package controller.main;  
  
import controller.mythread.MyThread;  
import java.io.IOException;  
import java.net.ServerSocket;  
  
public class MainS {  
  
 public static int *port* = 1122;  
  
 public static void main(String[] args) {  
 try {  
 ServerSocket serverSocket = new ServerSocket(*port*);  
 while (true) {  
 *//serverSocket = new ServerSocket(port);* Object ob = new Object();  
 Thread.*sleep*(1000);  
 MyThread myThread = new MyThread(serverSocket,ob);  
 myThread.start();  
 *// myThread.join();* }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

package controller.factory;  
  
import java.io.IOException;  
  
public interface IController {  
  
 void saveDate(String msg) throws IOException, ClassNotFoundException;  
  
 void editDate(String msg) throws IOException, ClassNotFoundException;  
  
 void deleteDate(String msg) throws IOException, ClassNotFoundException;  
  
 void getDate(String msg) throws IOException, ClassNotFoundException;  
  
 void start();  
}

package controller.factory;  
  
import controller.AdminController;  
import controller.ClientController;  
  
*/\*\*  
 \* @author Ataeyv I.M.  
 \* Factory pattern  
 \* \*/*public class FactoryController {  
 public static IController getType(String type) {  
 switch (type) {  
 case "admin":  
 return new AdminController();  
 case "client":  
 return new ClientController();  
 default:  
 throw new RuntimeException();  
 }  
 }  
}

package controller.mythread;  
  
import controller.ServerController;  
import java.net.ServerSocket;  
  
*/\*\*  
 \* @author Atayev I.M.  
 \* \*/*public class MyThread extends Thread {  
  
 private Object locker;  
 private ServerSocket serverSocket;  
 public MyThread(ServerSocket serverSocket) {  
 this.serverSocket = serverSocket;  
 }  
 public MyThread(ServerSocket serverSocket,Object locker) {  
 this.serverSocket = serverSocket;  
 this.locker = locker;  
 }  
  
 @Override  
 public void run() {  
 synchronized (this.locker){  
 ServerController serverController = new ServerController();  
 serverController.work(serverSocket);  
 }  
 }  
}

package controller;  
  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.myexception.MyException;  
import controller.factory.FactoryController;  
import controller.factory.IController;  
import model.bd.dbhclient.DBHClient;  
import model.bd.idbhandler.IDBHandler;  
import java.net.ServerSocket;  
import java.util.ArrayList;  
  
public class ServerController {  
  
 private static int *countClient* = 0;  
 public static Connect *connect*;  
  
 public ServerController(ServerSocket serverSocket) {  
 this.*connect* = new Connect(serverSocket);  
 *countClient*++;  
 }  
  
 public ServerController() {  
 }  
  
 public ServerController(Connect connect) {  
 this.*connect* = connect;  
 *countClient*++;  
 }  
  
  
 public void work(ServerSocket serverSocket) {  
 try {  
  
 IDBHandler idbHandler = new DBHClient();  
 *connect* = new Connect(serverSocket);  
  
 System.*out*.println("Client connect --> " + ++*countClient*);  
  
  
 while (true) {  
 String req = *connect*.readLine();  
 switch (req) {  
 case "signIn": {  
 String login = *connect*.readLine();  
 String pass = *connect*.readLine();  
  
ArrayList<Client> clients = rrayList<Client>)idbHandler.getList().clone();  
  
 for (Client client : clients) {  
 System.*out*.println(client.toString());  
if (pass.equals(client.getPassword()) && login.equals(client.getLogin())) {  
 *connect*.writeLine("true");  
  
 if (client.getFlag() == 1) {  
  
 *connect*.writeLine("adminUI");  
IController iController = FactoryController.*getType*("admin");  
 iController.start();  
 return;  
  
 } else if (client.getFlag() == 2) {  
 System.*out*.println(client.toString());  
 *connect*.writeLine("clientUI");  
 *connect*.writeObj(client);  
  
IController iController = FactoryController.*getType*("client");  
 iController.start();  
 return;  
  
 } else {  
new MyException("do not flags please view database and class Client");  
 break;  
 }  
 }  
 }  
 *connect*.writeLine("false");  
 *connect*.writeLine("false");  
 break;  
 }  
 case "signUp": {  
 if (idbHandler.addObj(*connect*.readObj())) {  
 *connect*.writeLine("true");  
 } else {  
 *connect*.writeLine("false");  
 }  
 break;  
 }  
 default: {

new MyException("class ServerController switch(connect.readLine()) error"); break;  
 }  
 }  
 }  
 } catch (Exception e) {  
 new MyException(e);  
 }  
 }  
}

package controller;  
  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.myexception.MyException;  
import com.example.model.order.Order;  
import com.example.model.tour.Tour;  
import controller.factory.IController;  
import model.bd.dbhclient.DBHClient;  
import model.bd.dbhorder.DBHOrder;  
import model.bd.dbhticket.DBHTicket;  
import model.bd.dbhtour.DBHTour;  
import model.bd.idbhandler.IDBHandler;  
import model.delete.Delete;  
import java.io.IOException;  
import java.util.ArrayList;  
  
public class ClientController implements IController {  
  
 public Connect connect = ServerController.*connect*;  
 private IDBHandler idbHandler = new DBHClient();  
 private IDBHandler idbHandlerTour = new DBHTour();  
 private IDBHandler idbHandlerOrder = new DBHOrder();  
 private IDBHandler idbHandlerTicket = new DBHTicket();  
  
  
 @Override  
 public void saveDate(String msg) throws IOException, ClassNotFoundException {  
 switch (msg) {  
 case "orderTour": {  
 String tourCode = connect.readLine();  
 Client client = (Client) connect.readObj();  
 System.*out*.println("order client -> " + client);  
 if (makeOrderTour(tourCode, client, idbHandlerTour.getList())) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 default: {  
 new MyException("клиент сервер заказ тура case default ");  
 break;  
 }  
 }  
 }  
  
 @Override  
 public void editDate(String msg) throws IOException, ClassNotFoundException {  
 }  
  
 @Override  
 public void deleteDate(String msg) throws IOException, ClassNotFoundException {  
 switch (msg) {  
 case "deleteOrder": {  
 String id = connect.readLine();  
 Delete delete = new Delete();  
 if (delete.deleteOrder(Integer.*parseInt*(id), idbHandlerOrder.getList())) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 default: {  
 new MyException("client server do not response deleteOrder case default");  
 break;  
 }  
 }  
 }  
  
 @Override  
 public void getDate(String msg) throws IOException, ClassNotFoundException {  
 switch (msg) {  
 case "viewUser": {  
 connect.writeObjList(idbHandler.getList());  
 break;  
 }  
 case "viewTicket": {  
 connect.writeObjList(idbHandlerTicket.getList());  
 break;  
 }  
 case "viewTour": {  
 connect.writeObjList(idbHandlerTour.getList());  
 break;  
 }  
 case "viewOrder": {  
 final String clientCode = connect.readLine();  
 boolean flagOrder = checkOrderClient(clientCode, idbHandlerOrder.getList());  
  
 if (flagOrder) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
  
 if (flagOrder) {  
 ArrayList<Object> objects = getClientOrder(clientCode, idbHandlerOrder.getList());  
 connect.writeObjList(objects);  
 }  
 break;  
 }  
 default: {  
 new MyException("client server view default case ");  
 break;  
 }  
 }  
 }  
  
  
 @Override  
 public void start() {  
 System.*out*.println("start client controller");  
 try {  
 while (true) {  
 String msg = connect.readLine();  
 switch (msg) {  
 case "view": {  
 this.getDate(connect.readLine());  
 break;  
 }  
 case "add": {  
 this.saveDate(connect.readLine());  
 break;  
 }  
 case "delete": {  
 this.deleteDate(connect.readLine());  
 break;  
 }  
 case "edit": {  
 this.editDate(connect.readLine());  
 break;  
 }  
 case "close": {  
 connect.close();  
 return;  
 }  
 default: {  
 msg = null;  
 *//connect.clearConnect();* new MyException("поличичли что-то не то client controller ");  
 break;  
 }  
 }  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 } catch (ClassNotFoundException e) {  
 new MyException(e);  
 }  
 }  
  
  
 private boolean makeOrderTour(String tourCode, Client client, ArrayList<Object> objects) {  
 ArrayList<Tour> tours = (ArrayList<Tour>) idbHandlerTour.getList().clone();  
 if (checkTour(tourCode, tours)) {  
 for (Tour t : tours) {  
 if (tourCode.equals(t.getTourCode())) {  
 Order order = new Order();  
 order.setClientCode(client.getClientCode());  
 order.setTourCode(t.getTourCode());  
 return idbHandlerOrder.addObj(order);  
 }  
 }  
 }  
 return false;  
 }  
  
 private ArrayList<Object> getClientOrder(String clientCode, ArrayList<Object> objects) {  
 ArrayList<Order> orders = new ArrayList<>();  
 ArrayList<Order> orderArrayList = (ArrayList<Order>) objects.clone();  
 for (Order o : orderArrayList) {  
 if (clientCode.equals(o.getClientCode())) {  
 orders.add(o);  
 }  
 }  
 return (ArrayList<Object>) orders.clone();  
 }  
  
 private boolean checkTour(String tourCode, ArrayList<Tour> tours) {  
 for (Tour t : tours) {  
 if (tourCode.equals(t.getTourCode())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkOrderClient(String clientCode, ArrayList<Object> objects) {  
 ArrayList<Order> orders = (ArrayList<Order>) objects.clone();  
 for (Order o : orders) {  
 if (clientCode.equals(o.getClientCode())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
}

package controller;  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.myexception.MyException;  
import com.example.model.order.Order;  
import com.example.model.ticket.Ticket;  
import com.example.model.tour.Tour;  
import controller.factory.IController;  
import model.bd.dbhclient.DBHClient;  
import model.bd.dbhorder.DBHOrder;  
import model.bd.dbhticket.DBHTicket;  
import model.bd.dbhtour.DBHTour;  
import model.bd.idbhandler.IDBHandler;  
import model.delete.Delete;  
import java.io.IOException;  
import java.util.ArrayList;  
  
public class AdminController implements IController {  
  
 public Connect connect = ServerController.*connect*;  
 private IDBHandler idbHandler = new DBHClient();  
 private IDBHandler idbHandlerTour = new DBHTour();  
 private IDBHandler idbHandlerOrder = new DBHOrder();  
 private IDBHandler idbHandlerTicket = new DBHTicket();  
  
 @Override  
 public void saveDate(String msg) throws IOException, ClassNotFoundException {  
 switch (msg) {  
 case "addUser": {  
 boolean flagAddClient = idbHandler.addObj(connect.readObj());  
 if (flagAddClient) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "addTour": {  
 boolean flagAddTour = idbHandlerTour.addObj(connect.readObj());  
 if (flagAddTour) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "addTicket": {  
 String idOrder = connect.readLine();  
 Ticket ticket = (Ticket) connect.readObj();  
boolean flag = makeOrder(Integer.parseInt(idOrder), ticket);  
 if (flag) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 }  
 }  
  
 @Override  
 public void editDate(String msg) throws IOException, ClassNotFoundException {  
 switch (msg) {  
 case "editUser": {  
 if (idbHandler.editObj(connect.readObj())) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "editTour": {  
 Object ob = connect.readObj();  
 if (idbHandlerTour.editObj(ob)) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 }  
 }  
  
 @Override  
 public void deleteDate(String msg) throws IOException, ClassNotFoundException {  
 switch (msg) {  
 case "deleteUser": {  
 String login = connect.readLine();  
 String pass = connect.readLine();  
 String clientCode = connect.readLine();  
 Delete delete = new Delete();  
 if (delete.deleteUser(login, pass, clientCode, idbHandler.getList())) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "deleteTour": {  
 String idTour = connect.readLine();  
 Delete delete = new Delete();  
 if (delete.deleteTour(Integer.parseInt(idTour), idbHandlerTour.getList())) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "ticketDelete": {  
 String idTicket = connect.readLine();  
 Delete delete = new Delete();  
 if (delete.deleteTicket(Integer.parseInt(idTicket), idbHandlerTicket.getList())) {  
 connect.writeLine("true");  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 } }  
  
 @Override  
 public void getDate(String msg) throws IOException {  
 switch (msg) {  
 case "viewUser": {  
 connect.writeObjList(idbHandler.getList());  
 break;  
 }  
 case "viewTicket": {  
 connect.writeObjList(idbHandlerTicket.getList());  
 break;  
 }  
 case "viewTour": {  
 connect.writeObjList(idbHandlerTour.getList());  
 break;  
 }  
 case "viewOrder": {  
 connect.writeObjList(idbHandlerOrder.getList());  
 break;  
 }  
 }  
 }  
  
 @Override  
 public void start() {  
 System.*out*.println("start admin controller");  
 try {  
 while (true) {  
 String msg = connect.readLine();  
 switch (msg) {  
 case "view": {  
 this.getDate(connect.readLine());  
 break;  
 }  
 case "add": {  
 this.saveDate(connect.readLine());  
 break;  
 }  
 case "delete": {  
 this.deleteDate(connect.readLine());  
 break;  
 }  
 case "search": {  
 String req = connect.readLine();  
 this.search(req);  
 req = null;  
 break;  
 }  
 case "edit": {  
 this.editDate(connect.readLine());  
 break;  
 }  
 case "close": {  
 connect.close();  
 return;  
 }  
 default: {  
 msg = null;  
 new MyException("поличичли что-то не то то client controller ");  
 break;  
 }  
 }  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 }  
  
  
  
  
  
  
 private void search(String msg) throws IOException {  
 switch (msg) {  
 case "searchUser": {  
 String fio = connect.readLine();  
 String login = connect.readLine();  
 String pass = connect.readLine();  
 int counter = 0;  
 ArrayList<Client> list = (ArrayList<Client>) idbHandler.getList().clone();  
 for (Client c : list) {  
 if (fio.equals(c.getFIO()) && login.equals(c.getLogin()) && pass.equals(c.getPassword())) {  
 ++counter;  
 connect.writeLine("true");  
 connect.writeObj(c);  
 }  
 }  
 if (counter == 0) {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "searchClient": {  
 final String clientCode = connect.readLine();  
 if (checkOrderClientCode(clientCode)) {  
 ArrayList<Object> objects = getSearchOrders(clientCode);  
 connect.writeLine("true");  
 connect.writeObjList(objects);  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 case "searchTour": {  
 final String tourCode = connect.readLine();  
 if (checkOrderTourCode(tourCode)) {  
 ArrayList<Object> objects = getSearchOrdersTour(tourCode);  
 connect.writeLine("true");  
 connect.writeObjList(objects);  
 } else {  
 connect.writeLine("false");  
 }  
 break;  
 }  
 }  
 }private boolean checkOrderTourCode(String tourCode) {  
 ArrayList<Order> orders = (ArrayList<Order>) idbHandlerOrder.getList().clone();  
 for (Order o : orders) {  
 if (tourCode.equals(o.getTourCode())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkOrderClientCode(String clientCode) {  
 ArrayList<Order> orders = (ArrayList<Order>) idbHandlerOrder.getList().clone();  
 for (Order o : orders) {  
 if (clientCode.equals(o.getClientCode())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkClient(String clientCode,ArrayList<Client> clients) {  
 for (Client c : clients) {  
 if (clientCode.equals(c.getClientCode())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkTour(String tourCode, ArrayList<Tour> tours) {  
 for (Tour t : tours) {  
 if (tourCode.equals(t.getTourCode())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkTicket(int id) {  
 ArrayList<Ticket> tickets = (ArrayList<Ticket>) idbHandlerTicket.getList().clone();  
 for (Ticket t : tickets) {  
 if (id == t.getId()) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 */\*\*  
 \* 1-й исход работы) Возврашаем Order из бд прохотимся по списку заказов, и если есть такоей id,  
 \* то вызываем запрос на добавление, если билет создан успешно, то возврашаем (CreateTicket),  
 \* иначе (NoCreateTicket).  
 \* <p>  
 \* 2-й исход работы) Создаём счетчик int i  
 \* Возврашаем Order из бд прохотимся по списку заказов, если i равен нулю, то (NoOrder)  
 \* если i больше и менше или равно Order.size то,(NoIdOrder) нету такого заказа.  
 \* @param idOrder  
 \* @param ticket  
 \*/* private boolean makeOrder(int idOrder, Ticket ticket) {  
  
 boolean flagAddTicket;  
 boolean flagClient;  
 boolean flagTour;  
  
 ArrayList<Tour> tourArrayList = (ArrayList<Tour>) idbHandlerTour.getList().clone();  
 ArrayList<Order> orderArrayList = (ArrayList<Order>) idbHandlerOrder.getList().clone();  
 ArrayList<Client> clientArrayList = (ArrayList<Client>) idbHandler.getList().clone();  
  
 for (Order o : orderArrayList) {  
 if (idOrder == o.getId()) {  
 flagClient = checkClient(o.getClientCode(), clientArrayList);  
 flagTour = checkTour(o.getTourCode(), tourArrayList);  
 if (flagClient == true && flagTour == true) {  
 ticket.setUserCode(o.getClientCode());  
 for (Tour t : tourArrayList) {  
 if (o.getTourCode().equals(t.getTourCode())) {  
 ticket.setDepartureData(t.getTourDate());  
 ticket.setArrivalPoint(t.getCountryName() + "-" + t.getCityName());  
 flagAddTicket = idbHandlerTicket.addObj(ticket);  
 boolean ff = idbHandlerOrder.deleteObj(o);  
 System.out.println("i am delete order flag -> " + ff);  
 return flagAddTicket;  
 }  
 }  
 } else {  
 return false;  
 }  
 }  
 }  
 return false;  
 }  
  
 private ArrayList<Object> getSearchOrders(String clientCode) {  
 ArrayList<Order> orders = (ArrayList<Order>) idbHandlerOrder.getList().clone();  
 ArrayList<Object> objects = new ArrayList<>();  
 for (Order o : orders) {  
 if (clientCode.equals(o.getClientCode())) {  
 objects.add(o);  
 }  
 }  
 return objects;  
 }  
  
 private ArrayList<Object> getSearchOrdersTour(String tourCode) {  
 ArrayList<Order> orders = (ArrayList<Order>) idbHandlerOrder.getList().clone();  
 ArrayList<Object> objects = new ArrayList<>();  
 for (Order o : orders) {  
 if (tourCode.equals(o.getTourCode())) {  
 objects.add(o);  
 }  
 }  
 return objects;  
 }  
}

package model.bd.dbhclient;  
  
import com.example.model.client.Client;  
import com.example.model.myexception.MyException;  
import model.bd.idbhandler.IDBHandler;  
import model.configs.clientBD.ConstClient;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
  
public class DBHClient implements IDBHandler {  
  
 @Override  
 public boolean addObj(Object obj) {  
 try {  
 Client client = (Client) obj;  
 String insert = "INSERT INTO " + ConstClient.CLIENT\_TABLE + "("  
 + ConstClient.CLIENT\_FIO + ","  
 + ConstClient.CLIENT\_CODE + ","  
 + ConstClient.CLIENT\_PASSPORT\_ID + ","  
 + ConstClient.CLIENT\_MAIL + ","  
 + ConstClient.CLIENT\_MOBILE\_NUMBER + ","  
 + ConstClient.CLIENT\_LOGIN + ","  
 + ConstClient.CLIENT\_PASSWORD + ","  
 + ConstClient.CLIENT\_FLAG + ")" + "VALUES(?,?,?,?,?,?,?,?)";  
  
 PreparedStatement prSt = getDbConnection().prepareStatement(insert);  
 prSt.setString(1, client.getFIO());  
 prSt.setString(2, client.getClientCode());  
 prSt.setString(3, client.getPassportId());  
 prSt.setString(4, client.getMail());  
 prSt.setString(5, client.getMobileNumber());  
 prSt.setString(6, client.getLogin());  
 prSt.setString(7, client.getPassword());  
 prSt.setInt(8, client.getFlag());  
 prSt.executeUpdate();  
 } catch (ClassNotFoundException | SQLException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public ArrayList<Object> getList() {  
 ArrayList<Object> arrayList = new ArrayList<>();  
 try {  
 String select = "SELECT \* FROM " + ConstClient.CLIENT\_TABLE;  
 Statement statement = getDbConnection().createStatement();  
 ResultSet resSet = statement.executeQuery(select);  
 while (resSet.next()) {  
 Client client = new Client();  
 client.setId(resSet.getInt(1));  
 client.setFIO(resSet.getString(2));  
 client.setClientCode(resSet.getString(3));  
 client.setPassportId(resSet.getString(4));  
 client.setMail(resSet.getString(5));  
 client.setMobileNumber(resSet.getString(6));  
 client.setLogin(resSet.getString(7));  
 client.setPassword(resSet.getString(8));  
 client.setFlag(resSet.getInt(9));  
 arrayList.add(client);  
 }  
 } catch (SQLException e) {  
 new MyException(e);  
 } catch (ClassNotFoundException e) {  
 new MyException(e);  
 }  
 return arrayList;  
 }  
  
 @Override  
 public boolean deleteObj(Object obj) {  
 try {  
 Client client = (Client) obj;  
 String DELETE = "DELETE FROM " + ConstClient.CLIENT\_TABLE + " WHERE " + ConstClient.CLIENT\_ID + "='" + client.getId() + "'; ";  
 PreparedStatement preparedStatementDelete = getDbConnection().prepareStatement(DELETE);  
 preparedStatementDelete.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public boolean editObj(Object obj) {  
 try {  
 Client client = (Client) obj;  
 String update = "UPDATE " + ConstClient.CLIENT\_TABLE +  
 " SET " + ConstClient.CLIENT\_FIO + "=?, " + ConstClient.CLIENT\_CODE + "=?, " + ConstClient.CLIENT\_PASSPORT\_ID + "=?, " + ConstClient.CLIENT\_MAIL + "=?, " + ConstClient.CLIENT\_MOBILE\_NUMBER + "=?, " + ConstClient.CLIENT\_LOGIN + "=?, " + ConstClient.CLIENT\_PASSWORD + "=?, " + ConstClient.CLIENT\_FLAG + "=? " +  
 " WHERE " + ConstClient.CLIENT\_ID + "=?";  
  
 PreparedStatement preparedStatement = getDbConnection().prepareStatement(update);  
  
 preparedStatement.setString(1, client.getFIO());  
 preparedStatement.setString(2, client.getClientCode());  
 preparedStatement.setString(3, client.getPassportId());  
 preparedStatement.setString(4, client.getMail());  
 preparedStatement.setString(5, client.getMobileNumber());  
 preparedStatement.setString(6, client.getLogin());  
 preparedStatement.setString(7, client.getPassword());  
 preparedStatement.setInt(8, client.getFlag());  
 preparedStatement.setInt(9, client.getId());  
  
 preparedStatement.executeUpdate();  
  
 } catch (SQLException | ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
}

package model.bd.dbhorder;  
  
import com.example.model.myexception.MyException;  
import com.example.model.order.Order;  
import model.bd.idbhandler.IDBHandler;  
import model.configs.orderBD.ConstOrder;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
  
*/\*\*  
 \* @author Ataeyv I.M.  
 \* @implNote IDBHandler  
 \* \*/*public class DBHOrder implements IDBHandler {  
  
 @Override  
 public boolean addObj(Object obj) {  
 try {  
 Order order = (Order) obj;  
 String insert = "INSERT INTO " + ConstOrder.ORDER\_TABLE + "(*"* + ConstOrder.ORDER\_CLIENT\_CODE + ",*"* + ConstOrder.ORDER\_TOUR\_CODE + ")" + "VALUES(?,?)";  
  
 PreparedStatement prSt = getDbConnection().prepareStatement(insert);  
 prSt.setString(1, order.getClientCode());  
 prSt.setString(2, order.getTourCode());  
 prSt.executeUpdate();  
  
 } catch (ClassNotFoundException | SQLException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public ArrayList<Object> getList() {  
 ArrayList<Object> arrayList = new ArrayList<>();  
 try {  
 String select = "SELECT \* FROM " + ConstOrder.*ORDER\_TABLE*;  
 Statement statement = getDbConnection().createStatement();  
 ResultSet resSet = statement.executeQuery(select);  
 while (resSet.next()) {  
 Order order = new Order();  
 order.setId(resSet.getInt(1));  
 order.setClientCode(resSet.getString(2));  
 order.setTourCode(resSet.getString(3));  
 arrayList.add(order);  
 }  
 } catch (ClassNotFoundException | SQLException e) {  
 new MyException(e);  
 }  
 return arrayList;  
 }  
  
 @Override  
 public boolean deleteObj(Object obj) {  
 try {  
 Order order = (Order) obj;  
 String DELETE = "DELETE FROM " + ConstOrder.*ORDER\_TABLE* + " WHERE " + ConstOrder.*ORDER\_ID* + "*=*'" + order.getId() + "'; ";  
 PreparedStatement preparedStatementDelete = getDbConnection().prepareStatement(DELETE);  
 preparedStatementDelete.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public boolean editObj(Object obj) {  
 return false;  
 }  
}

package model.bd.dbhticket;  
  
import com.example.model.client.Client;  
import com.example.model.myexception.MyException;  
import com.example.model.ticket.Ticket;  
import model.bd.idbhandler.IDBHandler;  
import model.configs.clientBD.ConstClient;  
import model.configs.ticketBD.ConstTicket;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
  
*/\*\*  
 \* @author Ataeyv I.M. (ataewisma@gmail.com)  
 \* \*/*public class DBHTicket implements IDBHandler {  
  
 @Override  
 public boolean addObj(Object obj) {  
 try {  
 Ticket ticket = (Ticket) obj;  
 String insert = "INSERT INTO " + ConstTicket.*TICKET\_TABLE* + "(*"* + ConstTicket.*TICKET\_CODE* + ",*"* + ConstTicket.*TICKET\_USER\_CODE* + ",*"* + ConstTicket.*TICKET\_TRANSPORT\_TYPE* + ",*"* + ConstTicket.*TICKET\_DEPARTURE\_POINT* + ","  
 + ConstTicket.*TICKET\_ARRIVAL\_POINT* + "*,*"  
 + ConstTicket.*TICKET\_DEPARTURE\_DATA* + "*)*" + "VALUES(?,?,?,?,?,?)";  
  
 PreparedStatement prSt = getDbConnection().prepareStatement(insert);  
 prSt.setString(1, ticket.getTicketCode());  
 prSt.setString(2, ticket.getUserCode());  
 prSt.setString(3, ticket.getTransportType());  
 prSt.setString(4, ticket.getDeparturePoint());  
 prSt.setString(5, ticket.getArrivalPoint());  
 prSt.setString(6, ticket.getDepartureData());  
  
 prSt.executeUpdate();  
 } catch (ClassNotFoundException | SQLException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
  
 }  
  
 @Override  
 public ArrayList<Object> getList() {  
 ArrayList<Object> arrayList = new ArrayList<>();  
 try {  
 String select = "SELECT \* FROM " + ConstTicket.TICKET\_TABLE;  
 Statement statement = getDbConnection().createStatement();  
 ResultSet resSet = statement.executeQuery(select);  
 while (resSet.next()) {  
 Ticket t = new Ticket();  
 t.setId(resSet.getInt(1));  
 t.setTicketCode(resSet.getString(2));  
 t.setUserCode(resSet.getString(3));  
 t.setTransportType(resSet.getString(4));  
 t.setDeparturePoint(resSet.getString(5));  
 t.setArrivalPoint(resSet.getString(6));  
 t.setDepartureData(resSet.getString(7));  
 arrayList.add(t);  
 }  
 } catch (SQLException e) {  
 new MyException(e);  
 } catch (ClassNotFoundException e) {  
 new MyException(e);  
 }  
 return arrayList;  
 }  
  
 @Override  
 public boolean deleteObj(Object obj) {  
 try {  
 Ticket ticket = (Ticket) obj;  
 String DELETE = "DELETE FROM " + ConstTicket.TICKET\_TABLE + " WHERE " + ConstTicket.TICKET\_ID + "*=*'" + ticket.getId() + "'; ";  
 PreparedStatement preparedStatementDelete = getDbConnection().prepareStatement(DELETE);  
 preparedStatementDelete.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public boolean editObj(Object obj) {  
 return false;  
 }  
}

package model.bd.dbhtour;  
  
import com.example.model.myexception.MyException;  
import com.example.model.tour.Tour;  
import model.bd.idbhandler.IDBHandler;  
import model.configs.clientBD.ConstClient;  
import model.configs.tourBD.ConstTour;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
  
public class DBHTour implements IDBHandler {  
  
 @Override  
 public boolean addObj(Object obj) {  
 try {  
 Tour tour = (Tour) obj;  
 String insert = "INSERT INTO " + ConstTour.TOUR\_TABLE + "("  
 + ConstTour.TOUR\_COUNTRY\_NAME + ","  
 + ConstTour.TOUR\_CITY\_NAME + ","  
 + ConstTour.TOUR\_PRICE + ","  
 + ConstTour.TOUR\_DURATION + ","  
 + ConstTour.TOUR\_CODE + ","  
 + ConstTour.TOUR\_DATE + ","  
 + ConstTour.TOUR\_NAME + ","  
 + ConstTour.TOUR\_TYPE + ")" + "VALUES(?,?,?,?,?,?,?,?)";  
  
  
 PreparedStatement prSt = getDbConnection().prepareStatement(insert);  
 prSt.setString(1, tour.getCountryName());  
 prSt.setString(2, tour.getCityName());  
 prSt.setFloat(3, tour.getPrice());  
 prSt.setString(4, tour.getDuration());  
 prSt.setString(5, tour.getTourCode());  
 prSt.setString(6, tour.getTourDate());  
 prSt.setString(7, tour.getTourName());  
 prSt.setString(8, tour.getTourType());  
  
 prSt.executeUpdate();  
  
 } catch (SQLException e) {  
 new MyException(e);  
 return false;  
 } catch (ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public ArrayList<Object> getList() {  
 ArrayList<Object> arrayList = new ArrayList<>();  
 try {  
 String select = "SELECT \* FROM " + ConstTour.TOUR\_TABLE;  
 Statement statement = getDbConnection().createStatement();  
 ResultSet resSet = statement.executeQuery(select);  
 while (resSet.next()) {  
 Tour t = new Tour();  
 t.setId(resSet.getInt(1));  
 t.setCountryName(resSet.getString(2));  
 t.setCityName(resSet.getString(3));  
 t.setPrice(resSet.getFloat(4));  
 t.setDuration(resSet.getString(5));  
 t.setTourCode(resSet.getString(6));  
 t.setTourDate(resSet.getString(7));  
 t.setTourName(resSet.getString(8));  
 t.setTourType(resSet.getString(9));  
 arrayList.add(t);  
 }  
 } catch (SQLException e) {  
 new MyException(e);  
 } catch (ClassNotFoundException e) {  
 new MyException(e);  
 }  
 return arrayList;  
 }  
  
 @Override  
 public boolean deleteObj(Object obj) {  
 try {  
 Tour tour = (Tour) obj;  
 String DELETE = "DELETE FROM " + ConstTour.TOUR\_TABLE + " WHERE " + ConstTour.TOUR\_ID + "='" + tour.getId() + "'; ";  
 PreparedStatement preparedStatementDelete = getDbConnection().prepareStatement(DELETE);  
 preparedStatementDelete.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public boolean editObj(Object obj) {  
 try {  
 Tour t = (Tour) obj;  
 System.out.println(t);  
 String update = "UPDATE " + ConstTour.TOUR\_TABLE +  
 " SET "  
 + ConstTour.TOUR\_COUNTRY\_NAME + "=?, "  
 + ConstTour.TOUR\_CITY\_NAME + "=?, "  
 + ConstTour.TOUR\_PRICE + "=?, "  
 + ConstTour.TOUR\_DURATION + "=?, "  
 + ConstTour.TOUR\_CODE + "=?, "  
 + ConstTour.TOUR\_DATE + "=?, "  
 + ConstTour.TOUR\_NAME + "=?, "  
 + ConstTour.TOUR\_TYPE + "=? " +  
 " WHERE " + ConstTour.TOUR\_ID + "=?";  
  
 PreparedStatement preparedStatement = getDbConnection().prepareStatement(update);  
  
 preparedStatement.setString(1, t.getCountryName());  
 preparedStatement.setString(2, t.getCityName());  
 preparedStatement.setFloat(3, t.getPrice());  
 preparedStatement.setString(4, t.getDuration());  
 preparedStatement.setString(5, t.getTourCode());  
 preparedStatement.setString(6, t.getTourDate());  
 preparedStatement.setString(7, t.getTourName());  
 preparedStatement.setString(8, t.getTourType());  
 preparedStatement.setInt(9, t.getId());  
  
 preparedStatement.executeUpdate();  
  
 } catch (SQLException | ClassNotFoundException e) {  
 new MyException(e);  
 return false;  
 }  
 return true;  
 }  
}

package model.bd.idbhandler;  
  
import model.configs.constBD.Const;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.util.ArrayList;  
  
public interface IDBHandler {  
  
 Const *tempConst* = new Const();  
  
 default Connection getDbConnection() throws SQLException, ClassNotFoundException {  
 String connectionString = "jdbc:mysql://" + *tempConst*.getDbHost() + ":" + *tempConst*.getDbPort() + "/" + *tempConst*.getDbName();  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 Connection dbConnection = DriverManager.*getConnection*(connectionString, *tempConst*.getDbUser(), *tempConst*.getDbPass());  
 return dbConnection;  
 }  
  
 boolean addObj(Object obj);  
  
 ArrayList<Object> getList();  
  
 boolean deleteObj(Object obj);  
  
 boolean editObj(Object obj);  
}

package model.configs.constBD;  
  
public class Const {  
  
 protected final String dbHost = "localhost";  
  
 protected final String dbPort = "3306";  
  
 protected final String dbUser = "root";  
  
 protected final String dbPass = "admin123";  
  
 protected final String dbName = "tour";  
  
 public String getDbHost() {  
 return dbHost;  
 }  
  
 public String getDbPort() {  
 return dbPort;  
 }  
  
 public String getDbUser() {  
 return dbUser;  
 }  
  
 public String getDbPass() {  
 return dbPass;  
 }  
  
 public String getDbName() {  
 return dbName;  
 }  
}

package model.configs.clientBD;  
  
  
import model.configs.constBD.Const;  
  
*/\*\*  
 \* @author Ataev  
 \* class for need connect with database  
 \* \*/*public class ConstClient extends Const {  
  
 public static final String *CLIENT\_TABLE* = "client";  
  
 public static final String *CLIENT\_ID* = "idclient";  
  
 public static final String *CLIENT\_FIO* = "FIO";  
  
 public static final String *CLIENT\_CODE* = "clientCode";  
  
 public static final String *CLIENT\_PASSPORT\_ID* = "passportId";  
  
 public static final String *CLIENT\_MAIL* = "mail";  
  
 public static final String *CLIENT\_MOBILE\_NUMBER* = "mobileNumber";  
  
 public static final String *CLIENT\_LOGIN* = "login";  
  
 public static final String *CLIENT\_PASSWORD* = "password";  
  
 public static final String *CLIENT\_FLAG* = "flag";  
}

package model.configs.orderBD;  
  
import model.configs.constBD.Const;  
  
public class ConstOrder extends Const {  
  
 public static final String *ORDER\_TABLE* = "`order`";  
  
 public static final String *ORDER\_ID* = "idorder";  
  
 public static final String *ORDER\_CLIENT\_CODE* = "clientCode";  
  
 public static final String *ORDER\_TOUR\_CODE* = "tourCode";  
}

package model.configs.ticketBD;  
  
import model.configs.constBD.Const;  
  
public class ConstTicket extends Const {  
  
 public static final String *TICKET\_TABLE* = "ticket";  
  
 public static final String *TICKET\_ID* = "id";  
  
 public static final String *TICKET\_CODE* = "ticketCode";  
  
 public static final String *TICKET\_USER\_CODE* = "userCode";  
  
 public static final String *TICKET\_TRANSPORT\_TYPE* = "transportType";  
  
 public static final String *TICKET\_DEPARTURE\_POINT* = "departurePoint";  
  
 public static final String *TICKET\_ARRIVAL\_POINT* = "arrivalPoint";  
  
 public static final String *TICKET\_DEPARTURE\_DATA* = "departureDate";  
}

package model.configs.tourBD;  
  
import model.configs.constBD.Const;  
  
public class ConstTour extends Const {  
  
 public static final String *TOUR\_TABLE* = "generictour";  
  
 public static final String *TOUR\_ID* = "idgenericTour";  
  
 public static final String *TOUR\_COUNTRY\_NAME* = "countryName";  
  
 public static final String *TOUR\_CITY\_NAME* = "cityName";  
  
 public static final String *TOUR\_PRICE* = "price";  
  
 public static final String *TOUR\_DURATION* = "duration";  
  
 public static final String *TOUR\_CODE* = "tourCode";  
  
 public static final String *TOUR\_DATE* = "tourDate";  
  
 public static final String *TOUR\_NAME* = "tourName";  
  
 public static final String TOUR\_TYPE = "tourType";  
}

package model.delete;  
  
import com.example.model.client.Client;  
import com.example.model.order.Order;  
import com.example.model.ticket.Ticket;  
import com.example.model.tour.Tour;  
import model.bd.dbhclient.DBHClient;  
import model.bd.dbhorder.DBHOrder;  
import model.bd.dbhticket.DBHTicket;  
import model.bd.dbhtour.DBHTour;  
import model.bd.idbhandler.IDBHandler;  
import java.util.ArrayList;  
  
public class Delete {  
  
 private IDBHandler idbHandlerClient = new DBHClient();  
 private IDBHandler idbHandlerTour = new DBHTour();  
 private IDBHandler idbHandlerOrder = new DBHOrder();  
 private IDBHandler idbHandlerTicket = new DBHTicket();  
  
 public boolean deleteOrder(int id,ArrayList<Object> objects) {  
 ArrayList<Order> orders = (ArrayList<Order>) objects.clone();  
 for (Order o : orders) {  
 if (id == o.getId()) {  
 return idbHandlerOrder.deleteObj(o);  
 }  
 }  
 return false;  
 }  
  
 public boolean deleteTour(int id, ArrayList<Object> objects) {  
 ArrayList<Tour> tours = (ArrayList<Tour>) objects.clone();  
 for (Tour t : tours) {  
 if (id == t.getId()) {  
 return idbHandlerTour.deleteObj(t);  
 }  
 }  
 return false;  
 }  
  
 public boolean deleteUser(String login, String pass, String clientCode, ArrayList<Object> objects) {  
 ArrayList<Client> clients = (ArrayList<Client>) objects.clone();  
 for (Client c : clients) {  
 if (login.equals(c.getLogin()) && pass.equals(c.getPassword()) && clientCode.equals(c.getClientCode())) {  
 return idbHandlerClient.deleteObj(c);  
 }  
 }  
 return false;  
 }  
  
 public boolean deleteTicket(int id,ArrayList<Object> objects) {  
 ArrayList<Ticket> tickets = (ArrayList<Ticket>) objects.clone();  
 for (Ticket t : tickets) {  
 if (id == t.getId()) {  
 return idbHandlerTicket.deleteObj(t);  
 }  
 }  
 return false;  
 }  
  
}

package com.example.model.animation;  
import javafx.animation.TranslateTransition;  
import javafx.scene.Node;  
import javafx.util.Duration;  
  
public class Shake {  
  
 private TranslateTransition translateTransition;  
  
 public Shake(Node node) {  
 translateTransition = new TranslateTransition(Duration.*millis*(80), node);  
 translateTransition.setFromX(0f);  
 translateTransition.setByX(10f);  
 translateTransition.setCycleCount(3);  
 translateTransition.setAutoReverse(true);  
 }  
  
 public void playAnim() {  
 translateTransition.playFromStart();  
 }  
}

package com.example.model.check;  
  
  
*/\*\*  
 \* @author Ataev Ismayll  
 \* class Check intended for check String and Integer.  
 \* All methods static.  
 \* I thing, what is use thread then will bad ideas or call methods for static. :-)  
 \* \*/*public class Check {  
  
 */\*\*  
 \* Check value na zero, null and number fort integer.  
 \*/* public static boolean isNumber(String number) {  
 return !number.equals("") && number != null && number.matches("\\d+?");  
 }  
  
 public static boolean isFloat(String fltStr) {  
 try {  
 Float.parseFloat(fltStr);  
 return true;  
 } catch (NumberFormatException ex) {  
 return false;  
 }  
 }  
  
 */\*\*  
 \* Check value zero and null.  
 \* @param str  
 \*/* public static boolean isString(String str) {  
 return !str.equals("") && str != null;  
 }  
}

package com.example.model.client;  
  
  
import java.io.Serializable;  
import java.util.Objects;  
  
public class Client implements Serializable {  
  
 private int id;  
  
 private String FIO;  
  
 private String clientCode;  
  
 private String passportId;  
  
 private String mail;  
  
 private String mobileNumber;  
  
 private String login;  
  
 private String password;  
  
 private int flag;  
  
 public Client() {  
  
 }  
  
 public Client(String FIO, String clientCode, String passportId, String mail,  
 String mobileNumber, String login, String password, int flag) {  
 this.FIO = FIO;  
 this.clientCode = clientCode;  
 this.passportId = passportId;  
 this.mail = mail;  
 this.mobileNumber = mobileNumber;  
 this.login = login;  
 this.password = password;  
 this.flag = flag;  
 }  
  
 public String getFIO() {  
 return FIO;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public void setFIO(String FIO) {  
 this.FIO = FIO;  
 }  
  
 public String getClientCode() {  
 return clientCode;  
 }  
  
 public void setClientCode(String clientCode) {  
 this.clientCode = clientCode;  
 }  
  
 public String getPassportId() {  
 return passportId;  
 }  
  
 public void setPassportId(String passportId) {  
 this.passportId = passportId;  
 }  
  
 public String getMail() {  
 return mail;  
 }  
  
 public void setMail(String mail) {  
 this.mail = mail;  
 }  
  
 public String getMobileNumber() {  
 return mobileNumber;  
 }  
  
 public void setMobileNumber(String mobileNumber) {  
 this.mobileNumber = mobileNumber;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public int getFlag() {  
 return flag;  
 }  
  
 public void setFlag(int flag) {  
 this.flag = flag;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Client client = (Client) o;  
 return flag == client.flag  
 && Objects.*equals*(FIO, client.FIO)  
 && Objects.*equals*(clientCode, client.clientCode)  
 && Objects.*equals*(passportId, client.passportId)  
 && Objects.*equals*(mail, client.mail)  
 && Objects.*equals*(mobileNumber, client.mobileNumber)  
 && Objects.*equals*(login, client.login)  
 && Objects.*equals*(password, client.password);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(FIO, clientCode, passportId, mail, mobileNumber, login, password, flag);  
 }  
  
 @Override  
 public String toString() {  
 return "Client{" +  
 "id=" + id +  
 ", FIO='" + FIO + '\'' +  
 ", clientCode='" + clientCode + '\'' +  
 ", passportId='" + passportId + '\'' +  
 ", mail='" + mail + '\'' +  
 ", mobileNumber='" + mobileNumber + '\'' +  
 ", login='" + login + '\'' +  
 ", password='" + password + '\'' +  
 ", flag=" + flag +  
 '}';  
 }  
}

package com.example.model.connect;  
  
import com.example.model.myexception.MyException;  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
  
*/\*\*  
 \* @author Atayev.I.M  
 \* \*/*public class Connect implements Closeable {  
  
 private Socket socket;  
  
 private BufferedReader bufferedReader;  
  
 private BufferedWriter bufferedWriter;  
  
 private ObjectOutputStream oos;  
  
 private ObjectInputStream ois;  
  
  
 public Connect(String ip, int port) {  
 try {  
 this.socket = new Socket(ip, port);  
 this.bufferedReader = createReader();  
 this.bufferedWriter = createWrite();  
 this.oos = createOOS();  
 this.ois = createOIS();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public Connect(ServerSocket serverSocket) {  
 try {  
 this.socket = serverSocket.accept();  
 this.bufferedReader = createReader();  
 this.bufferedWriter = createWrite();  
 this.oos = createOOS();  
 this.ois = createOIS();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private BufferedReader createReader() throws IOException {  
 return new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 }  
  
 private BufferedWriter createWrite() throws IOException {  
 return new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
 }  
  
 private ObjectOutputStream createOOS() throws IOException {  
 return new ObjectOutputStream(socket.getOutputStream());  
 }  
  
 private ObjectInputStream createOIS() throws IOException {  
 return new ObjectInputStream(socket.getInputStream());  
 }  
  
 public void writeLine(String message) throws IOException {  
 bufferedWriter.write(message);  
 bufferedWriter.newLine();  
 bufferedWriter.flush();  
 }  
  
 public String readLine() throws IOException {  
 return bufferedReader.readLine();  
 }  
  
 public Object readObj() throws IOException, ClassNotFoundException {  
 return ois.readObject();  
 }  
  
 public void writeObj(Object obj) throws IOException {  
 oos.writeObject(obj);  
 }  
  
 public ArrayList<Object> readObjList() throws IOException, ClassNotFoundException {  
 return (ArrayList<Object>) ois.readObject();  
 }  
  
 public void writeObjList(ArrayList<Object> objList) throws IOException {  
 oos.writeObject(objList);  
 oos.flush();  
 }  
  
 public void clearConnect() throws IOException {  
 oos.flush();  
 bufferedWriter.newLine();  
 bufferedWriter.flush();  
 }  
  
 @Override  
 public void close() {  
 try {  
 bufferedReader.close();  
 bufferedWriter.close();  
 oos.close();  
 ois.close();  
 socket.close();  
 } catch (IOException e) {  
 new MyException("Class Connect ",e);  
 }  
 }  
}

package com.example.model.dialog;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import java.io.IOException;  
  
public class InputDialog {  
  
 public InputDialog(ActionEvent actionEvent) {  
  
 Stage stage = new Stage();  
 Parent root = null;  
 try {  
 root = FXMLLoader.*load*(getClass().getResource("sign-up-ui.fxml"));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 stage.setTitle("Новое окно");  
 stage.setScene(new Scene(root, 300, 300));  
 stage.initModality(Modality.*WINDOW\_MODAL*);  
 stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());  
 stage.show();  
 }  
  
 public InputDialog(ActionEvent actionEvent, String path) {  
  
 Stage stage = new Stage();  
 Parent root = null;  
 try {  
 root = FXMLLoader.*load*(getClass().getResource(path));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 stage.setTitle("Новое окно");  
 stage.setScene(new Scene(root, 300, 300));  
 stage.initModality(Modality.*WINDOW\_MODAL*);  
 stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());  
 stage.show();  
 }  
  
 public InputDialog(ActionEvent actionEvent, String path, int h, int w) {  
  
 Stage stage = new Stage();  
 Parent root = null;  
 try {  
 root = FXMLLoader.*load*(getClass().getResource(path));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 stage.setTitle("Новое окно");  
 stage.setScene(new Scene(root, h, w));  
 stage.initModality(Modality.WINDOW\_MODAL);  
 stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());  
 stage.show();  
 }  
}

package com.example.model.myexception;  
  
import org.apache.logging.log4j.Level;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
*/\*\*  
 \* @author Ataev Ismayll  
 \* on class create logger for write error  
 \* логер должен создоватся в проекте 1 раз  
 \*/*public class MyException extends Exception {  
  
 private static final Logger *logger* = LogManager.*getLogger*(MyException.class);  
  
 public MyException() {  
 super();  
 }  
  
 public MyException(String msg) {  
 super();  
 *logger*.log(Level.*ERROR*, msg);  
 }  
  
 public MyException(Exception exception) {  
 super();  
 *logger*.log(Level.*ERROR*, exception.getMessage());  
 }  
  
 public MyException(String msg, Exception exception) {  
 super();  
 *logger*.log(Level.*ERROR*, msg + exception.getMessage());  
 }  
}

package com.example.model.order;  
  
  
import java.io.Serializable;  
import java.util.Objects;  
  
*/\*\*  
 \* @author Atayev I.M.  
 \* \*/*public class Order implements Serializable {  
  
 private int id;  
  
 private String clientCode;  
  
 private String tourCode;  
  
 public Order() {  
 }  
  
 public Order(String clientCode, String tourCode) {  
 this.clientCode = clientCode;  
 this.tourCode = tourCode;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getClientCode() {  
 return clientCode;  
 }  
  
 public void setClientCode(String clientCode) {  
 this.clientCode = clientCode;  
 }  
  
 public String getTourCode() {  
 return tourCode;  
 }  
  
 public void setTourCode(String tourCode) {  
 this.tourCode = tourCode;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Order order = (Order) o;  
 return Objects.*equals*(clientCode, order.clientCode) && Objects.*equals*(tourCode, order.tourCode);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(clientCode, tourCode);  
 }  
  
 @Override  
 public String toString() {  
 return "Order{" +  
 "id=" + id +  
 ", clientCode='" + clientCode + '\'' +  
 ", tourCode='" + tourCode + '\'' +  
 '}';  
 }  
  
 @Override  
 protected Object clone() throws CloneNotSupportedException {  
 return new Order(clientCode, tourCode);  
 }  
}

package com.example.model.rand;  
  
  
*/\*\*  
 \* @author Ataeyv I.M.  
 \* It is class generates random number.  
 \* \*/*public class Rand {  
 public static int random(int min, int max) {  
 return min + (int) (Math.*random*() \* ((max - min) + 1));  
 }  
}

package com.example.model.ticket;  
  
import java.io.Serializable;  
import java.util.Objects;  
  
*/\*\*  
 \* @author Atayev I.M.  
 \*/*public class Ticket implements Serializable {  
  
 private int id;  
  
 private String ticketCode;  
  
 private String userCode;  
  
 private String transportType;  
  
 private String departurePoint;  
  
 private String arrivalPoint;  
  
 private String departureData;  
  
 public Ticket() {  
 }  
  
 public Ticket(String ticketCode, String userCode, String transportType,  
 String departurePoint, String arrivalPoint, String departureData) {  
 this.ticketCode = ticketCode;  
 this.userCode = userCode;  
 this.transportType = transportType;  
 this.departurePoint = departurePoint;  
 this.arrivalPoint = arrivalPoint;  
 this.departureData = departureData;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getTicketCode() {  
 return ticketCode;  
 }  
  
 public void setTicketCode(String ticketCode) {  
 this.ticketCode = ticketCode;  
 }  
  
 public String getUserCode() {  
 return userCode;  
 }  
  
 public void setUserCode(String userCode) {  
 this.userCode = userCode;  
 }  
  
 public String getTransportType() {  
 return transportType;  
 }  
  
 public void setTransportType(String transportType) {  
 this.transportType = transportType;  
 }  
  
 public String getDeparturePoint() {  
 return departurePoint;  
 }  
  
 public void setDeparturePoint(String departurePoint) {  
 this.departurePoint = departurePoint;  
 }  
  
 public String getArrivalPoint() {  
 return arrivalPoint;  
 }  
  
 public void setArrivalPoint(String arrivalPoint) {  
 this.arrivalPoint = arrivalPoint;  
 }  
  
 public String getDepartureData() {  
 return departureData;  
 }  
  
 public void setDepartureData(String departureData) {  
 this.departureData = departureData;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Ticket ticket = (Ticket) o;  
 return Objects.equals(ticketCode, ticket.ticketCode)  
 && Objects.equals(userCode, ticket.userCode)  
 && Objects.equals(transportType, ticket.transportType)  
 && Objects.equals(departurePoint, ticket.departurePoint)  
 && Objects.equals(arrivalPoint, ticket.arrivalPoint)  
 && Objects.equals(departureData, ticket.departureData);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.hash(ticketCode, userCode, transportType, departurePoint, arrivalPoint, departureData);  
 }  
  
 @Override  
 public String toString() {  
 return "Ticket{" +  
 "id=" + id +  
 ", ticketCode='" + ticketCode + '\'' +  
 ", userCode='" + userCode + '\'' +  
 ", transportType='" + transportType + '\'' +  
 ", departurePoint='" + departurePoint + '\'' +  
 ", arrivalPoint='" + arrivalPoint + '\'' +  
 ", departureData='" + departureData + '\'' +  
 '}';  
 }  
}

package com.example.model.tour;  
  
import java.io.Serializable;  
import java.util.Objects;  
  
*/\*\*  
 \* @author Ataeyv I.M.  
 \* @implNote Serializable  
 \* \*/*public class Tour implements Serializable {  
  
 private int id;  
  
 private String countryName;  
  
 private String cityName;  
  
 private float price;  
  
 private String duration;  
  
 private String tourCode;  
  
 private String tourDate;  
  
 private String tourName;  
  
 private String tourType;  
  
 public Tour() {  
 }  
  
 public Tour(String countryName, String cityName, float price, String duration,  
 String tourCode, String tourDate, String tourName, String tourType) {  
 this.countryName = countryName;  
 this.cityName = cityName;  
 this.price = price;  
 this.duration = duration;  
 this.tourCode = tourCode;  
 this.tourDate = tourDate;  
 this.tourName = tourName;  
 this.tourType = tourType;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getCountryName() {  
 return countryName;  
 }  
  
 public void setCountryName(String countryName) {  
 this.countryName = countryName;  
 }  
  
 public String getCityName() {  
 return cityName;  
 }  
  
 public void setCityName(String cityName) {  
 this.cityName = cityName;  
 }  
  
 public float getPrice() {  
 return price;  
 }  
  
 public void setPrice(float price) {  
 this.price = price;  
 }  
  
 public String getDuration() {  
 return duration;  
 }  
  
 public void setDuration(String duration) {  
 this.duration = duration;  
 }  
  
 public String getTourCode() {  
 return tourCode;  
 }  
  
 public void setTourCode(String tourCode) {  
 this.tourCode = tourCode;  
 }  
  
 public String getTourDate() {  
 return tourDate;  
 }  
  
 public void setTourDate(String tourDate) {  
 this.tourDate = tourDate;  
 }  
  
 public String getTourName() {  
 return tourName;  
 }  
  
 public void setTourName(String tourName) {  
 this.tourName = tourName;  
 }  
  
 public String getTourType() {  
 return tourType;  
 }  
  
 public void setTourType(String tourType) {  
 this.tourType = tourType;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Tour tour = (Tour) o;  
 return Float.*compare*(tour.price, price) == 0  
 && Objects.*equals*(countryName, tour.countryName)  
 && Objects.*equals*(cityName, tour.cityName)  
 && Objects.*equals*(duration, tour.duration)  
 && Objects.*equals*(tourCode, tour.tourCode)  
 && Objects.*equals*(tourDate, tour.tourDate)  
 && Objects.*equals*(tourName, tour.tourName)  
 && Objects.*equals*(tourType, tour.tourType);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(countryName, cityName, price, duration, tourCode, tourDate, tourName, tourType);  
 }  
  
 @Override  
 public String toString() {  
 return "Tour{" +  
 "id=" + id +  
 ", countryName='" + countryName + '\'' +  
 ", cityName='" + cityName + '\'' +  
 ", price=" + price +  
 ", duration='" + duration + '\'' +  
 ", tourCode='" + tourCode + '\'' +  
 ", tourDate='" + tourDate + '\'' +  
 ", tourName='" + tourName + '\'' +  
 ", tourType='" + tourType + '\'' +  
 '}';  
 }  
}

package com.example.client.tour;  
  
import com.example.client.MainController;  
import com.example.model.animation.Shake;  
import com.example.model.check.Check;  
import com.example.model.connect.Connect;  
import com.example.model.tour.Tour;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.Node;  
import javafx.scene.control.\*;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
  
  
*/\*\*  
 \*  
 \* \*/*public class AddTourController {  
  
 private Connect connect = MainController.*connect*;  
  
 @FXML  
 private Button addTourPaneBtn;  
  
 @FXML  
 private Label flagTourAddLabel;  
  
 @FXML  
 private Button closeTourPaneBtn;  
  
 @FXML  
 private TextField countryNameField;  
  
 @FXML  
 private TextField cityNameField;  
  
 @FXML  
 private TextField priceTourField;  
  
 @FXML  
 private TextField durationTourField;  
  
 @FXML  
 private TextField tourNameField;  
  
 @FXML  
 private TextField tourTypeField;  
  
 @FXML  
 private DatePicker dateTourField;  
  
 @FXML  
 private ComboBox<String> boxTourType;  
  
 ObservableList<String> boxTourTypeObservableList = FXCollections.observableArrayList("Самолёт", "На море",  
 "Экскурсия", "Горные лыжи", "Экскурсия с отдам на море",  
 "Новый год", "Оздоровление", "Проезд", "Детский отдых");  
  
  
 @FXML  
 void addTourPane(ActionEvent event) {  
 Tour tour = new Tour();  
 String price = priceTourField.getText().trim();  
 String duration = durationTourField.getText().trim();  
 LocalDate dateTour = dateTourField.getValue();  
 if (Check.isFloat(price) && Check.isNumber(duration) ) {  
  
 tour.setPrice(Float.parseFloat(price));  
 tour.setDuration(duration);  
 tour.setTourDate(dateTour.format(DateTimeFormatter.ofPattern("MM-dd-yyyy")));  
 tour.setTourName(tourNameField.getText().trim());  
 tour.setTourType(boxTourType.getValue());  
 tour.setCountryName(countryNameField.getText().trim());  
 tour.setCityName(cityNameField.getText().trim());  
 int a = 10;  
 int b = 10000;  
 int x = a + (int) (Math.random() \* ((b - a) + 1));  
 tour.setTourCode("T" + String.valueOf(x));  
  
 try {  
 connect.writeLine("add");  
 connect.writeLine("addTour");  
 connect.writeObj(tour);  
 String flagAddTout = connect.readLine();  
 if(flagAddTout.equals("true")){  
  
 flagTourAddLabel.setText("Тур успешно добавлен!");  
 }else {  
 flagTourAddLabel.setText("Тур не добавлен!");  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 } else {  
  
 Shake shakePrice = new Shake(priceTourField);  
 Shake shakeDuration = new Shake(durationTourField);  
  
 shakeDuration.playAnim();  
 shakePrice.playAnim();  
 }  
 }  
  
 @FXML  
 void closeTourPane(ActionEvent event) {  
 closeStage(event);  
 }  
  
 private void closeStage(ActionEvent event) {  
 Node source = (Node) event.getSource();  
 Stage stage = (Stage) source.getScene().getWindow();  
 stage.close();  
 }  
  
  
 @FXML  
 void initialize() {  
 boxTourType.setItems(boxTourTypeObservableList);  
 }  
}

package com.example.client;  
  
import java.io.IOException;  
import java.net.URL;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
import java.util.ArrayList;  
import java.util.ResourceBundle;  
import com.example.model.animation.Shake;  
import com.example.model.check.Check;  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.dialog.InputDialog;  
import com.example.model.myexception.MyException;  
import com.example.model.order.Order;  
import com.example.model.ticket.Ticket;  
import com.example.model.tour.Tour;  
import eu.hansolo.tilesfx.chart.SunburstChart;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.Node;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.layout.FlowPane;  
import javafx.stage.Stage;  
  
*/\*\*@author Ataeyv I.M. (ataewisma@gmail.com)\*/*public class AdminController {  
  
 private Client clientSearch;  
 private String flagSearchClient = "false";  
 private Connect connect = MainController.connect;  
  
 @FXML  
 private ResourceBundle resources;  
 @FXML  
 private URL location;  
  
 @FXML  
 private TabPane glavnyPane;  
 @FXML  
 private Label putText4;  
 @FXML  
 private Tab u1;  
 @FXML  
 private Tab u2;  
 @FXML  
 private Tab u3;  
 @FXML  
 private Tab u4;  
 @FXML  
 private Button closeBtn;  
  
 */\*\*  
 \* Main components  
 \*/* @FXML  
 private Button usersBtn;  
 @FXML  
 private Button toursBtn;  
 @FXML  
 private Button ticketsBtn;  
 @FXML  
 private Button viewUsersBtn;  
 @FXML  
 private Button deleteUsersBtn;  
 @FXML  
 private Button addUserBtn;  
  
 */\*\*  
 \* Ticket  
 \*/* @FXML  
 private Label errorTicketDeleteLabel;  
 @FXML  
 private TextField ticketIdDeleteField;  
 @FXML  
 private Button ticketDeleteBtn;  
 @FXML  
 private Button ticketViewBtn;  
 @FXML  
 private TableView<Ticket> ticketTableColumn;  
  
 */\*\*  
 \* Order  
 \*/* @FXML  
 private Label errorOrderPane;  
 @FXML  
 private TextField searchClientCodeField;  
 @FXML  
 private TextField searchTourCodeField;  
 @FXML  
 private Button searchOrderClientCodeBtn;  
 @FXML  
 private Button searchOrderTourCodeBtn;  
 @FXML  
 private TableView<Order> tabViewOrdersSearch;  
 @FXML  
 private TableView<Order> tabViewOrders;  
 private ArrayList<Order> orderArrayList = new ArrayList<>();  
  
  
 */\*\*Tour\*/* @FXML  
 private Button searchTourIdBtn;  
 @FXML  
 private Button editTourBtn;  
 @FXML  
 private Label errorMsgEditTourLabel;  
 @FXML  
 private TextField searchIdTourEditFiled;  
  
 */\*\*Edit\*/* @FXML  
 private TextField idTourEditFiled;  
 @FXML  
 private TextField countryTourEditField;  
 @FXML  
 private TextField cityTourEditField;  
 @FXML  
 private TextField priceTourEditField;  
 @FXML  
 private TextField durationTourEditField;  
 @FXML  
 private TextField codeTourEditField;  
 @FXML  
 private TextField nameTourEditField;  
 @FXML  
 private TextField typeTourEditField;  
 @FXML  
 private DatePicker dateTourEditField;  
  
  
  
 @FXML  
 private Button orderBtn;  
 @FXML  
 private Label errorTourDeleteId;  
 @FXML  
 private TextField deleteIdTourField;  
 @FXML  
 private Button viewOrderBtn;  
 @FXML  
 private Button viewToursBtn;  
 @FXML  
 private Button deleteTourBtn;  
 @FXML  
 private Button addTourBtn;  
 @FXML  
 private TableView<Tour> tabViewTours;  
  
 */\*\*Client and User\*/* @FXML  
 private TableView<Client> usersTableView;  
 @FXML  
 private TextField signUpFIOField;  
 @FXML  
 private TextField signUpClientCodeField;  
 @FXML  
 private TextField signUpPassportIdField;  
 @FXML  
 private TextField signUpMailField;  
 @FXML  
 private TextField signUpMobileNumberField;  
 @FXML  
 private TextField signUpLoginField;  
 @FXML  
 private TextField signUpPasswordField;  
 @FXML  
 private TextField signUpFlagField;  
 @FXML  
 private TextField loginDeleteField;  
 @FXML  
 private TextField passwordDeleteField;  
 @FXML  
 private TextField clientCodeDeleteField;  
  
 @FXML  
 private Label outPutErrorAddUserLabel;  
 @FXML  
 private Label errorDeleteUserLabel;  
 @FXML  
 private Label errorSearchLabel;  
 @FXML  
 private Label errorEditLabel;  
  
 */\*\*Edit table\*/* @FXML  
 private TextField signUpSearchFIOField;  
 @FXML  
 private TextField signUpSearchLoginField;  
 @FXML  
 private Button searchBtn;  
 @FXML  
 private TextField signUpSearchPasswordField;  
 @FXML  
 private TextField signUpEditFlagField;  
 @FXML  
 private TextField signUpEditPasswordField;  
 @FXML  
 private TextField signUpEditMailField;  
 @FXML  
 private TextField signUpEditFIOField;  
 @FXML  
 private TextField signUpEditClientCodeField;  
 @FXML  
 private TextField signUpEditPassportIdField;  
 @FXML  
 private TextField signUpEditMobileNumberField;  
 @FXML  
 private TextField signUpEditLoginField;  
 @FXML  
 private Button editBtn;  
  
 private ArrayList<Tour> tourArrayList = new ArrayList<>();  
  
  
  
  
 @FXML  
 void initialize() {  
  
 closeBtn.setOnAction(ActionEvent -> {  
 try {  
 connect.writeLine("close");  
 connect.close();  
 System.exit(1);  
 } catch (IOException e) {  
 new MyException(e);  
 }  
 });  
  
 usersBtn.setOnAction(actionEvent -> {  
 putText4.setText("Пользователи");  
 glavnyPane.getSelectionModel().select(u1);  
 });  
  
 toursBtn.setOnAction(actionEvent -> {  
 putText4.setText("Туры");  
 *//errorTourDeleteId.setText("");* glavnyPane.getSelectionModel().select(u2);  
 });  
  
 ticketsBtn.setOnAction(actionEvent -> {  
 putText4.setText("Билеты");  
 glavnyPane.getSelectionModel().select(u3);  
 });  
  
 orderBtn.setOnAction(ActionEvent -> {  
 errorOrderPane.setText("");  
 searchTourCodeField.setText("");  
 searchClientCodeField.setText("");  
 putText4.setText("Заказы");  
 glavnyPane.getSelectionModel().select(u4);  
 });  
  
  
 */\*\*Просмотр пользователя\*/* viewUsersBtn.setOnAction(actionEvent -> {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewUser");  
 ArrayList<Client> clientArrayList = (ArrayList<Client>) connect.readObjList().clone();  
 ObservableList<Client> observableList = FXCollections.observableArrayList(clientArrayList);  
 usersTableView.setItems(observableList);  
 usersTableView.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("FIO"));  
 usersTableView.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("clientCode"));  
 usersTableView.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("passportId"));  
 usersTableView.getColumns().get(3).setCellValueFactory(new PropertyValueFactory("mail"));  
 usersTableView.getColumns().get(4).setCellValueFactory(new PropertyValueFactory("mobileNumber"));  
 usersTableView.getColumns().get(5).setCellValueFactory(new PropertyValueFactory("login"));  
 usersTableView.getColumns().get(6).setCellValueFactory(new PropertyValueFactory("password"));  
 usersTableView.getColumns().get(7).setCellValueFactory(new PropertyValueFactory("flag"));  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 });  
  
 */\*\*Добавить пользователя\*/* addUserBtn.setOnAction(actionEvent -> {  
  
 String fio = signUpFIOField.getText().trim();  
 String clientCode = signUpClientCodeField.getText().trim();  
 String passportId = signUpPassportIdField.getText().trim();  
 String mail = signUpMailField.getText().trim();  
 String mobileNumber = signUpMobileNumberField.getText().trim();  
 String login = signUpLoginField.getText().trim();  
 String password = signUpPasswordField.getText().trim();  
 String flag = signUpFlagField.getText().trim();  
  
 if (flag.equals("1") || flag.equals("2") && Check.isNumber(mobileNumber)  
 && Check.isString(fio) && Check.isString(clientCode) && Check.isString(passportId)  
 && Check.isString(mail) && Check.isString(login) && Check.isString(password)) {  
  
 signUpFlagField.setText("");  
 signUpFIOField.setText("");  
 signUpMobileNumberField.setText("");  
 signUpClientCodeField.setText("");  
 signUpPassportIdField.setText("");  
 signUpMailField.setText("");  
 signUpLoginField.setText("");  
 signUpPasswordField.setText("");  
  
 Client client = new Client();  
 client.setFIO(fio);  
 client.setClientCode(clientCode);  
 client.setPassportId(passportId);  
 client.setMail(mail);  
 client.setMobileNumber(mobileNumber);  
 client.setLogin(login);  
 client.setPassword(password);  
 client.setFlag(Integer.parseInt(flag));  
  
 try {  
 connect.writeLine("add");  
 connect.writeLine("addUser");  
 connect.writeObj(client);  
 String flagAddClient = connect.readLine();  
  
 if (flagAddClient.equals("true")) {  
 outPutErrorAddUserLabel.setText("Ползователь добавлен в базу");  
 } else if (flagAddClient.equals("false")) {  
 outPutErrorAddUserLabel.setText("Ползователь не добавлен");  
 } else {  
 outPutErrorAddUserLabel.setText(" ошибка дабавления обратитесь к администрации ");  
 }  
 outPutErrorAddUserLabel.setText("");  
  
 } catch (IOException e) {  
 new MyException(e);  
 }  
  
 } else {  
 Shake shakeFIO = new Shake(signUpFIOField);  
 Shake shakeClientCode = new Shake(signUpClientCodeField);  
 Shake shakeMobileNumber = new Shake(signUpMobileNumberField);  
 Shake shakePassword = new Shake(signUpPasswordField);  
 Shake shakeMail = new Shake(signUpMailField);  
 Shake shakeLogin = new Shake(signUpLoginField);  
 Shake shakePassportId = new Shake(signUpPassportIdField);  
 Shake shakeFlag = new Shake(signUpFlagField);  
  
 shakeFlag.playAnim();  
 shakeClientCode.playAnim();  
 shakeFIO.playAnim();  
 shakeLogin.playAnim();  
 shakeMail.playAnim();  
 shakeMobileNumber.playAnim();  
 shakePassportId.playAnim();  
 shakePassword.playAnim();  
 outPutErrorAddUserLabel.setText("неправелный ввод данныз проверти поля");  
 }  
 });  
  
 */\*\*Удалить пользователя\*/* deleteUsersBtn.setOnAction(actionEvent -> {  
 try {  
 String loginDelete = loginDeleteField.getText().trim();  
 String passwordDelete = passwordDeleteField.getText().trim();  
 String clientCodeDelete = clientCodeDeleteField.getText().trim();  
  
 Shake shakeLogin = new Shake(loginDeleteField);  
 Shake shakePassword = new Shake(passwordDeleteField);  
 Shake shakeClientCode = new Shake(clientCodeDeleteField);  
  
 if (Check.isString(loginDelete) && Check.isString(passwordDelete) && Check.isString(clientCodeDelete)) {  
  
 connect.writeLine("delete");  
 connect.writeLine("deleteUser");  
 connect.writeLine(loginDelete);  
 connect.writeLine(passwordDelete);  
 connect.writeLine(clientCodeDelete);  
 String flagDelete = connect.readLine();  
  
 if (flagDelete.equals("true")) {  
 loginDeleteField.setText("");  
 passwordDeleteField.setText("");  
 clientCodeDeleteField.setText("");  
 errorDeleteUserLabel.setText("пользователя успешно удален !!!");  
 } else {  
 shakeClientCode.playAnim();  
 shakeLogin.playAnim();  
 shakePassword.playAnim();  
 errorDeleteUserLabel.setText(" нету такого пользователя!!!");  
 }  
  
 } else {  
 shakeClientCode.playAnim();  
 shakeLogin.playAnim();  
 shakePassword.playAnim();  
 errorDeleteUserLabel.setText(" введите соответствуюшие данные !!!");  
 }  
 errorDeleteUserLabel.setText("");  
 } catch (Exception e) {  
 new MyException(e);  
 }  
 });  
  
 */\*\*Поиск пользователя\*/* searchBtn.setOnAction(actionEvent -> {  
 try {  
 errorSearchLabel.setText("");  
  
 String searchFio = signUpSearchFIOField.getText().trim();  
 String searchLogin = signUpSearchLoginField.getText().trim();  
 String searchPassword = signUpSearchPasswordField.getText().trim();  
  
 Shake shakeSearchFio = new Shake(signUpSearchFIOField);  
 Shake shakeSearchLogin = new Shake(signUpSearchLoginField);  
 Shake shakeSearchPassword = new Shake(signUpSearchPasswordField);  
  
 if (Check.isString(searchFio) && Check.isString(searchLogin) && Check.isString(searchPassword)) {  
 connect.writeLine("search");  
 connect.writeLine("searchUser");  
 connect.writeLine(searchFio);  
 connect.writeLine(searchLogin);  
 connect.writeLine(searchPassword);  
 String flagSearchUser = connect.readLine();  
 this.flagSearchClient = flagSearchUser;  
 Client client;  
  
 if (flagSearchUser.equals("true")) {  
 this.clientSearch = null;  
 client = (Client) connect.readObj();  
 this.clientSearch = client;  
 if (client != null) {  
 signUpEditFIOField.setText(client.getFIO());  
 signUpEditClientCodeField.setText(client.getClientCode());  
 signUpEditPassportIdField.setText(client.getPassportId());  
 signUpEditMailField.setText(client.getMail());  
 signUpEditPasswordField.setText(client.getPassword());  
 signUpEditLoginField.setText(client.getLogin());  
 signUpEditMobileNumberField.setText(client.getMobileNumber());  
 signUpEditFlagField.setText(String.valueOf(client.getFlag()));  
 }  
 } else if (flagSearchUser.equals("false")) {  
 signUpEditFIOField.setText("");  
 signUpEditClientCodeField.setText("");  
 signUpEditPassportIdField.setText("");  
 signUpEditMailField.setText("");  
 signUpEditPasswordField.setText("");  
 signUpEditLoginField.setText("");  
 signUpEditMobileNumberField.setText("");  
 signUpEditFlagField.setText("");  
 errorSearchLabel.setText("нету такого пользователя");  
 } else {  
 errorSearchLabel.setText("проблема с сервером обратитесь к администрации");  
 }  
 } else {  
 signUpEditFIOField.setText("");  
 signUpEditClientCodeField.setText("");  
 signUpEditPassportIdField.setText("");  
 signUpEditMailField.setText("");  
 signUpEditPasswordField.setText("");  
 signUpEditLoginField.setText("");  
 signUpEditMobileNumberField.setText("");  
 signUpEditFlagField.setText("");  
 shakeSearchFio.playAnim();  
 shakeSearchLogin.playAnim();  
 shakeSearchPassword.playAnim();  
 errorSearchLabel.setText("заполните пустые поля для поиска!!!");  
 }  
 } catch (Exception e) {  
 new MyException(e);  
 }  
 signUpSearchFIOField.setText("");  
 signUpSearchLoginField.setText("");  
 signUpSearchPasswordField.setText("");  
 });  
  
 */\*\*Изменение пользователя\*/* editBtn.setOnAction(actionEvent -> {  
 Shake shakeFIO = new Shake(signUpEditFIOField);  
 Shake shakeClientCode = new Shake(signUpEditClientCodeField);  
 Shake shakeMobileNumber = new Shake(signUpEditMobileNumberField);  
 Shake shakePassword = new Shake(signUpEditPasswordField);  
 Shake shakeMail = new Shake(signUpEditMailField);  
 Shake shakeLogin = new Shake(signUpEditLoginField);  
 Shake shakePassportId = new Shake(signUpEditPassportIdField);  
 Shake shakeFlag = new Shake(signUpEditFlagField);  
  
 Client client = new Client();  
 String fio = signUpEditFIOField.getText().trim();  
 String clientCode = signUpEditClientCodeField.getText().trim();  
 String passportId = signUpEditPassportIdField.getText().trim();  
 String mail = signUpEditMailField.getText().trim();  
 String pass = signUpEditPasswordField.getText().trim();  
 String login = signUpEditLoginField.getText().trim();  
 String mobileNumber = signUpEditMobileNumberField.getText().trim();  
 String flag = signUpEditFlagField.getText().trim();  
  
 if (this.flagSearchClient.equals("true") && clientSearch != null  
 && Check.*isString*(fio) && Check.*isString*(clientCode)  
 && Check.*isString*(passportId) && Check.*isString*(mail)  
 && Check.*isString*(pass) && Check.*isString*(login)  
 && Check.*isString*(mobileNumber) && Check.*isNumber*(flag)) {  
 client.setId(this.clientSearch.getId());  
 client.setFIO(fio);  
 client.setClientCode(clientCode);  
 client.setPassportId(passportId);  
 client.setMail(mail);  
 client.setPassword(pass);  
 client.setLogin(login);  
 client.setMobileNumber(mobileNumber);  
 client.setFlag(Integer.parseInt(flag));  
 try {  
 connect.writeLine("edit");  
 connect.writeLine("editUser");  
 connect.writeObj(client);  
 String flagEdit = connect.readLine();  
 System.out.println(flagEdit + " i am flagEdit ");  
 if (flagEdit.equals("true")) {  
 signUpEditFIOField.setText("");  
 signUpEditClientCodeField.setText("");  
 signUpEditPassportIdField.setText("");  
 signUpEditMailField.setText("");  
 signUpEditPasswordField.setText("");  
 signUpEditLoginField.setText("");  
 signUpEditMobileNumberField.setText("");  
 signUpEditFlagField.setText("");  
 this.errorEditLabel.setText("пользователь успешно обнавлён.");  
 } else {  
 shakeFIO.playAnim();  
 shakeClientCode.playAnim();  
 shakeMobileNumber.playAnim();  
 shakePassword.playAnim();  
 shakeMail.playAnim();  
 shakeLogin.playAnim();  
 shakePassportId.playAnim();  
 shakeFlag.playAnim();  
 this.errorEditLabel.setText("пользователь не изменён ошибка на сервере ");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 }  
 } else {  
 shakeFIO.playAnim();  
 shakeClientCode.playAnim();  
 shakeMobileNumber.playAnim();  
 shakePassword.playAnim();  
 shakeMail.playAnim();  
 shakeLogin.playAnim();  
 shakePassportId.playAnim();  
 shakeFlag.playAnim();  
 }  
 this.errorEditLabel.setText("");  
 });  
  
 */\*\*Просмотр билетов\*/* ticketViewBtn.setOnAction(actionEvent -> {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewTicket");  
 ArrayList<Ticket> ticketAdminArrayList = (ArrayList<Ticket>) connect.readObjList().clone();  
 ObservableList<Ticket> observableList = FXCollections.observableArrayList(ticketAdminArrayList);  
 ticketTableColumn.setItems(observableList);  
 ticketTableColumn.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 ticketTableColumn.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("ticketCode"));  
 ticketTableColumn.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("userCode"));  
 ticketTableColumn.getColumns().get(3).setCellValueFactory(new PropertyValueFactory("transportType"));  
 ticketTableColumn.getColumns().get(4).setCellValueFactory(new PropertyValueFactory("departurePoint"));  
 ticketTableColumn.getColumns().get(5).setCellValueFactory(new PropertyValueFactory("arrivalPoint"));  
 ticketTableColumn.getColumns().get(6).setCellValueFactory(new PropertyValueFactory("departureData"));  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 });  
  
 */\*\*Удалить билет\*/* ticketDeleteBtn.setOnAction(ActionEvent -> {  
 try {  
 String idTicket = ticketIdDeleteField.getText().trim();  
 if (Check.isNumber(idTicket)) {  
 connect.writeLine("delete");  
 connect.writeLine("ticketDelete");  
 connect.writeLine(idTicket);  
 final String flag = connect.readLine();  
 if (flag.equals("true")) {  
 errorTicketDeleteLabel.setText("БИЛЕТ УСПЕШНО УДАЛЁН");  
 } else if (flag.equals("false")) {  
 errorTicketDeleteLabel.setText("НЕТУ ТАКОГО БИЛЕТ С ТАКИМ ID");  
 }  
 } else {  
 Shake shake = new Shake(ticketIdDeleteField);  
 shake.playAnim();  
 errorTicketDeleteLabel.setText("ВВЕДИТЕ ЧИСЛО");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 } finally {  
 ticketIdDeleteField.setText("");  
 }  
 });  
  
 */\*\*Поиск заказов по клиенту\*/* searchOrderClientCodeBtn.setOnAction(ActionEvent -> {  
 try {  
 String clientCode = searchClientCodeField.getText().trim();  
 if (Check.isString(clientCode)) {  
  
 connect.writeLine("view");  
 connect.writeLine("viewOrder");  
 ArrayList<Order> orderArrayList = (ArrayList<Order>) connect.readObjList().clone();  
 ArrayList<Order> orders = new ArrayList<>();  
  
 for (Order o : orderArrayList) {  
 if (clientCode.equals(o.getClientCode())) {  
 orders.add(o);  
 }  
 }  
  
 if (orders.isEmpty()) {  
 errorOrderPane.setText("Нету стаким заказом клиент");  
 tabViewOrdersSearch.getItems().clear();  
 } else {  
 ObservableList<Order> observableList = FXCollections.observableArrayList(orders);  
 tabViewOrdersSearch.setItems(observableList);  
 tabViewOrdersSearch.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 tabViewOrdersSearch.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("clientCode"));  
 tabViewOrdersSearch.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("tourCode"));  
 errorOrderPane.setText("");  
 }  
 } else {  
 Shake shakeSearch = new Shake(searchClientCodeField);  
 shakeSearch.playAnim();  
 errorOrderPane.setText("Введите код клиента");  
 tabViewOrdersSearch.getItems().clear();  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 } finally {  
 searchTourCodeField.setText("");  
 searchClientCodeField.setText("");  
 }  
 });  
  
 */\*\*Поиск заказов по туру\*/* searchOrderTourCodeBtn.setOnAction(ActionEvent -> {  
 try {  
 String tourCode = searchTourCodeField.getText().trim();  
 if (Check.isString(tourCode)) {  
  
 connect.writeLine("view");  
 connect.writeLine("viewOrder");  
 ArrayList<Order> orderArrayList = (ArrayList<Order>) connect.readObjList().clone();  
 ArrayList<Order> orders = new ArrayList<>();  
  
 for (Order o : orderArrayList) {  
 if (tourCode.equals(o.getTourCode())) {  
 orders.add(o);  
 }  
 }  
  
 if (orders.isEmpty()) {  
 errorOrderPane.setText("Нету стаким заказом тура");  
 tabViewOrdersSearch.getItems().clear();  
 } else {  
 ObservableList<Order> observableList = FXCollections.observableArrayList(orders);  
 tabViewOrdersSearch.setItems(observableList);  
 tabViewOrdersSearch.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 tabViewOrdersSearch.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("clientCode"));  
 tabViewOrdersSearch.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("tourCode"));  
 searchTourCodeField.setText("");  
 }  
 } else {  
 Shake shakeSearch = new Shake(searchTourCodeField);  
 shakeSearch.playAnim();  
 errorOrderPane.setText("Введите код тура");  
 tabViewOrdersSearch.getItems().clear();  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 } finally {  
 searchTourCodeField.setText("");  
 searchClientCodeField.setText("");  
 }  
 });  
  
 */\*\*Поиск тура\*/* searchTourIdBtn.setOnAction(ActionEvent -> {  
 try {  
 String id = searchIdTourEditFiled.getText().trim();  
 if (Check.isNumber(id)) {  
 int idTour = Integer.parseInt(id);  
 connect.writeLine("view");  
 connect.writeLine("viewTour");  
 ArrayList<Tour> tours = (ArrayList<Tour>) connect.readObjList().clone();  
 int i = 0;  
  
 for (Tour t : tours) {  
 if (idTour == t.getId()) {  
 ++i;  
 idTourEditFiled.setText(String.valueOf(t.getId()));  
 countryTourEditField.setText(t.getCountryName());  
 cityTourEditField.setText(t.getCityName());  
 priceTourEditField.setText(String.valueOf(t.getPrice()));  
 durationTourEditField.setText(t.getDuration());  
 codeTourEditField.setText(t.getTourCode());  
 nameTourEditField.setText(t.getTourName());  
 typeTourEditField.setText(t.getTourType());  
 }  
 }  
 if (i == 0) {  
 errorMsgEditTourLabel.setText("Нету тура с таким Id");  
 }  
 } else {  
 Shake shake = new Shake(searchIdTourEditFiled);  
 shake.playAnim();  
 errorMsgEditTourLabel.setText("Введите Id тура");  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 } finally {  
 searchIdTourEditFiled.setText("");  
 }  
 });  
  
 */\*\*Изменение тура\*/* editTourBtn.setOnAction(ActionEvent -> {  
 try {  
 String idTour = idTourEditFiled.getText().trim();  
 String countryTour = countryTourEditField.getText().trim();  
 String cityTour = cityTourEditField.getText().trim();  
 String priceTour = priceTourEditField.getText().trim();  
 String durationTour = durationTourEditField.getText().trim();  
 String codeTour = codeTourEditField.getText().trim();  
 String nameTour = nameTourEditField.getText().trim();  
 String typeTour = typeTourEditField.getText().trim();  
 LocalDate localDate = dateTourEditField.getValue();  
 dateTourEditField.setAccessibleText("hi ella");  
  
 if (Check.isNumber(idTour) && Check.isNumber(durationTour) && Check.isFloat(priceTour) && Check.isString(countryTour)  
 && Check.isString(cityTour) && Check.isString(codeTour) && Check.isString(nameTour) && Check.isString(typeTour) && Check.isString(localDate.toString())) {  
  
 Tour t = new Tour();  
 t.setId(Integer.parseInt(idTour));  
 t.setCountryName(countryTour);  
 t.setCityName(cityTour);  
 t.setPrice(Float.parseFloat(priceTour));  
 t.setDuration(durationTour);  
 t.setTourCode(codeTour);  
 t.setTourName(nameTour);  
 t.setTourType(typeTour);  
 t.setTourDate(localDate.format(DateTimeFormatter.ofPattern("MM-dd-yyyy")));  
  
 connect.writeLine("edit");  
 connect.writeLine("editTour");  
 connect.writeObj(t);  
 String flag = connect.readLine();  
  
 if (flag.equals("true")) {  
 errorMsgEditTourLabel.setText("Тур успешно редактировано");  
 } else {  
 errorMsgEditTourLabel.setText("Тур не редактировано");  
 }  
  
 } else {  
 errorMsgEditTourLabel.setText("Нажмите кнопку поиск");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 } finally {  
 idTourEditFiled.setText("");  
 countryTourEditField.setText("");  
 cityTourEditField.setText("");  
 priceTourEditField.setText("");  
 durationTourEditField.setText("");  
 codeTourEditField.setText("");  
 nameTourEditField.setText("");  
 typeTourEditField.setText("s");  
 }  
 });  
  
  
 }  
  
 */\*\*Удаление тура\*/* @FXML  
 void deleteTour(ActionEvent event) {  
 try {  
 String flagDeleteTourServer = null;  
 String id = deleteIdTourField.getText().trim();  
 if (Check.isNumber(id)) {  
 connect.writeLine("delete");  
 connect.writeLine("deleteTour");  
 connect.writeLine(id);  
 flagDeleteTourServer = connect.readLine();  
  
 if (flagDeleteTourServer.equals("true")) {  
 errorTourDeleteId.setText("Тур успешно удален");  
 } else {  
 errorTourDeleteId.setText("Тур не удалён");  
 System.out.println("Тур не удалён");  
 }  
 } else {  
 Shake shakeTourDelete = new Shake(deleteIdTourField);  
 shakeTourDelete.playAnim();  
 *// errorTourDeleteId.setText("");* System.out.println("Введите число");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 } finally {  
 deleteIdTourField.setText("");  
 }  
 }  
  
 */\*\*Просмотр Туров\*/* @FXML  
 void getToursView(ActionEvent event) {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewTour");  
 ArrayList<Tour> tourArrayList = (ArrayList<Tour>) connect.readObjList().clone();  
 this.tourArrayList = tourArrayList;  
 ObservableList<Tour> observableList = FXCollections.observableArrayList(tourArrayList);  
 tabViewTours.setItems(observableList);  
 tabViewTours.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 tabViewTours.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("countryName"));  
 tabViewTours.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("cityName"));  
 tabViewTours.getColumns().get(3).setCellValueFactory(new PropertyValueFactory("price"));  
 tabViewTours.getColumns().get(4).setCellValueFactory(new PropertyValueFactory("duration"));  
 tabViewTours.getColumns().get(5).setCellValueFactory(new PropertyValueFactory("tourCode"));  
 tabViewTours.getColumns().get(6).setCellValueFactory(new PropertyValueFactory("tourDate"));  
 tabViewTours.getColumns().get(7).setCellValueFactory(new PropertyValueFactory("tourName"));  
 tabViewTours.getColumns().get(8).setCellValueFactory(new PropertyValueFactory("tourType"));  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 }  
  
 */\*\*Просмотр Заказов\*/* @FXML  
 void getOrderView(ActionEvent event) {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewOrder");  
 ArrayList<Order> orderArrayList = (ArrayList<Order>) connect.readObjList().clone();  
 this.orderArrayList = orderArrayList;  
 ObservableList<Order> observableList = FXCollections.observableArrayList(orderArrayList);  
 tabViewOrders.setItems(observableList);  
 tabViewOrders.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 tabViewOrders.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("clientCode"));  
 tabViewOrders.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("tourCode"));  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 }  
  
 */\*\*Добавить тур\*/* @FXML  
 void addTour(ActionEvent event) {  
 new InputDialog(event, "add-tour.fxml", 530, 475);  
 }  
  
 */\*\*Создать билет\*/* @FXML  
 void checkAndCreateTicket(ActionEvent event) {  
 new InputDialog(event, "check-create-ticket.fxml", 400, 400);  
 }  
  
}

package com.example.client;  
  
import java.io.IOException;  
import java.net.URL;  
import java.util.ResourceBundle;  
import com.example.model.check.Check;  
import com.example.model.connect.Connect;  
import com.example.model.myexception.MyException;  
import com.example.model.rand.Rand;  
import com.example.model.ticket.Ticket;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.Node;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
*/\*\*  
 \* @author Ataeyv I.M.  
 \* it is class create Ticket get user, tour code and create ticket.  
 \* \*/*public class CheckAndCreateTicketController {  
  
 private Connect connect = MainController.connect;  
 @FXML  
 private ResourceBundle resources;  
 @FXML  
 private URL location;  
 @FXML  
 private Button createTicketBtn;  
 @FXML  
 private TextField idOrderField;  
 @FXML  
 private TextField departurePointField;  
 @FXML  
 private Label errorIdOrderLabel;  
 @FXML  
 private ComboBox<String> boxTransportType;  
 ObservableList<String> list = FXCollections.observableArrayList("Самолёт", "Автобус", "Поезд", "Корабль");  
  
  
 @FXML  
 void closeCreateTicketPane(ActionEvent event) {  
 Node source = (Node) event.getSource();  
 Stage stage = (Stage) source.getScene().getWindow();  
 stage.close();  
 }  
  
 @FXML  
 void initialize() {  
 boxTransportType.setItems(list);  
 createTicketBtn.setOnAction(ActionEvent -> {  
 try {  
 String idOrder = idOrderField.getText().trim();  
 String transportType = boxTransportType.getValue();  
 String departurePoint = departurePointField.getText().trim();  
  
 if (Check.*isString*(transportType) && Check.*isString*(departurePoint)) {  
 if (Check.*isNumber*(idOrder)) {  
  
 Ticket ticket = new Ticket();  
 ticket.setTransportType(transportType);  
 ticket.setDeparturePoint(departurePoint);  
 ticket.setTicketCode("B" + String.*valueOf*(Rand.*random*(10, 10000)));  
  
 connect.writeLine("add");  
 connect.writeLine("addTicket");  
 connect.writeLine(idOrder);  
 connect.writeObj(ticket);  
 String flagAddTicketAndCheckOrder = connect.readLine();  
  
 System.*out*.println(flagAddTicketAndCheckOrder);  
 if (flagAddTicketAndCheckOrder.equals("true")) {  
 errorIdOrderLabel.setText("БИЛЕТ СОЗДАН");  
 } else {  
 errorIdOrderLabel.setText("БИЛЕТ НЕ СОЗДАН");  
 }  
  
 } else {  
 errorIdOrderLabel.setText("Введите число в поле id заказов!!!");  
 }  
 } else {  
 errorIdOrderLabel.setText("Запольните поля!!!");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 }  
 });  
 }  
}

package com.example.client;  
  
import java.io.IOException;  
import java.net.URL;  
import java.util.ArrayList;  
import java.util.ResourceBundle;  
import com.example.model.animation.Shake;  
import com.example.model.check.Check;  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.myexception.MyException;  
import com.example.model.order.Order;  
import com.example.model.ticket.Ticket;  
import com.example.model.tour.Tour;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
  
public class ClientController {  
  
 private Connect connect = MainController.*connect*;  
 private Client profile = new Client();  
  
 @FXML  
 private ResourceBundle resources;  
 @FXML  
 private URL location;  
  
 */\*\*Main components\*/* @FXML  
 private Label putTextULabel;  
 @FXML  
 private Button closeBtn;  
 @FXML  
 private TabPane glvTabPane;  
 @FXML  
 private Tab u1;  
 @FXML  
 private Tab u2;  
 @FXML  
 private Tab u3;  
 @FXML  
 private Tab u4;  
 @FXML  
 private Button toursBtn;  
 @FXML  
 private Button myOrderBtn;  
 @FXML  
 private Button profileBtn;  
 @FXML  
 private Button myTicketBtn;  
  
  
 */\*\*Order\*/* @FXML  
 private Label errorOrderClientLabel;  
 @FXML  
 private TextField cancelOrderClientIdField;  
 @FXML  
 private Button cancelOrderClientBtn;  
 @FXML  
 private Button viewOrderClientBtn;  
 @FXML  
 private TableView<Order> tabViewOrdersClient;  
  
  
 */\*\*Ticket\*/* @FXML  
 private Button ticketClientViewBtn;  
 @FXML  
 private Button ticketViewBtn;  
 @FXML  
 private TableView<Ticket> ticketTableColumn;  
  
  
 */\*\*User\*/* @FXML  
 private TextField signUpFIOField;  
 @FXML  
 private TextField signUpClientCodeField;  
 @FXML  
 private TextField signUpPassportIdField;  
 @FXML  
 private TextField signUpMailField;  
 @FXML  
 private TextField signUpMobileNumberField;  
 @FXML  
 private TextField signUpLoginField;  
 @FXML  
 private TextField signUpPasswordField;  
  
 */\*\*Tour\*/* @FXML  
 private Label errorOrderTour;  
 @FXML  
 private TextField inputTourCodeMakeOrderField;  
 @FXML  
 private Button makeOrderTourBtn;  
 @FXML  
 private Button tourViewBtn;  
 @FXML  
 private TableView<Tour> tabViewTours;  
  
  
 @FXML  
 void initialize() {  
 this.profile = MainController.client;  
  
 closeBtn.setOnAction(ActionEvent -> {  
 try {  
 connect.writeLine("close");  
 tabViewTours = null;  
 ticketTableColumn = null;  
 connect.close();  
 connect = null;  
 profile = null;  
 System.exit(2);  
 } catch (IOException e) {  
 new MyException(e);  
 }  
 });  
  
 toursBtn.setOnAction(ActionEvent -> {  
 errorOrderTour.setText("");  
 putTextULabel.setText("Туры");  
 glvTabPane.getSelectionModel().select(u1);  
 });  
  
 myTicketBtn.setOnAction(ActionEvent -> {  
 putTextULabel.setText("Мои Билеты");  
 glvTabPane.getSelectionModel().select(u3);  
 });  
  
 myOrderBtn.setOnAction(ActionEvent -> {  
 errorOrderClientLabel.setText("");  
 cancelOrderClientIdField.setText("");  
 putTextULabel.setText("Мои Заказы");  
 glvTabPane.getSelectionModel().select(u4);  
 });  
  
  
 */\*\*Профил пользователя\*/* profileBtn.setOnAction(ActionEvent -> {  
 errorOrderTour.setText("");  
 putTextULabel.setText("Профил");  
 glvTabPane.getSelectionModel().select(u2);  
 signUpFIOField.setText(profile.getFIO());  
 signUpClientCodeField.setText(profile.getClientCode());  
 signUpPassportIdField.setText(profile.getPassportId());  
 signUpMailField.setText(profile.getMail());  
 signUpMobileNumberField.setText(profile.getMobileNumber());  
 signUpLoginField.setText(profile.getLogin());  
 signUpPasswordField.setText(profile.getPassword());  
 });  
  
 */\*\*Броноруем тур\*/* makeOrderTourBtn.setOnAction(ActionEvent -> {  
 try {  
 String inputTourCode = inputTourCodeMakeOrderField.getText().trim();  
 if (Check.isString(inputTourCode)) {  
  
 Client client = new Client();  
 client.setClientCode(profile.getClientCode());  
 client.setId(profile.getId());  
 client.setFlag(profile.getFlag());  
 client.setPassword(profile.getPassword());  
 client.setLogin(profile.getLogin());  
 client.setMail(profile.getMail());  
 client.setFIO(profile.getFIO());  
 client.setPassportId(profile.getPassportId());  
  
 connect.writeLine("add");  
 connect.writeLine("orderTour");  
 connect.writeLine(inputTourCode);  
 connect.writeObj(client);  
 String flagOrderAddOrNot = connect.readLine();  
  
 if (flagOrderAddOrNot.equals("true")) {  
 errorOrderTour.setText("Тур забронирован");  
 } else if (flagOrderAddOrNot.equals("false")) {  
 errorOrderTour.setText("Тур не забронирован");  
 } else {  
 errorOrderTour.setText("ошибка в работе");  
 }  
 flagOrderAddOrNot = "";  
 } else {  
 Shake shakeOrderTour = new Shake(inputTourCodeMakeOrderField);  
 shakeOrderTour.playAnim();  
 errorOrderTour.setText("Ведите код тура");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 } finally {  
 inputTourCodeMakeOrderField.setText("");  
 }  
 });  
  
 */\*\*Билеты\*/* ticketClientViewBtn.setOnAction(ActionEvent -> {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewTicket");  
 ArrayList<Ticket> ticketArrayList = (ArrayList<Ticket>) connect.readObjList().clone();  
 ArrayList<Ticket> tickets = new ArrayList<>();  
  
 String clientCode = profile.getClientCode();  
  
 for (Ticket t : ticketArrayList) {  
 if (clientCode.equals(t.getUserCode())) {  
 tickets.add(t);  
 }  
 }  
  
 if (tickets.isEmpty()) {  
 ticketTableColumn.getItems().clear();  
 } else {  
 ObservableList<Ticket> observableList = FXCollections.observableArrayList(tickets);  
 ticketTableColumn.setItems(observableList);  
 ticketTableColumn.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 ticketTableColumn.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("ticketCode"));  
 ticketTableColumn.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("userCode"));  
 ticketTableColumn.getColumns().get(3).setCellValueFactory(new PropertyValueFactory("transportType"));  
 ticketTableColumn.getColumns().get(4).setCellValueFactory(new PropertyValueFactory("departurePoint"));  
 ticketTableColumn.getColumns().get(5).setCellValueFactory(new PropertyValueFactory("arrivalPoint"));  
 ticketTableColumn.getColumns().get(6).setCellValueFactory(new PropertyValueFactory("departureData"));  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
  
 });  
  
 */\*\*Туры\*/* tourViewBtn.setOnAction(ActionEvent -> {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewTour");  
 ArrayList<Tour> tourArrayList = (ArrayList<Tour>) connect.readObjList().clone();  
 ObservableList<Tour> observableList = FXCollections.observableArrayList(tourArrayList);  
 tabViewTours.setItems(observableList);  
 tabViewTours.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("countryName"));  
 tabViewTours.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("cityName"));  
 tabViewTours.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("price"));  
 tabViewTours.getColumns().get(3).setCellValueFactory(new PropertyValueFactory("duration"));  
 tabViewTours.getColumns().get(4).setCellValueFactory(new PropertyValueFactory("tourCode"));  
 tabViewTours.getColumns().get(5).setCellValueFactory(new PropertyValueFactory("tourDate"));  
 tabViewTours.getColumns().get(6).setCellValueFactory(new PropertyValueFactory("tourName"));  
 tabViewTours.getColumns().get(7).setCellValueFactory(new PropertyValueFactory("tourType"));  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 });  
  
 */\*\*Заказы\*/* viewOrderClientBtn.setOnAction(ActionEvent -> {  
 try {  
 connect.writeLine("view");  
 connect.writeLine("viewOrder");  
 connect.writeLine(profile.getClientCode());  
 final String flagOrder = connect.readLine();  
 if (flagOrder.equals("true")) {  
 ArrayList<Order> orders = (ArrayList<Order>) connect.readObjList().clone();  
 ObservableList<Order> orderObservableList = FXCollections.observableArrayList(orders);  
 tabViewOrdersClient.setItems(orderObservableList);  
 tabViewOrdersClient.getColumns().get(0).setCellValueFactory(new PropertyValueFactory("id"));  
 tabViewOrdersClient.getColumns().get(1).setCellValueFactory(new PropertyValueFactory("clientCode"));  
 tabViewOrdersClient.getColumns().get(2).setCellValueFactory(new PropertyValueFactory("tourCode"));  
 } else if (flagOrder.equals("false")) {  
 tabViewOrdersClient.getItems().clear();  
 errorOrderClientLabel.setText("У вас нет заказов");  
 } else {  
 tabViewOrdersClient.getItems().clear();  
 }  
 } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 });  
  
 */\*\*Отменить заказа\*/* cancelOrderClientBtn.setOnAction(ActionEvent -> {  
 try {  
 String cancelOrderId = cancelOrderClientIdField.getText().trim();  
 if (Check.isNumber(cancelOrderId)) {  
 connect.writeLine("view");  
 connect.writeLine("viewOrder");  
 connect.writeLine(profile.getClientCode());  
 final String flagOrder = connect.readLine();  
 if (flagOrder.equals("true")) {  
  
 connect.readObjList();  
 connect.writeLine("delete");  
 connect.writeLine("deleteOrder");  
 connect.writeLine(cancelOrderId);  
 String msg = connect.readLine();  
  
 if (msg.equals("true")) {  
 errorOrderClientLabel.setText("Заказ успешно удалён");  
 } else {  
 cancelOrderClientIdField.setText("");  
 errorOrderClientLabel.setText("Заказ не удалён");  
 }  
 } else {  
 cancelOrderClientIdField.setText("");  
 errorOrderClientLabel.setText("У вас нету такого заказа");  
 }  
 } else {  
 Shake shakeOrderId = new Shake(cancelOrderClientIdField);  
 shakeOrderId.playAnim();  
 cancelOrderClientIdField.setText("");  
 errorOrderClientLabel.setText("Введите id заказа ");  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 } catch (ClassNotFoundException e) {  
 new MyException(e);  
 } finally {  
 cancelOrderClientIdField.setText("");  
 }  
 });  
 }  
}

package com.example.client;  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import java.io.IOException;  
*/\*\*@author Atayev I.M. (ataewisma@gmial@gmail.com)\*/*public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("mainView.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 600, 400);  
 stage.setTitle("Travel");  
 stage.setScene(scene);  
 stage.show();  
 }  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

package com.example.client;  
import java.io.IOException;  
import java.net.URL;  
import java.util.ResourceBundle;  
import com.example.model.animation.Shake;  
import com.example.model.check.Check;  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.dialog.InputDialog;  
import com.example.model.myexception.MyException;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
import javafx.scene.control.Tooltip;  
import javafx.stage.Stage;  
  
public class MainController {  
  
 public static Connect *connect*;  
  
 public static Client *client*;  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Tooltip addTourToolTip;  
  
 @FXML  
 private Tooltip passToolTip;  
  
 @FXML  
 private Tooltip loginToolTip;  
  
 @FXML  
 private TextField login\_field;  
  
 @FXML  
 private PasswordField password\_field;  
  
 @FXML  
 private Button authSignInButton;  
  
 @FXML  
 private Button loginSignUpButton;  
  
  
 static {  
 *connect* = new Connect("127.0.0.1", 1122);  
 }  
  
 @FXML  
 void initialize() {  
 authSignInButton.setTooltip(addTourToolTip);  
 password\_field.setTooltip(passToolTip);  
 login\_field.setTooltip(loginToolTip);  
 authSignInButton.setOnAction( actionEvent -> {  
 try {  
  
 String login = login\_field.getText().trim();  
 String pass = password\_field.getText().trim();  
  
 if (Check.*isString*(login) && Check.*isString*(pass)) {  
  
 *connect*.writeLine("signIn");  
 *connect*.writeLine(login);  
 *connect*.writeLine(pass);  
  
 String flag = *connect*.readLine();  
 String flagAdminOrClient = *connect*.readLine();  
  
 if (flag.equals("true")) {  
  
 System.*out*.println(flag);  
 System.*out*.println(flagAdminOrClient);  
  
 if (flagAdminOrClient.equals("adminUI")) {  
  
 System.out.println("admin");  
 openNewScene("admin-ui.fxml");  
  
 } else if (flagAdminOrClient.equals("clientUI")) {  
 Client c = (Client) connect.readObj();  
 System.out.println(c.toString());  
 client = c;  
 System.out.println("client");  
 openNewScene("client-ui.fxml");  
  
 } else {  
 login\_field.setText("");  
 password\_field.setText("");  
 System.out.println("do not user ");  
 }  
 } else {  
 Shake shakeLogin = new Shake(login\_field);  
 Shake shakePass = new Shake(password\_field);  
 shakeLogin.playAnim();  
 shakePass.playAnim();  
  
 login\_field.setText("");  
 password\_field.setText("");  
  
 login = null;  
 pass = null;  
 flag = null;  
 flagAdminOrClient = null;  
  
 }  
 } else {  
 login = null;  
 pass = null;  
  
 Shake shakeLogin = new Shake(login\_field);  
 Shake shakePass = new Shake(password\_field);  
 shakeLogin.playAnim();  
 shakePass.playAnim();  
 }  
  
  
 */\* connect.writeLine("signIn");  
 connect.writeLine(login);  
 connect.writeLine(pass);  
  
 String flag = connect.readLine();  
 String flagAdminOrClient = connect.readLine();  
  
 if (flag.equals("true")) {  
  
 System.out.println(flag);  
 System.out.println(flagAdminOrClient);  
  
 if (flagAdminOrClient.equals("adminUI")) {  
  
 System.out.println("admin");  
 openNewScene("admin-ui.fxml");  
 //handleButtonClick("admin-ui.fxml");  
 System.out.println(flagAdminOrClient);  
  
 } else if (flagAdminOrClient.equals("clientUI")) {  
 Client c = (Client) connect.readObj();  
 client = c;  
 System.out.println("client");  
 openNewScene("client-ui.fxml");  
 System.out.println(flagAdminOrClient);  
  
 } else {  
 System.out.println("do not user ");  
 }  
 } else if (flag.equals("false")) {  
 Shake shakeLogin = new Shake(login\_field);  
 Shake shakePass = new Shake(password\_field);  
 shakeLogin.playAnim();  
 shakePass.playAnim();  
 }\*/* } catch (IOException | ClassNotFoundException e) {  
 new MyException(e);  
 }  
 });  
 }  
  
 public void openNewScene(String window) {  
 try {  
 loginSignUpButton.getScene().getWindow().hide();  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource(window));  
 loader.load();  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.showAndWait();  
 } catch (IOException e) {  
 new MyException(e);  
 }  
 }  
  
 public void getOpenSignUp(ActionEvent actionEvent) {  
 new InputDialog(actionEvent, "sign-up-ui.fxml", 678, 400);  
 }  
  
}

package com.example.client;  
  
import java.io.IOException;  
import java.net.URL;  
import java.util.ResourceBundle;  
import com.example.model.animation.Shake;  
import com.example.model.check.Check;  
import com.example.model.client.Client;  
import com.example.model.connect.Connect;  
import com.example.model.myexception.MyException;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.Node;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
public class SignUpController {  
  
 private Connect connect = MainController.*connect*;  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private TextField singUpFIO;  
  
 @FXML  
 private TextField singUpPhoneNumber;  
  
 @FXML  
 private PasswordField singUpPassword;  
  
 @FXML  
 private Button singUpButton;  
  
 @FXML  
 private Button closeSignUp;  
 @FXML  
 private TextField singUpMail;  
 @FXML  
 private TextField singUpLogin;  
 @FXML  
 private TextField singUpPassportId;  
 @FXML  
 private Label lableErroAddClient;  
 @FXML  
 void clickOnSignUpBtn(ActionEvent event) {  
 try {  
 Shake shakeFIO = new Shake(singUpFIO);  
 Shake shakeMobileNumber = new Shake(singUpPhoneNumber);  
 Shake shakePassword = new Shake(singUpPassword);  
 Shake shakeMail = new Shake(singUpMail);  
 Shake shakeLogin = new Shake(singUpLogin);  
 Shake shakePassportId = new Shake(singUpPassportId);  
 String fio = singUpFIO.getText().trim();  
 String phoneNumber = singUpPhoneNumber.getText().trim();  
 String password = singUpPassword.getText().trim();  
 String mail = singUpMail.getText().trim();  
 String login = singUpLogin.getText().trim();  
 String passportId = singUpPassportId.getText().trim();  
 if (Check.*isNumber*(phoneNumber) && !fio.equals("") && !password.equals("") && !mail.equals("") && !login.equals("") && !passportId.equals("")) {  
 Client client = new Client();  
 int a = 100;  
 int b = 1000;  
 int x = a + (int) (Math.*random*() \* ((b - a) + 1));  
 client.setFIO(fio);  
 client.setClientCode("C" + String.*valueOf*(x));  
 client.setMobileNumber(phoneNumber);  
 client.setPassword(password);  
 client.setMail(mail);  
 client.setLogin(login);  
 client.setPassportId(passportId);  
 client.setFlag(2);  
 System.*out*.println(client);  
 connect.writeLine("signUp");  
 connect.writeObj(client);  
 String flag = connect.readLine();  
 if (flag.equals("false")) {  
 lableErroAddClient.setText(" ошибка при регистрации вы не зарегистрированы, Обратитесь админу !!! ");  
 } else if (flag.equals("true")) {  
 lableErroAddClient.setText("Регистрация прошла успешна.");  
 }  
 } else {  
 shakeFIO.playAnim();  
 shakeLogin.playAnim();  
 shakeMail.playAnim();  
 shakeMobileNumber.playAnim();  
 shakePassword.playAnim();  
 shakePassportId.playAnim();  
 }  
 } catch (IOException e) {  
 new MyException(e);  
 }  
 }  
 @FXML  
 void initialize() {  
  
 }  
 @FXML  
 void closeSignUp(ActionEvent event) {  
 closeStage(event);  
 }  
 private void closeStage(ActionEvent event) {  
 Node source = (Node) event.getSource();  
 Stage stage = (Stage) source.getScene().getWindow();  
 stage.close();  
 }  
}