# **Приложение**

1. Controllers
2. package com.example.hotelappwithhibernate.controllers;  
     
   import javafx.event.ActionEvent;  
   import javafx.fxml.FXML;  
   import javafx.fxml.FXMLLoader;  
   import javafx.scene.Node;  
   import javafx.scene.Parent;  
   import javafx.scene.Scene;  
   import javafx.scene.control.Label;  
   import javafx.stage.Stage;  
     
   import java.io.IOException;  
   import java.util.Objects;  
     
   public class AppController {  
    private Stage stage;  
    private Scene scene;  
     
    @FXML  
    private void setGuestScene(ActionEvent event) throws IOException {  
    Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/com.example.hotelappwithhibernate/scenes/guestScene.fxml")));  
    stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
    scene = new Scene(root,1110,625);  
     
    stage.setScene(scene);  
    stage.show();  
    }  
     
    @FXML  
    private void setRoomScene(ActionEvent event) throws IOException{  
    Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/com.example.hotelappwithhibernate/scenes/roomScene.fxml")));  
    stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
    scene = new Scene(root,910,510);  
     
    stage.setScene(scene);  
    stage.show();  
    }  
     
    @FXML  
    private void setMaidScene(ActionEvent event) throws IOException{  
    Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass()  
    .getResource("/com.example.hotelappwithhibernate/scenes/maidScene.fxml")));  
    stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
    scene = new Scene(root,910,510);  
     
    stage.setScene(scene);  
    stage.show();  
    }  
     
    @FXML  
    private void setScheduleScene(ActionEvent event) throws IOException{  
    Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass()  
    .getResource("/com.example.hotelappwithhibernate/scenes/scheduleScene.fxml")));  
    stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
    scene = new Scene(root,910,510);  
     
    stage.setScene(scene);  
    stage.show();  
    }  
    @FXML  
    private void setServiceScene(ActionEvent event) throws IOException{  
    Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass()  
    .getResource("/com.example.hotelappwithhibernate/scenes/serviceScene.fxml")));  
    stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
    scene = new Scene(root,910,510);  
     
    stage.setScene(scene);  
    stage.show();  
    }  
   }

package com.example.hotelappwithhibernate.controllers;  
  
import com.example.hotelappwithhibernate.DateToStringConverter;  
import com.example.hotelappwithhibernate.dao.AddressDao;  
import com.example.hotelappwithhibernate.dao.GuestDao;  
import com.example.hotelappwithhibernate.dao.PassportDao;  
import com.example.hotelappwithhibernate.dao.RoomDao;  
import com.example.hotelappwithhibernate.models.\*;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleObjectProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.value.ObservableValue;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.ComboBoxTableCell;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
import javafx.util.Callback;  
import javafx.util.converter.IntegerStringConverter;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.io.IOException;  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.List;  
import java.util.Objects;  
  
public class GuestController {  
//textfields  
 public TextField findTextField;  
 public TextField guestDDepartmentAdd;  
 public TextField guestGivenByAdd;  
 public TextField guestDEntryAdd;  
 public ComboBox<Room> guestNRoomAdd;  
 public TextField guestParkingAdd;  
 public TextField guestNAutoAdd;  
 public TextField guestIssuanceAdd;  
 public TextField guestBirthAdd;  
 public TextField guestNPassportAdd;  
 public TextField guestBuildingAdd;  
 public TextField guestStreetAdd;  
 public TextField guestCityAdd;  
 public TextField guestCountryAdd;  
 public TextField guestPatronymicAdd;  
 public TextField guestSurnameAdd;  
 public TextField guestNameAdd;  
 //Columns  
 public TableColumn<Guest, Integer> tableId;  
 public TableColumn<Guest, String> tableName;  
 public TableColumn<Guest, String> tableSurname;  
 public TableColumn<Guest, String> tablePatronymic;  
 public TableColumn<Guest, String> tableGender;  
 public TableColumn<Guest, Date> tableBirthDate;  
 public TableColumn<Guest, String> tableCountry;  
 public TableColumn<Guest, String> tableCity;  
 public TableColumn<Guest, String> tableStreet;  
 public TableColumn<Guest, String> tableBuilding;  
 public TableColumn<Guest, Integer> tableNPassport;  
 public TableColumn<Guest, Date> tableIssuance;  
 public TableColumn<Guest, String> tableGivenBy;  
 public TableColumn<Guest, String> tableNAuto;  
 public TableColumn<Guest, Integer> tableParking;  
 public TableColumn<Guest, Integer> tableNRoom;  
 public TableColumn<Guest, Date> tableDEntry;  
 public TableColumn<Guest, Date> tableDDepartment;  
 public TableColumn<Guest, String> tableDelete;  
  
 public TableView<Guest> table;  
 public TitledPane titledPane;  
 public TextField guestGenderAdd;  
 private Stage stage;  
 private Scene scene;  
 public ObservableList<Guest> obsGuestList = FXCollections.*observableArrayList*();  
 ObservableList<Room> rooms = FXCollections.*observableArrayList*();  
  
 Configuration configuration = new Configuration().addAnnotatedClass(Guest.class)  
 .addAnnotatedClass(Address.class)  
 .addAnnotatedClass(Passport.class)  
 .addAnnotatedClass(Room.class)  
 .addAnnotatedClass(Service.class)  
 .addAnnotatedClass(Maid.class)  
 .addAnnotatedClass(Schedule.class)  
 .addAnnotatedClass(Service.class);  
 SessionFactory sessionFactory = configuration.buildSessionFactory();  
 private final GuestDao guestDao = new GuestDao(sessionFactory);  
  
 @FXML  
 private void initialize(){  
 titledPaneAnimation();  
 tableCreation();  
 getRoom();  
 table.setEditable(true);  
 table.getSelectionModel().setCellSelectionEnabled(true);  
}  
 private void titledPaneAnimation(){  
 titledPane.setAnimated(true);  
 }  
  
 private Date dateWork(String str){  
 DateFormat df = new SimpleDateFormat("dd-MM-yyyy");  
 Date date = null;  
 try {  
 date = df.parse(str);  
 return date;  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 public void create() {  
 table.getItems().clear();  
 if(!guestNameAdd.getText().isEmpty() && !guestSurnameAdd.getText().isEmpty()  
 && !guestPatronymicAdd.getText().isEmpty() && !guestCountryAdd.getText().isEmpty()  
 && !guestCityAdd.getText().isEmpty() && !guestStreetAdd.getText().isEmpty()  
 && !guestBuildingAdd.getText().isEmpty() && !guestNPassportAdd.getText().isEmpty()  
 && !guestIssuanceAdd.getText().isEmpty() && !guestGivenByAdd.getText().isEmpty()  
 && !guestGenderAdd.getText().isEmpty()  
 && !guestBirthAdd.getText().isEmpty() && !guestNAutoAdd.getText().isEmpty()  
 && !guestParkingAdd.getText().isEmpty() && !guestDEntryAdd.getText().isEmpty()  
 && !guestDDepartmentAdd.getText().isEmpty()){  
 Date dateBirth = dateWork(guestBirthAdd.getText());  
 Date dateEntry = dateWork(guestDEntryAdd.getText());  
 Date dateDepart = dateWork(guestDDepartmentAdd.getText());  
  
 Guest guest = new Guest(guestNameAdd.getText(),guestSurnameAdd.getText(),  
 guestPatronymicAdd.getText(),guestGenderAdd.getText(),dateBirth,  
 Integer.*parseInt*(guestParkingAdd.getText()),guestNAutoAdd.getText(),dateEntry,  
 dateDepart);  
  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 Address address = new Address(guestCountryAdd.getText(),guestCityAdd.getText(),  
 guestStreetAdd.getText(),guestBuildingAdd.getText());  
 address.setGuest(guest);  
 addressDao.save(address);  
  
 PassportDao passportDao = new PassportDao(sessionFactory);  
 Date date = dateWork(guestIssuanceAdd.getText());  
 Passport passport = new Passport(Integer.*parseInt*(guestNPassportAdd.getText()),  
 date, guestGivenByAdd.getText());  
 passport.setGuest(guest);  
 passportDao.save(passport);  
 /\*List<Guest> guestsList = guestNRoomAdd.getValue().getGuests();  
 guestsList.add(guest);\*/  
 /\* if(guestsList.isEmpty()){  
 guestsList = new LinkedList<>();  
 guestsList.add(guest);  
 }else{\*/  
 //guestsList.add(guest);  
 /\*}\*/  
 //guestNRoomAdd.getValue().setGuests(guestsList);  
  
 guest.setRoom(guestNRoomAdd.getValue());  
  
 guest.setAddress(address);  
 guest.setPassport(passport);  
  
 guestDao.save(guest);  
 tableCreation();  
 }  
 }  
  
 private void getRoom(){  
 RoomDao roomDao = new RoomDao(sessionFactory);  
  
 rooms.setAll(roomDao.index());  
 guestNRoomAdd.setItems(rooms);  
 }  
  
  
  
 public void tableCreation() {  
 table.getItems().clear();  
  
 List<Guest> guestList = guestDao.index();  
  
 obsGuestList.addAll(guestList);  
  
 tableId.setCellValueFactory(new PropertyValueFactory<Guest, Integer>("Id"));  
 tableName.setCellValueFactory(new PropertyValueFactory<Guest, String>("Name"));  
 tableName.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tableSurname.setCellValueFactory(new PropertyValueFactory<Guest, String>("Surname"));  
 tableSurname.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tablePatronymic.setCellValueFactory(new PropertyValueFactory<Guest, String>("Patronymic"));  
 tablePatronymic.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tableGender.setCellValueFactory(new PropertyValueFactory<Guest, String>("Gender"));  
 tableGender.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 /\*tableBirthDate.setCellValueFactory(new PropertyValueFactory<Guest, Date>("BirthDate"));  
 tableBirthDate.setCellFactory(TextFieldTableCell.forTableColumn(new DateStringConverter()));\*/  
  
 tableNAuto.setCellValueFactory(new PropertyValueFactory<Guest, String>("auto\_number"));  
 tableNAuto.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tableParking.setCellValueFactory(new PropertyValueFactory<Guest, Integer>("parking\_lot\_number"));  
 tableParking.setCellFactory(TextFieldTableCell.*forTableColumn*(new IntegerStringConverter()));  
  
 /\*tableDEntry.setCellValueFactory(new PropertyValueFactory<Guest, Date>("entry\_date"));  
 tableDEntry.setCellFactory(TextFieldTableCell.forTableColumn(new DateStringConverter()));  
  
 tableDDepartment.setCellValueFactory(new PropertyValueFactory<Guest, Date>("department\_date"));  
 tableDDepartment.setCellFactory(TextFieldTableCell.forTableColumn(new DateStringConverter()));\*/  
  
  
 tableCountry.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Guest, String> g) {  
 return new SimpleStringProperty( g.getValue().getAddress().getCountry());  
 }  
 });  
  
 tableCountry.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableCity.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Guest, String> g) {  
 return new SimpleStringProperty( g.getValue().getAddress().getCity());  
 }  
 });  
 tableCity.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableStreet.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableStreet.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Guest, String> g) {  
 return new SimpleStringProperty( g.getValue().getAddress().getStreet());  
 }  
 });  
  
 tableBuilding.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableBuilding.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Guest,String> g) {  
 return new SimpleStringProperty(g.getValue().getAddress().getBuilding());  
 }  
 });  
  
 tableNPassport.setCellFactory(TextFieldTableCell.*forTableColumn*(new IntegerStringConverter()));  
 tableNPassport.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, Integer>, ObservableValue<Integer>>() {  
 @Override  
 public ObservableValue<Integer> call(TableColumn.CellDataFeatures<Guest,Integer> g) {  
 return new SimpleObjectProperty<Integer>(g.getValue().getPassport().getNumber());  
 }  
 });  
  
  
 tableIssuance.setCellFactory(TextFieldTableCell.*forTableColumn*(new DateToStringConverter()));  
 tableIssuance.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, Date>, ObservableValue<Date>>() {  
 @Override  
 public ObservableValue<Date> call(TableColumn.CellDataFeatures<Guest,Date> g) {  
 return new SimpleObjectProperty<Date>(g.getValue().getPassport().getIssuance());  
 }  
 });  
  
 tableBirthDate.setCellFactory(TextFieldTableCell.*forTableColumn*(new DateToStringConverter()));  
 tableBirthDate.setCellValueFactory(g -> new SimpleObjectProperty<>(g.getValue().getBirth\_date()));  
  
 tableDEntry.setCellFactory(TextFieldTableCell.*forTableColumn*(new DateToStringConverter()));  
 tableDEntry.setCellValueFactory(g -> new SimpleObjectProperty<Date>(g.getValue().getDate\_of\_entry()));  
  
 tableDDepartment.setCellFactory(TextFieldTableCell.*forTableColumn*(new DateToStringConverter()));  
 tableDDepartment.setCellValueFactory(g -> new SimpleObjectProperty<Date>(g.getValue().getDeparture\_date()));  
  
 tableGivenBy.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableGivenBy.setCellValueFactory(g -> new SimpleStringProperty(g.getValue().getPassport().getGivenBy()));  
 ////  
 //ObservableList<Room> rNumberList = FXCollections.observableArrayList(Room.values());  
 ObservableList<Integer> roomNumbers = FXCollections.*observableArrayList*();  
 getRoom();  
 for(Room room : rooms){  
 roomNumbers.add(room.getNumber());  
 System.*out*.println(room.getNumber());  
 }  
 tableNRoom.setCellFactory(ComboBoxTableCell.*forTableColumn*(roomNumbers));  
 tableNRoom.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, Integer>, ObservableValue<Integer>>() {  
  
 @Override  
 public ObservableValue<Integer> call(TableColumn.CellDataFeatures<Guest, Integer> param) {  
 /\* Guest guest = param.getValue();  
 int roomNumber = guest.getRoom().getNumber();\*/  
 //Gender gender = Gender.getByCode(genderCode);  
 return new SimpleObjectProperty<Integer>(param.getValue().getRoom().getNumber());  
 }  
 });  
  
 //tableNRoom.setCellFactory(ComboBoxTableCell.forTableColumn());  
  
 /\*tableNRoom.setOnEditCommit((TableColumn.CellEditEvent<Guest, Integer> event) -> {  
 TablePosition<Guest, Integer> pos = event.getTablePosition();  
  
 int newNRoom = event.getNewValue();  
  
 int row = pos.getRow();  
 Guest guest = event.getTableView().getItems().get(row);  
  
 guest.getRoom().setNumber(newNRoom);  
 });\*/  
  
 //genderCol.setMinWidth(120);  
  
 ////  
 /\*tableNRoom.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));  
 tableNRoom.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Guest, Integer>, ObservableValue<Integer>>() {  
 @Override  
 public ObservableValue<Integer> call(TableColumn.CellDataFeatures<Guest, Integer> g) {  
 return new SimpleObjectProperty<Integer>(g.getValue().getRoom().getNumber());  
 }  
 });\*/  
  
  
 Callback<TableColumn<Guest, String>, TableCell<Guest, String>> cellDeleteFactory = (param) -> {  
 final TableCell<Guest, String> cell = new TableCell<Guest, String>() {  
 @Override  
 public void updateItem(String item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty) {  
 setGraphic(null);  
 setText(null);  
 } else {  
 Button deleteButton = new Button("Delete");  
 deleteButton.setOnAction((event) -> {  
 Guest guest = getTableView().getItems().get(getIndex());  
  
 guestDao.delete(guest.getId());  
 tableCreation();  
 });  
 setGraphic(deleteButton);  
 setText(null);  
 }  
 }  
 };  
 return cell;  
 };  
 tableDelete.setCellFactory(cellDeleteFactory);  
 table.setItems(obsGuestList);  
  
 }  
  
 public void onNameChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateName(guest.getId(),guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onSurnameChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateSurname(guest.getId(),guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onPatronymicChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updatePatronymic(guest.getId(),guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onGenderChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateGender(guest.getId(),guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onBirthChange(TableColumn.CellEditEvent<Guest, Date> guestDateCellEditEvent) {  
  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateBirth(guest.getId(), guestDateCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onCountryChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateCountry(guest.getId(),guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
  
  
 public void onCityChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateCity(guest.getId(), guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onStreetChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateStreet(guest.getId(), guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onBuildingChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateBuilding(guest.getId(), guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onNPassportChange(TableColumn.CellEditEvent<Guest, Integer> cellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateNPassport(guest.getId(), cellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onIssuanceChange(TableColumn.CellEditEvent<Guest, Date> guestDateCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateIssuance(guest.getId(), guestDateCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onGivenByChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateGivenBy(guest.getId(), guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
 public void onParkingChange(TableColumn.CellEditEvent<Guest, Integer> guestIntCellEditEvent){  
  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateParking(guest.getId(), guestIntCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onNAutoChange(TableColumn.CellEditEvent<Guest, String> guestStringCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateNAuto(guest.getId(), guestStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
  
 public void onNRoomChange() {  
 tableNRoom.setOnEditCommit((TableColumn.CellEditEvent<Guest, Integer> event) -> {  
 TablePosition<Guest, Integer> pos = event.getTablePosition();  
  
 int newNRoom = event.getNewValue();  
  
 int row = pos.getRow();  
 Guest guest = event.getTableView().getItems().get(row);  
 guestDao.updateNRoom(guest.getId(),newNRoom);  
 guest.getRoom().setNumber(newNRoom);  
  
 });  
 tableCreation();  
 }  
  
 public void onDEntryChange(TableColumn.CellEditEvent<Guest, Date> guestDateCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateDEntry(guest.getId(), guestDateCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onDDepartmentChange(TableColumn.CellEditEvent<Guest, Date> guestDateCellEditEvent) {  
 Guest guest = table.getSelectionModel().getSelectedItem();  
 guestDao.updateDDepartment(guest.getId(), guestDateCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
  
  
 public void backAction(ActionEvent event)throws IOException {  
 Parent root = FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/com.example.hotelappwithhibernate/scenes/app.fxml")));  
 stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 scene = new Scene(root,910,510);  
  
 stage.setScene(scene);  
 stage.show();  
 }  
  
  
 public void findInTable() {  
 if(findTextField.getText().isEmpty()){  
 tableCreation();  
 return;  
 }  
 List<Guest> resList = guestDao.findByFields(findTextField.getText());  
 obsGuestList.clear();  
 obsGuestList.addAll(resList);  
 table.setItems(obsGuestList);  
 }  
}

package com.example.hotelappwithhibernate.controllers;  
  
import com.example.hotelappwithhibernate.dao.AddressDao;  
import com.example.hotelappwithhibernate.dao.MaidDao;  
import com.example.hotelappwithhibernate.models.\*;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.value.ObservableValue;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
import javafx.util.Callback;  
import javafx.util.converter.IntegerStringConverter;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
import java.io.IOException;  
import java.util.List;  
import java.util.Objects;  
  
public class MaidController {  
 public TextField findTextField;  
 public TextField maidBuildingAdd;  
 public TextField maidStreetAdd;  
 public TextField maidCityAdd;  
 public TextField maidCountryAdd;  
 public TextField maidPatronymicAdd;  
 public TextField maidSurnameAdd;  
  
  
 private Stage stage;  
 private Scene scene;  
  
 Configuration configuration = new Configuration().addAnnotatedClass(Maid.class)  
 .addAnnotatedClass(Address.class)  
 .addAnnotatedClass(Guest.class)  
 .addAnnotatedClass(Passport.class)  
 .addAnnotatedClass(Room.class)  
 .addAnnotatedClass(Service.class)  
 .addAnnotatedClass(Schedule.class)  
 .addAnnotatedClass(Service.class);  
  
 SessionFactory sessionFactory = configuration.buildSessionFactory();  
 private final MaidDao maidDao = new MaidDao(sessionFactory);  
  
 public TableView<Maid> table;  
  
 public TitledPane titledPane;  
 public TextField maidNameAdd;  
 public TableColumn<Maid,Integer> tableId;  
 public TableColumn<Maid, String> tableName;  
 public TableColumn<Maid,String> tableSurname;  
  
 public TableColumn<Maid, String> tablePatronymic;  
 public TableColumn<Maid,String> tableCountry;  
 public TableColumn<Maid, String> tableCity;  
 public TableColumn<Maid, String> tableStreet;  
 public TableColumn<Maid, String> tableBuilding;  
  
 public TableColumn<Maid,String> tableDelete;  
  
 public ObservableList<Maid> obsMaidList = FXCollections.*observableArrayList*();  
  
  
 @FXML  
 private void initialize(){  
 titledPaneAnimation();  
 tableCreation();  
 //drawTable();  
 table.setEditable(true);  
 table.getSelectionModel().setCellSelectionEnabled(true);  
 //tableNumber.setCellFactory(TextFieldTableCell.forTableColumn());  
 }  
  
 @FXML  
 private void create(){  
 table.getItems().clear();  
 if(!maidNameAdd.getText().isEmpty() && !maidSurnameAdd.getText().isEmpty()  
 && !maidPatronymicAdd.getText().isEmpty() && !maidCountryAdd.getText().isEmpty()  
 && !maidCityAdd.getText().isEmpty() && !maidStreetAdd.getText().isEmpty()  
 && !maidBuildingAdd.getText().isEmpty()){  
  
 Maid maid = new Maid(maidNameAdd.getText(),maidSurnameAdd.getText(),maidPatronymicAdd.getText());  
  
 Address address = new Address(maidCountryAdd.getText(),maidCityAdd.getText(),  
 maidStreetAdd.getText(),maidBuildingAdd.getText());  
  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 address.setMaid(maid);  
  
 addressDao.save(address);  
  
 maid.setAddress(address);  
  
 maidDao.save(maid);  
 tableCreation();  
 }  
  
 }  
  
  
 private void titledPaneAnimation(){  
 titledPane.setAnimated(true);  
 }  
  
 public void tableCreation() {  
 table.getItems().clear();  
  
 List<Maid> maidList = maidDao.index();  
  
 obsMaidList.addAll(maidList);  
  
 tableId.setCellValueFactory(new PropertyValueFactory<Maid, Integer>("Id"));  
 tableName.setCellValueFactory(new PropertyValueFactory<Maid, String>("Name"));  
 tableName.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tableSurname.setCellValueFactory(new PropertyValueFactory<Maid, String>("Surname"));  
 tableSurname.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tablePatronymic.setCellValueFactory(new PropertyValueFactory<Maid, String>("Patronymic"));  
 tablePatronymic.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tableCountry.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Maid, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Maid, String> m) {  
 return new SimpleStringProperty( m.getValue().getAddress().getCountry());  
 }  
 });  
 tableCountry.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableCity.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Maid, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Maid, String> m) {  
 return new SimpleStringProperty( m.getValue().getAddress().getCity());  
 }  
 });  
 tableCity.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableStreet.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableStreet.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Maid, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Maid, String> m) {  
 return new SimpleStringProperty( m.getValue().getAddress().getStreet());  
 }  
 });  
  
 tableBuilding.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 tableBuilding.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Maid, String>, ObservableValue<String>>() {  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Maid,String> m) {  
 return new SimpleStringProperty(m.getValue().getAddress().getBuilding());  
 }  
 });  
  
 /\*tableCountry.setCellValueFactory(new PropertyValueFactory<Address, String>("Country"));  
 tableCity.setCellValueFactory(new PropertyValueFactory<Address, String>("City"));  
 tableCity.setCellValueFactory(new PropertyValueFactory<Address, String>("Street"));  
 tableBuilding.setCellValueFactory(new PropertyValueFactory<Address, Integer>("building"));\*/  
  
  
 //tableCountry.setCellFactory();  
 //tableCount.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));  
 Callback<TableColumn<Maid, String>, TableCell<Maid, String>> cellDeleteFactory = (param) -> {  
 final TableCell<Maid, String> cell = new TableCell<Maid, String>() {  
 @Override  
 public void updateItem(String item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty) {  
 setGraphic(null);  
 setText(null);  
 } else {  
 Button deleteButton = new Button("Delete");  
 deleteButton.setOnAction((event) -> {  
 Maid maid = getTableView().getItems().get(getIndex());  
  
 maidDao.delete(maid.getId());  
 tableCreation();  
 });  
 setGraphic(deleteButton);  
 setText(null);  
 }  
 }  
 };  
 return cell;  
 };  
 tableDelete.setCellFactory(cellDeleteFactory);  
 table.setItems(obsMaidList);  
  
 }  
  
 @FXML  
 private void backAction(ActionEvent event) throws IOException {  
 Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/com.example.hotelappwithhibernate/scenes/app.fxml")));  
 stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 scene = new Scene(root,910,510);  
  
 stage.setScene(scene);  
 stage.show();  
 }  
 @FXML  
 public void onNameChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updateName(maid.getId(),maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 @FXML  
 public void onSurnameChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updateSurname(maid.getId(),maidStringCellEditEvent.getNewValue());  
 // maid.setName(maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
 @FXML  
 public void onPatronymicChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updatePatronymic(maid.getId(),maidStringCellEditEvent.getNewValue());  
 // maid.setName(maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
 @FXML  
 public void onCountryChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updateCountry(maid.getId(),maidStringCellEditEvent.getNewValue());  
 //maid.setName(maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
 @FXML  
 public void onCityChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updateCity(maid.getId(),maidStringCellEditEvent.getNewValue());  
 //maid.setName(maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
 @FXML  
 public void onStreetChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updateStreet(maid.getId(),maidStringCellEditEvent.getNewValue());  
 // maid.setName(maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
 @FXML  
 public void onBuildingChange(TableColumn.CellEditEvent<Maid, String> maidStringCellEditEvent) {  
 Maid maid = table.getSelectionModel().getSelectedItem();  
 maidDao.updateBuilding(maid.getId(),maidStringCellEditEvent.getNewValue());  
 //maid.setName(maidStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void findInTable() {  
 if(findTextField.getText().isEmpty()){  
 tableCreation();  
 return;  
 }  
 List<Maid> resList = maidDao.findByFields(findTextField.getText());  
 obsMaidList.clear();  
 obsMaidList.addAll(resList);  
 table.setItems(obsMaidList);  
 }  
  
}

package com.example.hotelappwithhibernate.controllers;  
  
import com.example.hotelappwithhibernate.dao.RoomDao;  
import com.example.hotelappwithhibernate.models.\*;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
import javafx.util.Callback;  
import javafx.util.converter.IntegerStringConverter;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Objects;  
  
public class RoomController {  
  
 private Stage stage;  
 private Scene scene;  
 Configuration configuration = new Configuration().addAnnotatedClass(Room.class)  
 .addAnnotatedClass(Guest.class)  
 .addAnnotatedClass(Address.class)  
 .addAnnotatedClass(Passport.class)  
 .addAnnotatedClass(Service.class)  
 .addAnnotatedClass(Maid.class)  
 .addAnnotatedClass(Schedule.class)  
 .addAnnotatedClass(Service.class);  
 SessionFactory sessionFactory = configuration.buildSessionFactory();  
 private RoomDao roomDao = new RoomDao(sessionFactory);  
  
  
 public TextField findTextField;  
 public TextField roomCountAdd;  
 public TextField roomNumberAdd;  
  
  
 public TitledPane titledPane;  
  
  
 public TableView<Room> table;  
 public TableColumn<Room,Integer> tableId;  
 public TableColumn<Room, String> tableNumber;  
 public TableColumn<Room,Integer> tableCount;  
 public TableColumn<Room,String> tableDelete;  
  
 public ObservableList<Room> obsRoomList = FXCollections.*observableArrayList*();  
  
 @FXML  
 private void initialize(){  
 titledPaneAnimation();  
 tableCreation();  
 table.setEditable(true);  
 table.getSelectionModel().setCellSelectionEnabled(true);  
 }  
  
@FXML  
 private void create(){  
 table.getItems().clear();  
 if(!roomNumberAdd.getText().isEmpty() && !roomCountAdd.getText().isEmpty()){  
 Room room = new Room(Integer.*parseInt*(roomNumberAdd.getText()),Integer.*parseInt*(roomCountAdd.getText()));  
 roomDao.save(room);  
 tableCreation();  
 }  
 }  
  
  
 private void titledPaneAnimation(){  
 titledPane.setAnimated(true);  
 }  
  
 public void tableCreation() {  
 table.getItems().clear();  
 List<Room> roomList = roomDao.index();  
 obsRoomList.addAll(roomList);  
 tableId.setCellValueFactory(new PropertyValueFactory<Room, Integer>("Id"));  
 tableNumber.setCellValueFactory(new PropertyValueFactory<Room, String>("Number"));  
 tableCount.setCellValueFactory(new PropertyValueFactory<Room, Integer>("people\_count"));  
  
 tableCount.setCellFactory(TextFieldTableCell.*forTableColumn*(new IntegerStringConverter()));  
  
 Callback<TableColumn<Room, String>, TableCell<Room, String>> cellDeleteFactory = (param) -> {  
 final TableCell<Room, String> cell = new TableCell<Room, String>() {  
 @Override  
 public void updateItem(String item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty) {  
 setGraphic(null);  
 setText(null);  
 } else {  
 Button deleteButton = new Button("Delete");  
 deleteButton.setOnAction((event) -> {  
 Room room = getTableView().getItems().get(getIndex());  
  
 roomDao.delete(room.getId());  
 tableCreation();  
 });  
 setGraphic(deleteButton);  
 setText(null);  
 }  
 }  
 };  
 return cell;  
 };  
 tableDelete.setCellFactory(cellDeleteFactory);  
 table.setItems(obsRoomList);  
 }  
  
  
@FXML  
 public void onCountChange(TableColumn.CellEditEvent<Room, Integer> roomStringCellEditEvent) {  
 Room room = table.getSelectionModel().getSelectedItem();  
 roomDao.update(room.getId(),roomStringCellEditEvent.getNewValue());  
 room.setPeople\_count(roomStringCellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void findInTable() {  
 if(findTextField.getText().isEmpty()){  
 tableCreation();  
 return;  
 }  
 List<Room> resList = roomDao.findByFields(Integer.*parseInt*(findTextField.getText()));  
 obsRoomList.clear();  
 obsRoomList.addAll(resList);  
 table.setItems(obsRoomList);  
 }  
  
  
 @FXML  
 private void backAction(ActionEvent event) throws IOException {  
 Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/com.example.hotelappwithhibernate/scenes/app.fxml")));  
 stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 scene = new Scene(root,910,510);  
  
 stage.setScene(scene);  
 stage.show();  
 }  
}

package com.example.hotelappwithhibernate.controllers;  
  
import com.example.hotelappwithhibernate.dao.AddressDao;  
import com.example.hotelappwithhibernate.dao.MaidDao;  
import com.example.hotelappwithhibernate.dao.RoomDao;  
import com.example.hotelappwithhibernate.dao.ScheduleDao;  
import com.example.hotelappwithhibernate.models.\*;  
import javafx.beans.property.SimpleObjectProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.value.ObservableValue;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.ComboBoxTableCell;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
import javafx.util.Callback;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Objects;  
  
public class ScheduleController {  
 Configuration configuration = new Configuration().addAnnotatedClass(Schedule.class)  
 .addAnnotatedClass(Address.class)  
 .addAnnotatedClass(Guest.class)  
 .addAnnotatedClass(Passport.class)  
 .addAnnotatedClass(Room.class)  
 .addAnnotatedClass(Service.class)  
 .addAnnotatedClass(Maid.class)  
 .addAnnotatedClass(Service.class);  
  
 SessionFactory sessionFactory = configuration.buildSessionFactory();  
 private final ScheduleDao scheduleDao = new ScheduleDao(sessionFactory);  
 public TableView<Schedule> table;  
 public TextField findTextField;  
 public ComboBox<Room> scheduleRoomAdd;  
 public ComboBox<Maid> scheduleMaidAdd;  
 public TextField scheduleTimeAdd;  
 public TextField scheduleDayAdd;  
 public TitledPane titledPane;  
  
 public TableColumn<Schedule,Integer> tableId;  
 public TableColumn<Schedule, String> tableDay;  
 public TableColumn<Schedule, String> tableTime;  
 public TableColumn<Schedule, Maid> tableMaid;  
 public TableColumn<Schedule, Integer> tableNRoom;  
 public TableColumn<Schedule,String> tableDelete;  
  
 public ObservableList<Schedule> obsScheduleList = FXCollections.*observableArrayList*();  
 private ObservableList<Room> rooms = FXCollections.*observableArrayList*();  
 private ObservableList<Maid> maids = FXCollections.*observableArrayList*();  
 private Stage stage;  
 private Scene scene;  
  
  
  
  
 @FXML  
 private void initialize(){  
 titledPaneAnimation();  
 tableCreation();  
  
 table.setEditable(true);  
 table.getSelectionModel().setCellSelectionEnabled(true);  
  
 }  
 private void getRoom(){  
 RoomDao roomDao = new RoomDao(sessionFactory);  
  
 rooms.setAll(roomDao.index());  
 scheduleRoomAdd.setItems(rooms);  
 }  
  
 private void getMaid(){  
 MaidDao maidDao = new MaidDao(sessionFactory);  
  
 maids.setAll(maidDao.index());  
 scheduleMaidAdd.setItems(maids);  
 }  
  
 private void titledPaneAnimation(){  
 titledPane.setAnimated(true);  
 }  
  
 public void tableCreation() {  
 table.getItems().clear();  
  
 List<Schedule> scheduleList = scheduleDao.index();  
  
 obsScheduleList.addAll(scheduleList);  
  
 tableId.setCellValueFactory(new PropertyValueFactory<Schedule, Integer>("Id"));  
  
 tableDay.setCellValueFactory(new PropertyValueFactory<Schedule, String>("day"));  
 tableDay.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 tableTime.setCellValueFactory(new PropertyValueFactory<Schedule, String>("time"));  
 tableTime.setCellFactory(TextFieldTableCell.*forTableColumn*());  
  
 ObservableList<Integer> roomNumbers = FXCollections.*observableArrayList*();  
 getRoom();  
 for(Room room : rooms){  
 roomNumbers.add(room.getNumber());  
 System.*out*.println(room.getNumber());  
 }  
  
 tableNRoom.setCellFactory(ComboBoxTableCell.*forTableColumn*(roomNumbers));  
 tableNRoom.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Schedule, Integer>, ObservableValue<Integer>>() {  
  
 @Override  
 public ObservableValue<Integer> call(TableColumn.CellDataFeatures<Schedule, Integer> param) {  
 return new SimpleObjectProperty<Integer>(param.getValue().getRoom().getNumber());  
 }  
 });  
  
  
 ObservableList<String> maidNames = FXCollections.*observableArrayList*();  
 getMaid();  
 for(Maid maid : maids){  
 maidNames.add(maid.getName());  
 }  
  
 /\*tableMaid.setCellFactory(ComboBoxTableCell.forTableColumn(maidNames));  
 tableMaid.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Schedule, String>, ObservableValue<String>>() {  
  
 @Override  
 public ObservableValue<String> call(TableColumn.CellDataFeatures<Schedule, String> param) {  
 return new SimpleStringProperty(param.getValue().getMaid().getName());  
 }  
 });\*/  
 getMaid();  
 tableMaid.setCellFactory(ComboBoxTableCell.*forTableColumn*(maids));  
 tableMaid.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Schedule, Maid>, ObservableValue<Maid>>() {  
  
 @Override  
 public ObservableValue<Maid> call(TableColumn.CellDataFeatures<Schedule, Maid> param) {  
 return new SimpleObjectProperty<Maid>(param.getValue().getMaid());  
 }  
 });  
  
 Callback<TableColumn<Schedule, String>, TableCell<Schedule, String>> cellDeleteFactory = (param) -> {  
 final TableCell<Schedule, String> cell = new TableCell<Schedule, String>() {  
 @Override  
 public void updateItem(String item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty) {  
 setGraphic(null);  
 setText(null);  
 } else {  
 Button deleteButton = new Button("Delete");  
 deleteButton.setOnAction((event) -> {  
 Schedule schedule = getTableView().getItems().get(getIndex());  
  
 scheduleDao.delete(schedule.getId());  
 tableCreation();  
 });  
 setGraphic(deleteButton);  
 setText(null);  
 }  
 }  
 };  
 return cell;  
 };  
 tableDelete.setCellFactory(cellDeleteFactory);  
 table.setItems(obsScheduleList);  
 }  
  
 public void create() {  
 table.getItems().clear();  
 if(!scheduleDayAdd.getText().isEmpty() && !scheduleTimeAdd.getText().isEmpty()  
 && !scheduleRoomAdd.getSelectionModel().isEmpty() && !scheduleMaidAdd.getSelectionModel().isEmpty()){  
  
 Schedule schedule = new Schedule(scheduleDayAdd.getText(),scheduleTimeAdd.getText());  
  
 scheduleDao.save(schedule,scheduleRoomAdd.getValue(),scheduleMaidAdd.getValue());  
 tableCreation();  
 }  
 }  
  
 public void backAction(ActionEvent event) throws IOException {  
 Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/com.example.hotelappwithhibernate/scenes/app.fxml")));  
 stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 scene = new Scene(root,910,510);  
  
 stage.setScene(scene);  
 stage.show();  
 }  
  
   
  
 public void onDayChange(TableColumn.CellEditEvent<Schedule,String> cellEditEvent) {  
 Schedule schedule = table.getSelectionModel().getSelectedItem();  
 scheduleDao.updateDay(schedule.getId(),cellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onTimeChange(TableColumn.CellEditEvent<Schedule,String> cellEditEvent) {  
 Schedule schedule = table.getSelectionModel().getSelectedItem();  
 scheduleDao.updateTime(schedule.getId(),cellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 /\*public void onMaidChange(TableColumn.CellEditEvent<Schedule,String> cellEditEvent) {  
 tableNRoom.setOnEditCommit((TableColumn.CellEditEvent<Schedule, String> event) -> {  
 TablePosition<Schedule, String> pos = event.getTablePosition();  
  
 String newName = event.getNewValue();  
  
 int row = pos.getRow();  
 Schedule schedule = event.getTableView().getItems().get(row);  
 scheduleDao.updateMaid(schedule.getId(),newName);  
 //schedule.getRoom().setNumber(newNRoom);  
  
 });  
 tableCreation();  
 }\*/  
  
 public void onRoomChange() {  
 tableNRoom.setOnEditCommit((TableColumn.CellEditEvent<Schedule, Integer> event) -> {  
 TablePosition<Schedule, Integer> pos = event.getTablePosition();  
  
 int newNRoom = event.getNewValue();  
  
 int row = pos.getRow();  
 Schedule schedule = event.getTableView().getItems().get(row);  
 scheduleDao.updateNRoom(schedule.getId(),newNRoom);  
 //schedule.getRoom().setNumber(newNRoom);  
  
 });  
 tableCreation();  
 }  
  
 public void onMaidChange() {  
 tableMaid.setOnEditCommit((TableColumn.CellEditEvent<Schedule, Maid> event) -> {  
 TablePosition<Schedule, Maid> pos = event.getTablePosition();  
  
 Maid newMaid = event.getNewValue();  
  
 int row = pos.getRow();  
 Schedule schedule = event.getTableView().getItems().get(row);  
 scheduleDao.updateMaid(schedule.getId(),newMaid);  
 //schedule.getRoom().setNumber(newNRoom);  
  
 });  
 tableCreation();  
 }  
  
  
  
 public void findInTable() {  
 if(findTextField.getText().isEmpty()){  
 tableCreation();  
 return;  
 }  
 List<Schedule> resList = scheduleDao.findByFields(findTextField.getText());  
 obsScheduleList.clear();  
 obsScheduleList.addAll(resList);  
 table.setItems(obsScheduleList);  
 }  
}

package com.example.hotelappwithhibernate.controllers;  
  
import com.example.hotelappwithhibernate.dao.GuestDao;  
import com.example.hotelappwithhibernate.dao.ServiceDao;  
import com.example.hotelappwithhibernate.models.\*;  
import javafx.beans.property.SimpleObjectProperty;  
import javafx.beans.value.ObservableValue;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.ComboBoxTableCell;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.control.cell.TextFieldTableCell;  
import javafx.stage.Stage;  
import javafx.util.Callback;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.Objects;  
  
public class ServiceController {  
 private Stage stage;  
 private Scene scene;  
 Configuration configuration = new Configuration().addAnnotatedClass(Service.class)  
 .addAnnotatedClass(Guest.class)  
 .addAnnotatedClass(Address.class)  
 .addAnnotatedClass(Passport.class)  
 .addAnnotatedClass(Service.class)  
 .addAnnotatedClass(Maid.class)  
 .addAnnotatedClass(Schedule.class)  
 .addAnnotatedClass(Room.class);  
 SessionFactory sessionFactory = configuration.buildSessionFactory();  
 private ServiceDao serviceDao = new ServiceDao(sessionFactory);  
  
 public TableView<Service> table;  
 public TableColumn<Service,Integer> tableId;  
 public TableColumn<Service,Guest> tableGuest;  
 public TableColumn<Service, String> tableService;  
 public TableColumn<Service,String> tableDelete;  
 public TextField findTextField;  
 public ComboBox<Guest> guestAdd;  
 public TextField serviceAdd;  
 public TitledPane titledPane;  
  
 public ObservableList<Service> obsServiceList = FXCollections.*observableArrayList*();  
 private ObservableList<Guest> guests = FXCollections.*observableArrayList*();  
  
 public void backAction(ActionEvent event) throws IOException {  
 Parent root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/com.example.hotelappwithhibernate/scenes/app.fxml")));  
 stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 scene = new Scene(root,910,510);  
  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 @FXML  
 private void initialize(){  
 titledPaneAnimation();  
 tableCreation();  
 //drawTable();  
 table.setEditable(true);  
 table.getSelectionModel().setCellSelectionEnabled(true);  
 //tableNumber.setCellFactory(TextFieldTableCell.forTableColumn());  
 }  
  
 @FXML  
 private void create(){  
 table.getItems().clear();  
 if(!guestAdd.getSelectionModel().isEmpty() && !serviceAdd.getText().isEmpty()){  
 Service service = new Service(serviceAdd.getText());  
 Guest guest = (Guest)guestAdd.getValue();  
 serviceDao.save(service,guest);  
 tableCreation();  
 }  
  
 }  
 private void getGuests(){  
 GuestDao guestDao = new GuestDao(sessionFactory);  
  
 guests.setAll(guestDao.index());  
 guestAdd.setItems(guests);  
 }  
  
 private void titledPaneAnimation(){  
 titledPane.setAnimated(true);  
 }  
  
 public void tableCreation() {  
 table.getItems().clear();  
  
 List<Service> serviceList = serviceDao.index();  
  
 obsServiceList.addAll(serviceList);  
  
 tableId.setCellValueFactory(new PropertyValueFactory<Service, Integer>("Id"));  
 tableService.setCellValueFactory(new PropertyValueFactory<Service, String>("Name"));  
 tableService.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 getGuests();  
 /\*for(Guest guest : guests){  
 roomNumbers.add(room.getNumber());  
  
 }\*/  
 tableGuest.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Service,Guest>, ObservableValue<Guest>>() {  
 @Override  
 public ObservableValue<Guest> call(TableColumn.CellDataFeatures<Service, Guest> m) {  
 return new SimpleObjectProperty<Guest>( m.getValue().getGuests().get(0));  
 }  
 });  
 tableGuest.setCellFactory(ComboBoxTableCell.*forTableColumn*(guests));  
  
  
 Callback<TableColumn<Service, String>, TableCell<Service, String>> cellDeleteFactory = (param) -> {  
 final TableCell<Service, String> cell = new TableCell<Service, String>() {  
 @Override  
 public void updateItem(String item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty) {  
 setGraphic(null);  
 setText(null);  
 } else {  
 Button deleteButton = new Button("Delete");  
 deleteButton.setOnAction((event) -> {  
 Service service = getTableView().getItems().get(getIndex());  
 // Guest guest = (Guest) getTableView().getColumns().get(2).getCellData(service);  
  
 Guest guest = service.getGuests().get(0);  
 serviceDao.delete(service.getId(),guest);  
 tableCreation();  
 });  
 setGraphic(deleteButton);  
 setText(null);  
 }  
 }  
 };  
 return cell;  
 };  
 tableDelete.setCellFactory(cellDeleteFactory);  
 table.setItems(obsServiceList);  
  
 }  
  
 public void onServiceChange(TableColumn.CellEditEvent<Service,String> cellEditEvent) {  
 Service service = table.getSelectionModel().getSelectedItem();  
 serviceDao.updateName(service.getId(),cellEditEvent.getNewValue());  
 tableCreation();  
 }  
  
 public void onGuestChange() {  
 tableGuest.setOnEditCommit((TableColumn.CellEditEvent<Service, Guest> event) -> {  
 TablePosition<Service, Guest> pos = event.getTablePosition();  
  
 Guest guest = event.getNewValue();  
  
 int row = pos.getRow();  
 Service service = event.getTableView().getItems().get(row);  
 serviceDao.updateGuest(service.getId(),guest);  
 //schedule.getRoom().setNumber(newNRoom);  
  
 });  
 tableCreation();  
 }  
  
  
 public void findInTable() {  
 if(findTextField.getText().isEmpty()){  
 tableCreation();  
 return;  
 }  
 List<Service> resList = serviceDao.findByFields(findTextField.getText());  
 obsServiceList.clear();  
 obsServiceList.addAll(resList);  
 table.setItems(obsServiceList);  
 }  
}

2)DAO

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.models.Address;  
import com.example.hotelappwithhibernate.models.Maid;  
import com.example.hotelappwithhibernate.models.Room;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
  
import java.util.List;  
  
public class AddressDao {  
 private final SessionFactory sessionFactory;  
  
 public AddressDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
 }  
  
 public List<Address> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Address> res =session.createQuery("from Address",Address.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Address address){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 session.persist(address);  
  
 transaction.commit();  
 }  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
  
 session.remove(session.get(Address.class,id));  
 }  
  
 public void updateCountry(int id, String country){  
 Session session = sessionFactory.getCurrentSession();  
  
 Address address = session.get(Address.class,id);  
 address.setCountry(country);  
 }  
 public void updateCity(int id, String city){  
 Session session = sessionFactory.getCurrentSession();  
  
 Address address = session.get(Address.class,id);  
 address.setCity(city);  
 }  
  
 public void updateStreet(int id, String street){  
 Session session = sessionFactory.getCurrentSession();  
  
 Address address = session.get(Address.class,id);  
 address.setStreet(street);  
 }  
  
 public void updateBuilding(int id, String building){  
 Session session = sessionFactory.getCurrentSession();  
  
 Address address = session.get(Address.class,id);  
 address.setBuilding(building);  
 }  
}

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.DateToStringConverter;  
import com.example.hotelappwithhibernate.models.Guest;  
  
import com.example.hotelappwithhibernate.models.Room;  
import jakarta.persistence.Query;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import java.util.Date;  
import java.util.List;  
  
public class GuestDao {  
 private final SessionFactory sessionFactory;  
  
  
 public GuestDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
  
 }  
  
 public List<Guest> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Guest> res =session.createQuery("from Guest",Guest.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Guest guest){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 session.persist(guest);  
 transaction.commit();  
 }  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Guest guest = session.get(Guest.class,id);  
  
 ServiceDao serviceDao = new ServiceDao(sessionFactory);  
 serviceDao.delete(guest.getId());  
  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 PassportDao passportDao = new PassportDao(sessionFactory);  
  
 session.remove(session.get(Guest.class,id));  
 addressDao.delete(guest.getAddress().getId());  
 passportDao.delete(guest.getPassport().getId());  
  
 transaction.commit();  
 }  
 public void updateName(int id, String name){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 guestToBeUpdated.setName(name);  
  
 transaction.commit();  
 }  
 public void updateSurname(int id, String surname){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 guestToBeUpdated.setSurname(surname);  
  
 transaction.commit();  
 }  
 public void updatePatronymic(int id, String patronymic){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 guestToBeUpdated.setPatronymic(patronymic);  
  
 transaction.commit();  
 }  
 public void updateGender(int id, String gender){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 guestToBeUpdated.setGender(gender);  
  
 transaction.commit();  
 }  
 /\* private Date dateWork(String str){  
 DateFormat df = new SimpleDateFormat("dd-MM-yyyy");  
 Date date = null;  
 try {  
 date = df.parse(str);  
 return date;  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }\*/  
 public void updateBirth(int id, Date date){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
  
 guestToBeUpdated.setBirth\_date(date);  
  
 transaction.commit();  
 }  
 public void updateCountry(int id, String country){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateCountry(guestToBeUpdated.getAddress().getId(), country);  
  
 transaction.commit();  
 }  
 public void updateParking(int id, int nParking){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 guestToBeUpdated.setParking\_lot\_number(nParking);  
  
 transaction.commit();  
 }  
  
 public void updateCity(int id, String city){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateCity(guestToBeUpdated.getAddress().getId(), city);  
  
 transaction.commit();  
 }  
  
 public void updateStreet(int id, String street){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateStreet(guestToBeUpdated.getAddress().getId(), street);  
  
 transaction.commit();  
 }  
  
 public void updateBuilding(int id, String building){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateBuilding(guestToBeUpdated.getAddress().getId(), building);  
  
 transaction.commit();  
 }  
  
  
 public void updateNPassport(int id, int number){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 PassportDao passportDao = new PassportDao(sessionFactory);  
 passportDao.updateNPassport(guestToBeUpdated.getPassport().getId(), number);  
  
 transaction.commit();  
 }  
 public void updateIssuance(int id, Date date){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 PassportDao passportDao = new PassportDao(sessionFactory);  
 passportDao.updateIssuance(guestToBeUpdated.getPassport().getId(), date);  
  
 transaction.commit();  
 }  
  
 public void updateGivenBy(int id, String string){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 PassportDao passportDao = new PassportDao(sessionFactory);  
 passportDao.updateGivenBy(guestToBeUpdated.getPassport().getId(), string);  
  
 transaction.commit();  
 }  
  
 public void updateDEntry(int id, Date date) {  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class, id);  
 guestToBeUpdated.setDate\_of\_entry(date);  
  
 transaction.commit();  
 }  
  
 public void updateDDepartment(int id, Date date) {  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class, id);  
 guestToBeUpdated.setDeparture\_date(date);  
  
 transaction.commit();  
 }  
  
 public void updateNAuto(int id, String string){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 guestToBeUpdated.setAuto\_number(string);  
  
 transaction.commit();  
 }  
 public void updateNRoom(int id, int number){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 RoomDao roomDao = new RoomDao(this.sessionFactory);  
 Guest guestToBeUpdated = session.get(Guest.class,id);  
 Room room = roomDao.findRoomByNumber(number);  
 guestToBeUpdated.setRoom(room);  
  
 transaction.commit();  
 }  
  
  
 public List<Guest> findByFields(String fieldsToFind){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 DateToStringConverter dateToStringConverter = new DateToStringConverter();  
 String hql = "FROM Guest g JOIN g.address WHERE " +  
 " g.name LIKE '%" + fieldsToFind + "%' OR g.surname LIKE '%" + fieldsToFind + "%' " +  
 "OR g.patronymic LIKE '%" + fieldsToFind + "%' OR g.address.country LIKE '%" + fieldsToFind + "%'" +  
 "OR g.address.city LIKE '%" + fieldsToFind + "%' OR g.address.street LIKE '%" + fieldsToFind + "%'" +  
 "OR g.address.building LIKE '%" + fieldsToFind + "%' OR g.gender LIKE '%" + fieldsToFind + "%'" +  
 "OR g.auto\_number LIKE '%" + fieldsToFind + "%' OR cast(g.passport.number as string ) = '" + fieldsToFind + "' " +  
 "OR g.passport.givenBy LIKE '%" + fieldsToFind + "%' OR to\_char(g.passport.issuance,'DD-MM-YYYY') LIKE '%" + fieldsToFind + "%' " +  
 "OR cast(g.parking\_lot\_number as string) = '" + fieldsToFind + "' OR to\_char(g.date\_of\_entry,'DD-MM-YYYY') LIKE '%" + fieldsToFind + "%' " +  
 "OR to\_char(g.departure\_date,'DD-MM-YYYY') LIKE '%" + fieldsToFind + "%' OR to\_char(g.birth\_date,'DD-MM-YYYY') LIKE '%" + fieldsToFind + "%' " +  
 "OR cast(g.room.number as string) = '" + fieldsToFind + "' ";  
 Query query = session.createQuery(hql, Guest.class);  
 List<Guest> results = query.getResultList();  
  
  
 transaction.commit();  
 return results;  
 }  
  
  
 /\*public List<Guest> findByFields(String fieldsToFind) {  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 DateToStringConverter dateToStringConverter = new DateToStringConverter();  
  
 String hql = "FROM Guest g JOIN g.address WHERE " +  
 " g.name LIKE :searchTerm OR g.surname LIKE :searchTerm " +  
 " OR g.patronymic LIKE :searchTerm OR g.address.country LIKE :searchTerm " +  
 " OR g.address.city LIKE :searchTerm OR g.address.street LIKE :searchTerm " +  
 " OR g.address.building LIKE :searchTerm OR g.gender LIKE :searchTerm " +  
 " OR g.auto\_number LIKE :searchTerm OR g.passport.number = :numericSearchTerm " +  
 " OR g.passport.givenBy LIKE :searchTerm OR g.passport.issuance = :dateSearchTerm";  
  
 Query query = session.createQuery(hql, Guest.class);  
 query.setParameter("searchTerm", "%" + fieldsToFind + "%");  
  
 try {  
 int numericSearchTerm = Integer.parseInt(fieldsToFind);  
 query.setParameter("numericSearchTerm", numericSearchTerm);  
 } catch (NumberFormatException e) {  
 // Ignore if the conversion fails  
 }  
  
 try {  
 Date dateSearchTerm = dateToStringConverter.fromString(fieldsToFind);  
 query.setParameter("dateSearchTerm", dateSearchTerm);  
 } catch (Exception e) {  
 // Ignore if the conversion fails  
 }  
  
 List<Guest> results = query.getResultList();  
  
 transaction.commit();  
 return results;  
 }\*/  
  
}

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.models.Address;  
import com.example.hotelappwithhibernate.models.Maid;  
import com.example.hotelappwithhibernate.models.Room;  
import com.example.hotelappwithhibernate.models.Schedule;  
import jakarta.persistence.Query;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
  
import java.util.List;  
  
public class MaidDao {  
 private final SessionFactory sessionFactory;  
  
  
 public MaidDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
  
 }  
  
 public List<Maid> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Maid> res =session.createQuery("from Maid",Maid.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Maid maid){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 session.persist(maid);  
  
  
 session.persist(maid);  
 transaction.commit();  
 }  
  
 public void updateName(int id, String name){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
 maidToBeUpdated.setName(name);  
  
 transaction.commit();  
 }  
  
 public void updateSurname(int id, String surname){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
 maidToBeUpdated.setSurname(surname);  
  
 transaction.commit();  
 }  
  
 public void updatePatronymic(int id, String patronymic){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
 maidToBeUpdated.setPatronymic(patronymic);  
  
 transaction.commit();  
 }  
  
 public void updateCountry(int id, String country){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateCountry(maidToBeUpdated.getAddress().getId(),country);  
  
 transaction.commit();  
 }  
  
 public void updateCity(int id, String city){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateCity(maidToBeUpdated.getAddress().getId(),city);  
  
 transaction.commit();  
 }  
  
 public void updateStreet(int id, String street){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateStreet(maidToBeUpdated.getAddress().getId(),street);  
  
  
 transaction.commit();  
 }  
  
 public void updateBuilding(int id, String building){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Maid maidToBeUpdated = session.get(Maid.class,id);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 addressDao.updateBuilding(maidToBeUpdated.getAddress().getId(),building);  
  
 transaction.commit();  
 }  
  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Maid maid = session.get(Maid.class,id);  
 ScheduleDao scheduleDao = new ScheduleDao(sessionFactory);  
 AddressDao addressDao = new AddressDao(sessionFactory);  
 List<Schedule> schedules = maid.getSchedules();/\*  
 for(Schedule schedule : schedules){  
 session.remove(schedule);  
 }\*/  
 maid.getSchedules().forEach(session::remove);  
 session.remove(session.get(Maid.class,id));  
 addressDao.delete(maid.getAddress().getId());  
 transaction.commit();  
 }  
  
 public List<Maid> findByFields(String fieldsToFind){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 String hql = "FROM Maid m JOIN m.address WHERE " +  
 " m.name LIKE '%" + fieldsToFind + "%' OR m.surname LIKE '%" + fieldsToFind + "%' " +  
 "OR m.patronymic LIKE '%" + fieldsToFind + "%' OR m.address.country LIKE '%" + fieldsToFind + "%'" +  
 "OR m.address.city LIKE '%" + fieldsToFind + "%' OR m.address.street LIKE '%" + fieldsToFind + "%'" +  
 "OR m.address.building LIKE '%" + fieldsToFind + "%' ";  
 Query query = session.createQuery(hql, Maid.class);  
 List<Maid> results = query.getResultList();  
  
  
 transaction.commit();  
 return results;  
 }  
  
 public Maid findMaidById(int id){  
 Session session = sessionFactory.getCurrentSession();  
 Maid maid = session.get(Maid.class,id);  
 return maid;  
 }  
}

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.models.Address;  
import com.example.hotelappwithhibernate.models.Passport;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.List;  
  
public class PassportDao {  
 private final SessionFactory sessionFactory;  
  
 public PassportDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
 }  
  
 public List<Passport> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Passport> res =session.createQuery("from Passport",Passport.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Passport passport){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 session.persist(passport);  
  
 transaction.commit();  
 }  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
  
 session.remove(session.get(Passport.class,id));  
 }  
  
 public void updateNPassport(int id, int nPassport){  
 Session session = sessionFactory.getCurrentSession();  
  
 Passport passport = session.get(Passport.class,id);  
 passport.setNumber(nPassport);  
 }  
 public void updateIssuance(int id, Date date){  
 /\* DateFormat df = new SimpleDateFormat("dd-MM-yyyy");  
 Date date = null;  
 try {  
 date = df.parse(issuance);  
  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }\*/  
 Session session = sessionFactory.getCurrentSession();  
  
 Passport passport = session.get(Passport.class,id);  
 passport.setIssuance(date);  
 }  
  
 public void updateGivenBy(int id, String given){  
 Session session = sessionFactory.getCurrentSession();  
  
 Passport passport = session.get(Passport.class,id);  
 passport.setGivenBy(given);  
 }  
}

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.models.Room;  
  
  
import jakarta.persistence.Query;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
  
  
import java.util.List;  
import java.util.Optional;  
  
  
public class RoomDao {  
  
 private final SessionFactory sessionFactory;  
  
 public RoomDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
 }  
  
 public List<Room> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Room> res =session.createQuery("from Room",Room.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Room room){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 session.persist(room);  
 transaction.commit();  
 }  
  
 public void update(int id, int count){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Room roomToBeUpdated = session.get(Room.class,id);  
 roomToBeUpdated.setPeople\_count(count);  
  
 transaction.commit();  
 }  
  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 session.remove(session.get(Room.class,id));  
 transaction.commit();  
 }  
  
 public List<Room> findByFields(int fieldsToFind){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 String hql = "FROM Room r WHERE r.number = "+fieldsToFind+" OR r.people\_count = "+fieldsToFind+" ";  
 Query query = session.createQuery(hql, Room.class);  
 List<Room> results = query.getResultList();  
  
  
  
 transaction.commit();  
 return results;  
 }  
 public Room findRoomByNumber(int number){  
 Session session = sessionFactory.getCurrentSession();  
 String hql = "FROM Room r WHERE r.number = "+number+" ";  
 Query query = session.createQuery(hql, Room.class);  
 Room result = (Room)query.getSingleResult();  
 return result;  
 }  
  
}

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.DateToStringConverter;  
import com.example.hotelappwithhibernate.models.Guest;  
import com.example.hotelappwithhibernate.models.Maid;  
import com.example.hotelappwithhibernate.models.Room;  
import com.example.hotelappwithhibernate.models.Schedule;  
import jakarta.persistence.Query;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
  
import java.util.List;  
  
public class ScheduleDao {  
 private final SessionFactory sessionFactory;  
  
  
 public ScheduleDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
  
 }  
  
 public List<Schedule> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Schedule> res =session.createQuery("from Schedule", Schedule.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Schedule schedule, Room room, Maid maid){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 schedule.setMaid(maid);  
 //maid.setSchedules();  
 schedule.setRoom(room);  
 session.persist(schedule);  
 transaction.commit();  
 }  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 session.remove(session.get(Schedule.class,id));  
  
 transaction.commit();  
 }  
  
 public void updateDay(int id, String day){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Schedule schedule = session.get(Schedule.class,id);  
 schedule.setDay(day);  
  
 transaction.commit();  
 }  
  
 public void updateTime(int id, String time){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Schedule schedule = session.get(Schedule.class,id);  
 schedule.setTime(time);  
  
 transaction.commit();  
 }  
  
 public void updateNRoom(int id, int number){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Schedule scheduleToUpdate = session.get(Schedule.class,id);  
  
 RoomDao roomDao = new RoomDao(this.sessionFactory);  
 Room room = roomDao.findRoomByNumber(number);  
  
 scheduleToUpdate.setRoom(room);  
  
 transaction.commit();  
 }  
  
 public void updateMaid(int id, Maid maid){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 // MaidDao maidDao = new MaidDao(this.sessionFactory);  
 Schedule scheduleToUpdate = session.get(Schedule.class,id);  
 scheduleToUpdate.setMaid(maid);  
 //scheduleToUpdate.getMaid().setName(name);  
 /\*int maidsId = scheduleToUpdate.getMaid().getId();  
 Maid maid = maidDao.findMaidById(maidsId);  
 scheduleToUpdate.setMaid(maid);\*/  
  
 transaction.commit();  
 }  
  
 public List<Schedule> findByFields(String fieldsToFind){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 String hql = "FROM Schedule s WHERE " +  
 " s.day LIKE '%" + fieldsToFind + "%' OR s.time LIKE '%" + fieldsToFind + "%' " +  
 "OR s.maid.name LIKE '%" + fieldsToFind + "%' OR s.maid.surname LIKE '%" + fieldsToFind + "%'" +  
 "OR cast(s.room.number as string) = '" + fieldsToFind + "' ";  
 Query query = session.createQuery(hql, Schedule.class);  
 List<Schedule> results = query.getResultList();  
  
  
 transaction.commit();  
 return results;  
 }  
}

package com.example.hotelappwithhibernate.dao;  
  
import com.example.hotelappwithhibernate.models.Guest;  
import com.example.hotelappwithhibernate.models.Maid;  
import com.example.hotelappwithhibernate.models.Schedule;  
import com.example.hotelappwithhibernate.models.Service;  
import jakarta.persistence.Query;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class ServiceDao {  
 private final SessionFactory sessionFactory;  
  
  
 public ServiceDao(SessionFactory sessionFactory) {  
 this.sessionFactory = sessionFactory;  
  
 }  
  
 public List<Service> index(){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 List<Service> res =session.createQuery("from Service", Service.class).getResultList();  
 transaction.commit();  
 return res;  
 }  
  
 public void save(Service service, Guest guest){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 service.setGuests(new ArrayList<>(List.*of*(guest)));  
  
 guest.setServices(new ArrayList<>(List.*of*(service)));  
  
 session.persist(service);  
 transaction.commit();  
 }  
  
 public void delete(int id,Guest guest){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Service serviceToRemove = session.get(Service.class,id);  
 serviceToRemove.getGuests().remove(guest);  
  
 guest.getServices().remove(serviceToRemove);  
  
 session.remove(serviceToRemove);  
 transaction.commit();  
 }  
  
 public void delete(int id){  
 Session session = sessionFactory.getCurrentSession();  
 Guest guest = session.get(Guest.class,id);  
 int size = guest.getServices().size();  
 for(int i = 0; i<size;i++){  
 session.remove(guest.getServices().get(i));  
 }  
 }  
  
 public void updateGuest(int id, Guest guest){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 Service serviceToUpdate = session.get(Service.class,id);  
 serviceToUpdate.setGuests(new ArrayList<>(List.*of*(guest)));  
 session.merge(serviceToUpdate);  
 transaction.commit();  
 }  
  
 public void updateName(int id, String name){  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
  
 Service service = session.get(Service.class,id);  
 service.setName(name);  
  
 transaction.commit();  
 }  
  
 public List<Service> findByFields(String fieldsToFind){  
  
 Session session = sessionFactory.getCurrentSession();  
 Transaction transaction = session.beginTransaction();  
 String hql = "FROM Service s join s.guests g WHERE " +  
 " s.name LIKE '%" + fieldsToFind + "%' OR g.name LIKE '%" + fieldsToFind + "%' ";  
 System.*out*.println(hql);  
 Query query = session.createQuery(hql, Service.class);  
 List<Service> results = query.getResultList();  
  
 transaction.commit();  
 return results;  
 }  
}

1. models

package com.example.hotelappwithhibernate.models;  
  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "address1")  
public class Address {  
 @Id  
 @Column(name = "id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name = "country")  
 private String country;  
 @Column(name = "city")  
 private String city;  
 @Column(name = "street")  
 private String street;  
 @Column(name = "building")  
 private String building;  
  
  
 @OneToOne(mappedBy = "address",fetch = FetchType.*EAGER*)  
 private Maid maid;  
 @OneToOne(mappedBy = "address",fetch = FetchType.*EAGER*)  
 private Guest guest;  
  
  
 public Address() {  
 }  
  
  
 public Address(String country, String city, String street, String building) {  
 this.country = country;  
 this.city = city;  
 this.street = street;  
 this.building = building;  
  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getCountry() {  
 return country;  
 }  
  
 public void setCountry(String country) {  
 this.country = country;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 public String getStreet() {  
 return street;  
 }  
  
 public void setStreet(String street) {  
 this.street = street;  
 }  
  
 public String getBuilding() {  
 return building;  
 }  
  
 public void setBuilding(String building) {  
 this.building = building;  
 }  
  
 public Maid getMaid() {  
 return maid;  
 }  
  
 public void setMaid(Maid maid) {  
 this.maid = maid;  
 }  
  
 /\*public Guest getGuest() {  
 return guest;  
 }  
  
 public void setGuest(Guest guest) {  
 this.guest = guest;  
 }\*/  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "id=" + id +  
 ", country='" + country + '\'' +  
 ", city='" + city + '\'' +  
 ", street='" + street + '\'' +  
 ", building=" + building +  
 ", maid=" + maid +  
 '}';  
 }  
  
 public void setGuest(Guest guest) {  
 }  
  
 public Guest getGuest() {  
 return guest;  
 }  
}

package com.example.hotelappwithhibernate.models;  
  
import jakarta.persistence.\*;  
import org.hibernate.annotations.Type;  
  
import java.time.LocalDate;  
import java.util.Date;  
import java.util.List;  
import java.util.Objects;  
  
@Entity  
@Table(name ="guest")  
public class Guest {  
 @Id  
 @Column(name ="id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name ="name")  
 private String name;  
 @Column(name = "surname")  
 private String surname;  
 @Column(name ="patronymic")  
 private String patronymic;  
 @Column(name = "gender")  
 private String gender;  
 @Column(name = "birth\_date")  
 private Date birth\_date;  
 @OneToOne  
 @JoinColumn(name ="address\_id",referencedColumnName = "id")  
 private Address address;  
 @OneToOne  
 @JoinColumn(name ="passport\_id",referencedColumnName = "id")  
 private Passport passport;  
 @ManyToOne  
 @JoinColumn(name ="room\_id",referencedColumnName = "id")  
 private Room room;  
 @Column(name = "parking\_lot\_number")  
 private int parking\_lot\_number;  
 @Column(name = "auto\_number")  
 private String auto\_number;  
 @Column(name = "date\_of\_entry")  
 private Date date\_of\_entry;  
 @Column(name = "departure\_date")  
 private Date departure\_date;  
 @ManyToMany(mappedBy="guests",fetch = FetchType.*EAGER*)  
 private List<Service> services;  
  
  
 @Override  
 public String toString() {  
 return  
 name +  
 " " + passport;  
 }  
  
 public Guest(String name, String surname, String patronymic, String gender, Date birth\_date,  
 int parking\_lot\_number,  
 String auto\_number, Date date\_of\_entry, Date departure\_date) {  
 this.name = name;  
 this.surname = surname;  
 this.patronymic = patronymic;  
 this.gender = gender;  
 this.birth\_date = birth\_date;  
  
 this.parking\_lot\_number = parking\_lot\_number;  
 this.auto\_number = auto\_number;  
 this.date\_of\_entry = date\_of\_entry;  
 this.departure\_date = departure\_date;  
 }  
  
 public Guest() {  
 }  
  
 public List<Service> getServices() {  
 return services;  
 }  
  
 public void setServices(List<Service> services) {  
 this.services = services;  
 }  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public String getGender() {  
 return gender;  
 }  
  
 public void setGender(String gender) {  
 this.gender = gender;  
 }  
  
 public Date getBirth\_date() {  
 return birth\_date;  
 }  
  
 public void setBirth\_date(Date birth\_date) {  
 this.birth\_date = birth\_date;  
 }  
  
 public Address getAddress() {  
 return address;  
 }  
  
 public void setAddress(Address address) {  
 this.address = address;  
 }  
  
 public Passport getPassport() {  
 return passport;  
 }  
  
 public void setPassport(Passport passport) {  
 this.passport = passport;  
 }  
  
 public Room getRoom() {  
 return room;  
 }  
  
 public void setRoom(Room room) {  
 this.room = room;  
 }  
  
 public int getParking\_lot\_number() {  
 return parking\_lot\_number;  
 }  
  
 public void setParking\_lot\_number(int parking\_lot\_number) {  
 this.parking\_lot\_number = parking\_lot\_number;  
 }  
  
 public String getAuto\_number() {  
 return auto\_number;  
 }  
  
 public void setAuto\_number(String auto\_number) {  
 this.auto\_number = auto\_number;  
 }  
  
 public Date getDate\_of\_entry() {  
 return date\_of\_entry;  
 }  
  
 public void setDate\_of\_entry(Date date\_of\_entry) {  
 this.date\_of\_entry = date\_of\_entry;  
 }  
  
 public Date getDeparture\_date() {  
 return departure\_date;  
 }  
  
 public void setDeparture\_date(Date departure\_date) {  
 this.departure\_date = departure\_date;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Guest guest = (Guest) o;  
 return id == guest.id && parking\_lot\_number == guest.parking\_lot\_number && Objects.*equals*(name, guest.name) && Objects.*equals*(surname, guest.surname) && Objects.*equals*(patronymic, guest.patronymic) && Objects.*equals*(gender, guest.gender) && Objects.*equals*(birth\_date, guest.birth\_date) && Objects.*equals*(address, guest.address) && Objects.*equals*(passport, guest.passport) && Objects.*equals*(room, guest.room) && Objects.*equals*(auto\_number, guest.auto\_number) && Objects.*equals*(date\_of\_entry, guest.date\_of\_entry) && Objects.*equals*(departure\_date, guest.departure\_date);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, name, surname, patronymic, gender, birth\_date, address, passport, room, parking\_lot\_number, auto\_number, date\_of\_entry, departure\_date);  
 }  
}

package com.example.hotelappwithhibernate.models;  
  
  
import jakarta.persistence.\*;  
  
import java.util.List;  
  
@Entity  
@Table(name = "maid")  
public class Maid {  
 @Id  
 @Column(name ="id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name ="name")  
 private String name;  
 @Column(name = "surname")  
 private String surname;  
 @Column(name = "patronymic")  
 private String patronymic;  
 @OneToOne(fetch = FetchType.*EAGER*)  
 @JoinColumn(name = "address\_id", referencedColumnName = "id")  
 private Address address;  
  
  
  
 @OneToMany(mappedBy = "maid")  
 private List<Schedule> schedules;  
  
 @Override  
 public String toString() {  
 return "'" + name + '\'' +  
 " '" + surname + '\'';  
 }  
  
 public Maid(String name, String surname, String patronymic) {  
 this.name = name;  
 this.surname = surname;  
 this.patronymic = patronymic;  
  
 }  
  
  
 public Maid() {  
 }  
 public List<Schedule> getSchedules() {  
 return schedules;  
 }  
  
 public void setSchedules(List<Schedule> schedules) {  
 this.schedules = schedules;  
 }  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getPatronymic() {  
 return patronymic;  
 }  
  
 public void setPatronymic(String patronymic) {  
 this.patronymic = patronymic;  
 }  
  
 public Address getAddress() {  
 return address;  
 }  
  
 public void setAddress(Address address) {  
 this.address = address;  
 }  
}

package com.example.hotelappwithhibernate.models;  
  
import jakarta.persistence.\*;  
  
import java.util.Date;  
@Entity  
@Table(name ="passport")  
public class Passport {  
 @Id  
 @Column(name ="id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name ="number")  
 private int number;  
 @Column(name ="issuance")  
 private Date issuance;  
 @Column(name = "given\_by")  
 private String givenBy;  
 @OneToOne(mappedBy = "passport")  
 private Guest guest;  
  
 public Passport(int number, Date issuance, String givenBy) {  
 this.number = number;  
 this.issuance = issuance;  
 this.givenBy = givenBy;  
  
 }  
  
  
  
 public Passport() {  
 }  
 public Guest getGuest() {  
 return guest;  
 }  
  
 public void setGuest(Guest guest) {  
 this.guest = guest;  
 }  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getNumber() {  
 return number;  
 }  
  
 public void setNumber(int number) {  
 this.number = number;  
 }  
  
 public Date getIssuance() {  
 return issuance;  
 }  
  
 public void setIssuance(Date issuance) {  
 this.issuance = issuance;  
 }  
  
 public String getGivenBy() {  
 return givenBy;  
 }  
  
 @Override  
 public String toString() {  
 return  
 number+"";  
 }  
  
 public void setGivenBy(String givenBy) {  
 this.givenBy = givenBy;  
 }  
}

package com.example.hotelappwithhibernate.models;  
  
import jakarta.persistence.\*;  
  
import java.util.List;  
  
@Entity  
@Table(name = "room")  
public class Room {  
 @Id  
 @Column(name = "id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name = "number")  
 private int number;  
  
 @Column(name = "people\_count")  
 private int people\_count;  
  
 @OneToMany(mappedBy = "room")  
 private List<Guest> guests;  
 @OneToMany(mappedBy = "room")  
 private List<Schedule> schedules;  
  
 public Room(int number, int people\_count) {  
 this.number = number;  
 this.people\_count = people\_count;  
 }  
  
 public Room() {  
 }  
 public List<Guest> getGuests() {  
 return guests;  
 }  
  
 public void setGuests(List<Guest> guests) {  
 this.guests = guests;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getNumber() {  
 return number;  
 }  
  
 public void setNumber(int number) {  
 this.number = number;  
 }  
  
 public int getPeople\_count() {  
 return people\_count;  
 }  
  
 public void setPeople\_count(int people\_count) {  
 this.people\_count = people\_count;  
 }  
  
 @Override  
 public String toString() {  
 return "Room{" +  
 "id=" + id +  
 ", number=" + number +  
 ", people\_count=" + people\_count +  
 '}';  
 }  
}

package com.example.hotelappwithhibernate.models;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "schedule")  
public class Schedule {  
 @Id  
 @Column(name ="id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name ="day")  
 private String day;  
 @Column(name = "time")  
 private String time;  
 @ManyToOne  
 @JoinColumn(name = "maid\_id",referencedColumnName = "id")  
 private Maid maid;  
 @ManyToOne  
 @JoinColumn(name = "room\_id",referencedColumnName = "id")  
 private Room room;  
  
 public Schedule(String day, String time) {  
 this.day = day;  
 this.time = time;  
 }  
  
 public Schedule() {  
 }  
  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getDay() {  
 return day;  
 }  
  
 public void setDay(String day) {  
 this.day = day;  
 }  
  
 public String getTime() {  
 return time;  
 }  
  
 public void setTime(String time) {  
 this.time = time;  
 }  
  
 public Maid getMaid() {  
 return maid;  
 }  
  
 public void setMaid(Maid maid) {  
 this.maid = maid;  
 }  
  
 public Room getRoom() {  
 return room;  
 }  
  
 public void setRoom(Room room) {  
 this.room = room;  
 }  
}

package com.example.hotelappwithhibernate.models;  
  
import jakarta.persistence.\*;  
  
import java.util.List;  
import java.util.Objects;  
  
@Entity  
@Table(name="service")  
public class Service {  
 @Id  
 @Column(name ="id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
 @Column(name ="name")  
 private String name;  
 @ManyToMany(fetch = FetchType.*EAGER*)  
 @JoinTable(name = "guest\_service",joinColumns = @JoinColumn(name = "service\_id"),  
 inverseJoinColumns = @JoinColumn(name = "guest\_id"))  
 private List<Guest> guests;  
  
 public Service(String name) {  
 this.name = name;  
 }  
  
 public Service() {  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public List<Guest> getGuests() {  
 return guests;  
 }  
  
 public void setGuests(List<Guest> guests) {  
 this.guests = guests;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Service service = (Service) o;  
 return id == service.id && Objects.*equals*(name, service.name);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, name);  
 }  
}

package com.example.hotelappwithhibernate;  
  
import javafx.util.StringConverter;  
  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class DateToStringConverter extends StringConverter<Date> {  
  
 @Override  
 public String toString(Date d) {  
 DateFormat df = new SimpleDateFormat("dd-MM-yyyy");  
  
 return df.format(d);  
 }  
  
 @Override  
 public Date fromString(String s) {  
  
 DateFormat df = new SimpleDateFormat("dd-MM-yyyy");  
 Date date = null;  
 try {  
 date = df.parse(s);  
 return date;  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 return null;  
  
 }  
}

package com.example.hotelappwithhibernate;  
  
import com.example.hotelappwithhibernate.dao.AddressDao;  
import com.example.hotelappwithhibernate.dao.MaidDao;  
import com.example.hotelappwithhibernate.dao.RoomDao;  
import com.example.hotelappwithhibernate.models.Address;  
import com.example.hotelappwithhibernate.models.Maid;  
import com.example.hotelappwithhibernate.models.Room;  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import java.io.IOException;  
import java.util.List;  
  
public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("/com.example.hotelappwithhibernate/scenes/app.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 910, 510);  
 stage.setTitle("Ki-Really Hotel \*\*\*\*\*");  
 stage.setScene(scene);  
 stage.show();  
 stage.setResizable(false);  
  
 }  
  
 public static void main(String[] args) {  
 launch();  
 }  
}