fields)  
 {  
 sqlQuery.append(String.format("'%s',", field.Value));  
 }  
  
 sqlQuery.deleteCharAt(sqlQuery.length() - 1);  
 sqlQuery.append(")");  
  
 Execute(sqlQuery.toString());  
  
 CloseConnection();  
 }

public TEntity Read(UUID id)  
 {  
 var sqlQuery = String.format("select \* from %s where Id = '%s'", tableName, id);  
  
 var queryResult = ExecuteQuery(sqlQuery);  
  
 var entity = queryResult != null ? GetNextEntity(queryResult) : null;  
  
 CloseConnection();  
  
 return entity;  
 }  
  
 public ArrayList<TEntity> ReadTop(int amount, int offset)  
 {  
 var sqlQuery = String.format("select \* from %s order by Id offset %s rows fetch next %s rows only", tableName, offset, amount);  
  
 var queryResult = ExecuteQuery(sqlQuery);  
 if (queryResult == null)  
 {  
 return null;  
 }  
  
 var result = new ArrayList<TEntity>();  
 for (int i = 0; i < amount; i++)  
 {  
 var entity = GetNextEntity(queryResult);  
  
 if (entity == null)  
 {  
 break;  
 }  
  
 result.add(entity);  
 }  
  
 CloseConnection();  
  
 return result;  
 }  
 public void Update(TEntity entity)  
 {  
 if (ExistsWithId(entity.Id))  
 {  
 StringBuilder sqlQuery = new StringBuilder(String.format("update %s set ", tableName));  
  
 var fields = sqlMapper.GetFields(entity);  
 for (var field : fields)  
 {  
 sqlQuery.append(String.format("set %s = %s,", field.Name, field.Value));  
 }  
 sqlQuery.deleteCharAt(sqlQuery.length() - 1);  
 sqlQuery.append(String.format("where Id = %s", entity.Id));  
 }  
 CloseConnection();  
 }  
  
 public void Delete(UUID id)  
 {  
 var sqlQuery = String.format("delete from %s where Id = \'%s\'", tableName, id);  
  
 Execute(sqlQuery);  
  
 CloseConnection();  
 }  
  
 public boolean ExistsWithId(UUID id)  
 {  
 var sqlQuery = String.format("if exists

(select \* from Managers where id = '%s')", id) +  
 "select 1 as Exists else select 0 as Exists";  
  
 var resultSet = ExecuteQuery(sqlQuery);  
 try  
 {  
 if (!resultSet.next())  
 {  
 return false;  
 }  
  
 var exists = resultSet.getBoolean("Exists");  
  
 CloseConnection();  
  
 return exists;  
 }  
 catch (SQLException ex)  
 {  
 ex.printStackTrace();  
  
 CloseConnection();  
  
 return false;  
 }  
 }  
  
  
 private TEntity GetNextEntity(ResultSet resultSet)  
 {  
 try  
 {  
 var exists = resultSet.next();  
  
 return exists ? sqlMapper.CreateEntity(resultSet) : null;  
 }  
 catch (SQLException ex)  
 {  
 ex.printStackTrace();  
  
 return null;  
 }  
 }

private ResultSet ExecuteQuery(String sqlQuery)  
 {  
 System.out.println(sqlQuery);  
  
 try  
 {  
 var connection = ConnectionFactory.getConnection(connectionString);  
 var statement = connection.prepareStatement(sqlQuery);  
  
 return statement.executeQuery();  
 }  
 catch (SQLException ex)  
 {  
 ex.printStackTrace();  
 return null;  
 }  
 }  
  
 private void Execute(String sqlQuery)  
 {  
 System.out.println(sqlQuery);  
  
 try  
 {  
 var connection = ConnectionFactory.getConnection(connectionString);  
 var statement = connection.prepareStatement(sqlQuery);  
  
 statement.execute();  
  
 connection.close();  
 }  
 catch (SQLException ex)  
 {  
 ex.printStackTrace();  
 }  
 }  
  
 private void CloseConnection()  
 {  
 try  
 {  
 if(connection != null && !connection.isClosed())  
 {  
 connection.close();  
 }  
 }  
 catch (SQLException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

Класс Базового контроллера:

package Controllers;  
  
import Entities.BaseEntity;  
import Mappers.SqlMapper;  
import Repositories.Repository;  
import Views.AddEntityView;  
import Views.EntityInfoTable;  
import Views.RemoveEntityView;  
  
import javax.swing.\*;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class BaseController<TEntity extends BaseEntity>  
{  
 protected final EntityInfoTable<TEntity> EntityTable;  
 protected final Repository<TEntity> EntityRepository;  
 protected final SqlMapper<TEntity> EntitySqlMapper;  
  
 private int currentPage;  
  
 public BaseController(  
 Repository<TEntity> entityRepository,  
 EntityInfoTable<TEntity> entityTable,  
 SqlMapper<TEntity> entitySqlMapper)  
 {  
 EntityRepository = entityRepository;  
 EntityTable = entityTable;  
 EntitySqlMapper = entitySqlMapper;  
 }  
  
 public void LoadPage(int pageNumber)  
 {  
 currentPage = pageNumber;  
  
 var page = EntityRepository.ReadTop(5, currentPage \* 5);  
  
 EntityTable.SetTableData(page);  
 }  
  
 public void Show()  
 {  
 EntityTable.Show();  
 }  
  
 public void Hide()  
 {  
 EntityTable.Hide();  
 }  
  
 public void Add()  
 {  
 var addEntityView = new AddEntityView<>(EntitySqlMapper);  
 addEntityView.SetExitAction(() ->  
 {  
 var entity = addEntityView.Result;  
 EntityRepository.Create(entity);  
 LoadPage(currentPage);  
 });  
 }  
  
 public void Remove()  
 {  
 var removeEntityView = new RemoveEntityView();  
 removeEntityView.SetExitAction(() ->  
 {  
 var id = removeEntityView.EntityId;  
  
 EntityRepository.Delete(id);  
  
 LoadPage(currentPage);  
 });  
 }  
}

Фабрика подключений:

package Connections;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public class ConnectionFactory  
{  
 Connection connection = null;  
  
 public Connection getConnection(String url)  
 {  
 try  
 {  
 connection = DriverManager.getConnection(url);  
 } catch (SQLException throwables)  
 {  
 throwables.printStackTrace();  
 }  
  
 return connection;  
 }  
}

Интерфейс маппера:

package Mappers;  
  
import Entities.BaseEntity;  
  
import javax.swing.\*;  
import java.sql.ResultSet;  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public interface SqlMapper<TEntity extends BaseEntity>  
{  
 TEntity CreateEntity(ResultSet sql);  
 TEntity CreateEntity(HashMap<String, String> stringHashMap);  
  
 ArrayList<NameValuePair> GetFields(TEntity entity);  
  
 String[] GetFieldNames();  
}

Маппер менеджеров:

package Mappers;  
  
import Entities.Manager;  
  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.UUID;  
  
public class ManagerMapper implements SqlMapper<Manager>  
{  
 @Override  
 public Manager CreateEntity(ResultSet sql)  
 {  
 Manager manager = new Manager();  
  
 try  
 {  
 manager.Id = UUID.fromString(sql.getString("Id"));  
 manager.Name = sql.getString("Name");  
  
 return manager;  
 }  
 catch (SQLException sqlException)  
 {  
 sqlException.printStackTrace();  
 return null;  
 }  
 }  
 @Override  
 public Manager CreateEntity(HashMap<String, String> stringHashMap)  
 {  
 Manager manager = new Manager();  
  
 manager.Id = UUID.fromString(stringHashMap.get("Id"));  
 manager.Name = stringHashMap.get("Name");  
  
 return manager;  
 }  
 @Override  
 public ArrayList<NameValuePair> GetFields(Manager manager)  
 {  
 ArrayList<NameValuePair> result = new ArrayList<>();  
  
 result.add(new NameValuePair("Id", manager.Id.toString()));  
 result.add(new NameValuePair("Name", manager.Name));  
  
 return result;  
 }  
 @Override  
 public String[] GetFieldNames()  
 {  
 return new String[]  
 {  
 "Id",  
 "Name"  
 };  
 }  
}

Окно удаления сущности из базы данных:

package Views;  
  
import Actions.ActionToPerform;  
import javax.swing.\*;  
import java.util.UUID;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class RemoveEntityView  
{  
 private final JFrame Frame;  
  
 private final JTextField TextField;  
  
 private final ExecutorService executorService = Executors.newSingleThreadExecutor();  
 private ActionToPerform exitAction;  
  
 public UUID EntityId;  
  
 public RemoveEntityView()  
 {  
 Frame = new JFrame();  
 Frame.setTitle("Add Entity");  
 Frame.setSize(200, 130);  
 Frame.setLayout(null);  
  
 var label = new JLabel("id", SwingConstants.RIGHT);  
 label.setSize(60, 20);  
 label.setLocation(0, 10);  
  
 TextField = new JTextField();  
 TextField.setSize(100, 20);  
 TextField.setLocation(70, 10);  
  
 Frame.add(label);  
 Frame.add(TextField);  
  
 var saveButton = new JButton("Remove");  
 saveButton.setSize(100, 20);  
 saveButton.setLocation(70, 60);  
 saveButton.addActionListener(e -> RemoveButton\_Click());  
  
 Frame.add(saveButton);  
  
 Frame.setVisible(true);  
 }  
  
 public void SetExitAction(ActionToPerform actionListener)  
 {  
 exitAction = actionListener;  
 }  
  
 private void RemoveButton\_Click()  
 {  
 EntityId = UUID.fromString(TextField.getText());  
 Frame.setVisible(false);  
 executorService.execute(() -> exitAction.PerformAction());  
 }  
}

Окно добавления сущности:

package Views;  
import Actions.ActionToPerform;  
import Entities.BaseEntity;  
import Mappers.SqlMapper;  
import javax.swing.\*;  
import java.util.HashMap;  
import java.util.UUID;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class AddEntityView<TEntity extends BaseEntity>  
{  
 private final JFrame Frame;  
 private final HashMap<String, JTextField> TextFields;  
 private final String[] FieldNames;  
 private final SqlMapper<TEntity> EntitySqlMapper;  
  
 private final ExecutorService executorService = Executors.newSingleThreadExecutor();  
 private ActionToPerform exitAction;  
  
 public TEntity Result;  
  
 public AddEntityView(SqlMapper<TEntity> entitySqlMapper)  
 {  
 EntitySqlMapper = entitySqlMapper;  
  
 FieldNames = entitySqlMapper.GetFieldNames();  
  
 Frame = new JFrame();  
 Frame.setTitle("Add Entity");  
 Frame.setSize(200, 100 + FieldNames.length \* 30);  
 Frame.setLayout(null);  
  
 TextFields = new HashMap<>();  
  
 int y = 10;  
 for (var fieldName : FieldNames)  
 {  
 if(fieldName == "Id")  
 {  
 continue;  
 }  
  
 var label = new JLabel(fieldName, SwingConstants.RIGHT);  
 label.setSize(60, 20);  
 label.setLocation(0, y);  
  
 var textField = new JTextField();  
 textField.setSize(100, 20);  
 textField.setLocation(70, y);  
  
 Frame.add(label);  
 Frame.add(textField);  
  
 TextFields.put(fieldName, textField);  
  
 y += 30;  
 }  
  
 var saveButton = new JButton("Add");  
 saveButton.setSize(100, 20);  
 saveButton.setLocation(70, y + 20);  
 saveButton.addActionListener(e -> SaveButton\_Click());  
  
 Frame.add(saveButton);  
  
 Frame.setVisible(true);  
 }  
  
 public void SetExitAction(ActionToPerform actionListener)  
 {  
 exitAction = actionListener;  
 }  
  
 private void SaveButton\_Click()  
 {  
 var hashMap = new HashMap<String, String>();  
  
 for(var fieldName : FieldNames)  
 {  
 if(fieldName == "Id")  
 {  
 continue;  
 }  
  
 var field = TextFields.get(fieldName);  
  
 hashMap.put(fieldName, field.getText());  
 }  
  
 hashMap.put("Id", UUID.randomUUID().toString());  
  
 Result = EntitySqlMapper.CreateEntity(hashMap);  
  
 Frame.setVisible(false);  
  
 executorService.execute(() -> exitAction.PerformAction());  
 }  
}

Класс таблицы:

package Views;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.ActionListener;  
  
public class InfoTable  
{  
 private final Object[] columnNames;  
 private final JFrame Frame;  
  
 private final JButton TopButton;  
 private final JButton BottomButton;  
  
 private final JButton AddButton;  
 private final JButton RemoveButton;  
  
 private JScrollPane ScrollPane;  
 private JTable StudentInfoTable;  
  
  
 InfoTable(  
 String title,  
 String[] columns,  
 String topButtonLabel,  
 String bottomButtonLabel)  
 {  
 Frame = new JFrame();  
 Frame.setTitle(title);  
  
 TopButton = new JButton(topButtonLabel);  
 TopButton.setSize(100, 20);  
 TopButton.setLocation(10, 110);  
  
 BottomButton = new JButton(bottomButtonLabel);  
 BottomButton.setSize(100, 20);  
 BottomButton.setLocation(10, 140);  
  
 AddButton = new JButton("Add");  
 AddButton.setSize(100, 20);  
 AddButton.setLocation(150, 110);  
  
 RemoveButton = new JButton("Remove");  
 RemoveButton.setSize(100, 20);  
 RemoveButton.setLocation(150, 140);  
  
 Frame.add(TopButton);  
 Frame.add(BottomButton);  
 Frame.add(AddButton);  
 Frame.add(RemoveButton);  
  
 DefaultTableModel tableModel = new DefaultTableModel();  
  
 for (var column : columns)  
 {  
 tableModel.addColumn(column);  
 }  
  
 columnNames = columns;  
  
 StudentInfoTable = new JTable(tableModel);  
 StudentInfoTable.setBounds(30, 40, 200, 300);  
  
 ScrollPane = new JScrollPane(StudentInfoTable);  
 Frame.add(ScrollPane);  
 Frame.setSize(500, 400);  
 }  
 public void SetTableData(String[][] tableData)  
 {  
 StudentInfoTable = new JTable(tableData, columnNames);  
 StudentInfoTable.setBounds(30, 40, 200, 300);  
  
 Frame.remove(ScrollPane);  
 ScrollPane = new JScrollPane(StudentInfoTable);  
 Frame.add(ScrollPane);  
  
 Frame.update(Frame.getGraphics());  
 Frame.revalidate();  
 Frame.repaint();  
 }  
 public void AddRow(String[] row)  
 {  
 var model = (DefaultTableModel) StudentInfoTable.getModel();  
  
 var rowAmount = model.getDataVector().size();  
 model.addRow(row);  
 model.fireTableRowsUpdated(rowAmount - 2, rowAmount - 1);  
 }  
 public String[] GetRowAt(int index, int column)  
 {  
 var model = (DefaultTableModel) StudentInfoTable.getModel();  
 var result = model.getValueAt(index, column);  
  
 return (String[]) result;  
 }  
 public void Show()  
 {  
 Frame.setVisible(true);  
 }  
 public void Hide()  
 {  
 Frame.setVisible(false);  
 }  
 public void SetTopButtonAction(ActionListener actionListener)  
 {  
 TopButton.addActionListener(actionListener);  
 }  
 public void SetBottomButtonAction(ActionListener actionListener)  
 {  
 BottomButton.addActionListener(actionListener);  
 }  
 public void SetAddButtonAction(ActionListener actionListener)  
 {  
 AddButton.addActionListener(actionListener);  
 }  
  
 public void SetRemoveButtonAction(ActionListener actionListener)  
 {  
 RemoveButton.addActionListener(actionListener);  
 }