Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ИРКУТСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационных технологий и анализа данных

ОТЧЕТ

к лабораторной работе №4 по дисциплине:

|  |
| --- |
| Инструментальные средства информационных систем |
| **Создание MDA-приложения** |

наименование темы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | ИСМб-17-1 |  |  |  | Русанов С.А. |
|  | шифр группы |  | подпись |  | Фамилия ИО |
| Проверил |  |  |  |  | Хритова М.А. |
|  | должность |  | подпись |  | Фамилия ИО |

**Содержание**

[1 Постановка задачи 3](#_Toc59231799)

[2 Описание предметной области 4](#_Toc59231800)

[3 Диаграмма классов 5](#_Toc59231801)

[4 Экранные формы MDA-приложения 6](#_Toc59231802)

[5 Листинг 8](#_Toc59231803)

# Постановка задачи

В данной лабораторной работе необходимо:

* Разработать диаграмму классов в нотации UML;
* Создать проект в Delphi и импортировать в него разработанную модель;
* Создать MDA-приложения с использованием Bold for Delphi.

# Описание предметной области

Имеется расписание (цена, длительность, описание), зрительский зал (название зала) и место (номер места, куплено).

# Диаграмма классов

Создаем диаграмму классов при помощи приложения IBM Rational Rose (рисунок 1).

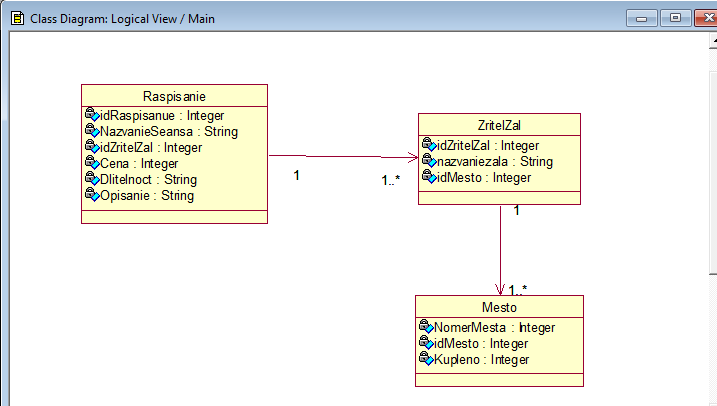


Рисунок 1 – Диаграмма классов

# Экранные формы MDA-приложения

Процесс создания MDA-приложения с использованием Bold for Delphi состоит из трех этапов:

1. Создание бизнес-уровня (рисунок 2);

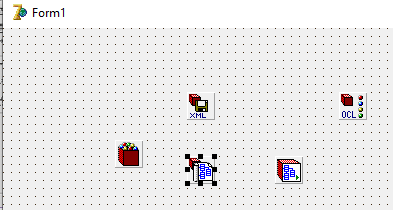


Рисунок 2 – Размещение компонентов BoldHandlers

1. Создание модели приложения (рисунок 2);

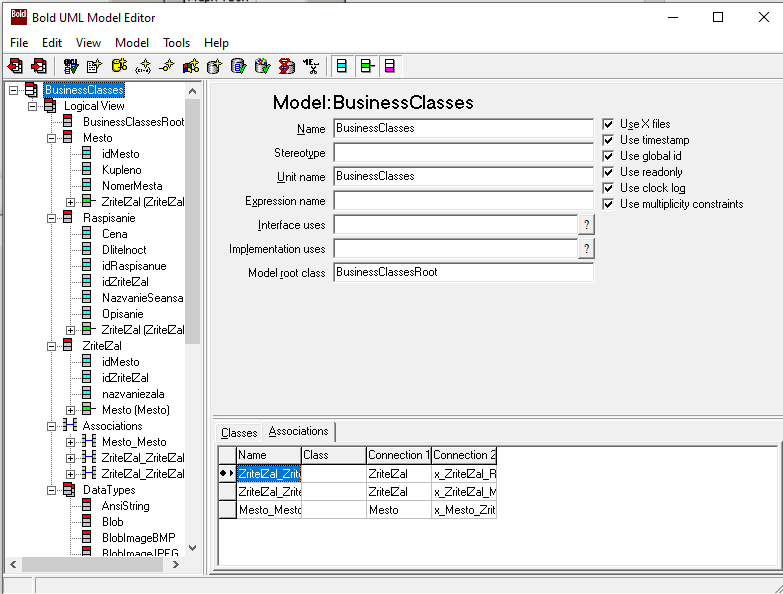


Рисунок 3 – Bold UML Model Editor

1. Создание уровня данных.

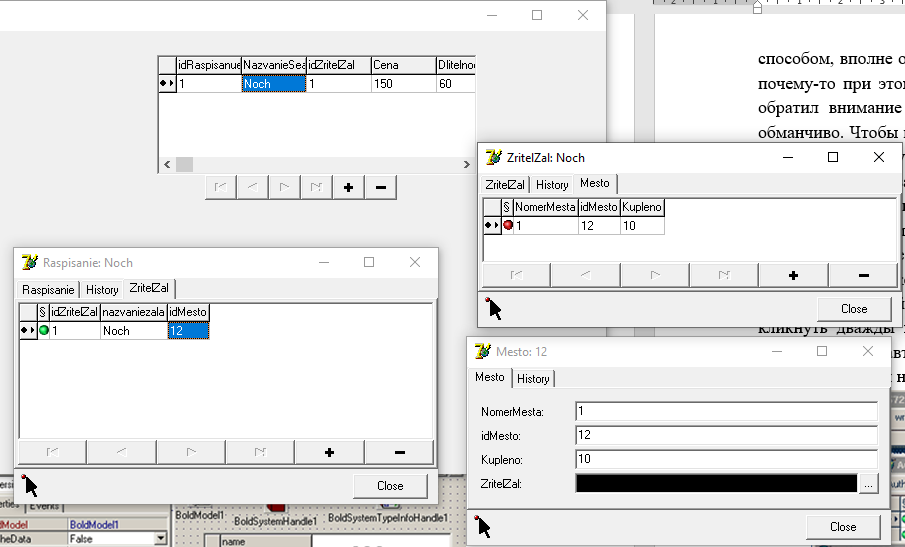


Рисунок 4 – Приложение

# Листинг

Листинг Unit1.pas

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, BoldHandles, BoldAbstractModel, BoldModel, BoldSubscription,

BoldSystemHandle, BoldRootedHandles, BoldAbstractListHandle,

BoldCursorHandle, BoldListHandle, Grids, BoldGrid, ExtCtrls,BoldAFPDefault,

BoldNavigatorDefs, BoldNavigator, BoldHandle, BoldPersistenceHandle,

BoldPersistenceHandleFile, BoldPersistenceHandleFileXML;

type

TForm1 = class(TForm)

BoldSystemHandle1: TBoldSystemHandle;

BoldModel1: TBoldModel;

BoldSystemTypeInfoHandle1: TBoldSystemTypeInfoHandle;

BoldListHandle1: TBoldListHandle;

BoldNavigator1: TBoldNavigator;

BoldGrid1: TBoldGrid;

BoldPersistenceHandleFileXML1: TBoldPersistenceHandleFileXML;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);

begin

BoldSystemHandle1.UpdateDatabase;

end;

end.

Листинг BusinessClasses.pas

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

(\* This file is autogenerated \*)

(\* Any manual changes will be LOST! \*)

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

(\* Generated 26.12.2020 21:02:17 \*)

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

(\* This file should be stored in the \*)

(\* same directory as the form/datamodule \*)

(\* with the corresponding model \*)

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

(\* Copyright notice: \*)

(\* \*)

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

unit code2;

{$DEFINE BusinessClasses\_unitheader}

{$INCLUDE BusinessClasses\_Interface.inc}

{ Includefile for methodimplementations }

const

BoldMemberAssertInvalidObjectType: string = 'Object of singlelink (%s.%s) is of wrong type (is %s, should be %s)';

{ TBusinessClassesRoot }

procedure TBusinessClassesRootList.Add(NewObject: TBusinessClassesRoot);

begin

if Assigned(NewObject) then

AddElement(NewObject);

end;

function TBusinessClassesRootList.IndexOf(anObject: TBusinessClassesRoot): Integer;

begin

result := IndexOfElement(anObject);

end;

function TBusinessClassesRootList.Includes(anObject: TBusinessClassesRoot) : Boolean;

begin

result := IncludesElement(anObject);

end;

function TBusinessClassesRootList.AddNew: TBusinessClassesRoot;

begin

result := TBusinessClassesRoot(InternalAddNew);

end;

procedure TBusinessClassesRootList.Insert(index: Integer; NewObject: TBusinessClassesRoot);

begin

if assigned(NewObject) then

InsertElement(index, NewObject);

end;

function TBusinessClassesRootList.GetBoldObject(index: Integer): TBusinessClassesRoot;

begin

result := TBusinessClassesRoot(GetElement(index));

end;

procedure TBusinessClassesRootList.SetBoldObject(index: Integer; NewObject: TBusinessClassesRoot);

begin;

SetElement(index, NewObject);

end;

{ TMesto }

function TMesto.\_Get\_M\_NomerMesta: TBAInteger;

begin

assert(ValidateMember('TMesto', 'NomerMesta', 0, TBAInteger));

Result := TBAInteger(BoldMembers[0]);

end;

function TMesto.\_GetNomerMesta: Integer;

begin

Result := M\_NomerMesta.AsInteger;

end;

procedure TMesto.\_SetNomerMesta(const NewValue: Integer);

begin

M\_NomerMesta.AsInteger := NewValue;

end;

function TMesto.\_Get\_M\_idMesto: TBAInteger;

begin

assert(ValidateMember('TMesto', 'idMesto', 1, TBAInteger));

Result := TBAInteger(BoldMembers[1]);

end;

function TMesto.\_GetidMesto: Integer;

begin

Result := M\_idMesto.AsInteger;

end;

procedure TMesto.\_SetidMesto(const NewValue: Integer);

begin

M\_idMesto.AsInteger := NewValue;

end;

function TMesto.\_Get\_M\_Kupleno: TBAInteger;

begin

assert(ValidateMember('TMesto', 'Kupleno', 2, TBAInteger));

Result := TBAInteger(BoldMembers[2]);

end;

function TMesto.\_GetKupleno: Integer;

begin

Result := M\_Kupleno.AsInteger;

end;

procedure TMesto.\_SetKupleno(const NewValue: Integer);

begin

M\_Kupleno.AsInteger := NewValue;

end;

function TMesto.\_Get\_M\_ZritelZal: TBoldObjectReference;

begin

assert(ValidateMember('TMesto', 'ZritelZal', 3, TBoldObjectReference));

Result := TBoldObjectReference(BoldMembers[3]);

end;

function TMesto.\_GetZritelZal: TZritelZal;

begin

assert(not assigned(M\_ZritelZal.BoldObject) or (M\_ZritelZal.BoldObject is TZritelZal), SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'ZritelZal', M\_ZritelZal.BoldObject.ClassName, 'TZritelZal']));

Result := TZritelZal(M\_ZritelZal.BoldObject);

end;

procedure TMesto.\_SetZritelZal(const value: TZritelZal);

begin

M\_ZritelZal.BoldObject := value;

end;

procedure TMestoList.Add(NewObject: TMesto);

begin

if Assigned(NewObject) then

AddElement(NewObject);

end;

function TMestoList.IndexOf(anObject: TMesto): Integer;

begin

result := IndexOfElement(anObject);

end;

function TMestoList.Includes(anObject: TMesto) : Boolean;

begin

result := IncludesElement(anObject);

end;

function TMestoList.AddNew: TMesto;

begin

result := TMesto(InternalAddNew);

end;

procedure TMestoList.Insert(index: Integer; NewObject: TMesto);

begin

if assigned(NewObject) then

InsertElement(index, NewObject);

end;

function TMestoList.GetBoldObject(index: Integer): TMesto;

begin

result := TMesto(GetElement(index));

end;

procedure TMestoList.SetBoldObject(index: Integer; NewObject: TMesto);

begin;

SetElement(index, NewObject);

end;

{ TRaspisanie }

function TRaspisanie.\_Get\_M\_idRaspisanue: TBAInteger;

begin

assert(ValidateMember('TRaspisanie', 'idRaspisanue', 0, TBAInteger));

Result := TBAInteger(BoldMembers[0]);

end;

function TRaspisanie.\_GetidRaspisanue: Integer;

begin

Result := M\_idRaspisanue.AsInteger;

end;

procedure TRaspisanie.\_SetidRaspisanue(const NewValue: Integer);

begin

M\_idRaspisanue.AsInteger := NewValue;

end;

function TRaspisanie.\_Get\_M\_NazvanieSeansa: TBAString;

begin

assert(ValidateMember('TRaspisanie', 'NazvanieSeansa', 1, TBAString));

Result := TBAString(BoldMembers[1]);

end;

function TRaspisanie.\_GetNazvanieSeansa: String;

begin

Result := M\_NazvanieSeansa.AsString;

end;

procedure TRaspisanie.\_SetNazvanieSeansa(const NewValue: String);

begin

M\_NazvanieSeansa.AsString := NewValue;

end;

function TRaspisanie.\_Get\_M\_idZritelZal: TBAInteger;

begin

assert(ValidateMember('TRaspisanie', 'idZritelZal', 2, TBAInteger));

Result := TBAInteger(BoldMembers[2]);

end;

function TRaspisanie.\_GetidZritelZal: Integer;

begin

Result := M\_idZritelZal.AsInteger;

end;

procedure TRaspisanie.\_SetidZritelZal(const NewValue: Integer);

begin

M\_idZritelZal.AsInteger := NewValue;

end;

function TRaspisanie.\_Get\_M\_Cena: TBAInteger;

begin

assert(ValidateMember('TRaspisanie', 'Cena', 3, TBAInteger));

Result := TBAInteger(BoldMembers[3]);

end;

function TRaspisanie.\_GetCena: Integer;

begin

Result := M\_Cena.AsInteger;

end;

procedure TRaspisanie.\_SetCena(const NewValue: Integer);

begin

M\_Cena.AsInteger := NewValue;

end;

function TRaspisanie.\_Get\_M\_Dlitelnoct: TBAString;

begin

assert(ValidateMember('TRaspisanie', 'Dlitelnoct', 4, TBAString));

Result := TBAString(BoldMembers[4]);

end;

function TRaspisanie.\_GetDlitelnoct: String;

begin

Result := M\_Dlitelnoct.AsString;

end;

procedure TRaspisanie.\_SetDlitelnoct(const NewValue: String);

begin

M\_Dlitelnoct.AsString := NewValue;

end;

function TRaspisanie.\_Get\_M\_Opisanie: TBAString;

begin

assert(ValidateMember('TRaspisanie', 'Opisanie', 5, TBAString));

Result := TBAString(BoldMembers[5]);

end;

function TRaspisanie.\_GetOpisanie: String;

begin

Result := M\_Opisanie.AsString;

end;

procedure TRaspisanie.\_SetOpisanie(const NewValue: String);

begin

M\_Opisanie.AsString := NewValue;

end;

function TRaspisanie.\_GetZritelZal: TZritelZalList;

begin

assert(ValidateMember('TRaspisanie', 'ZritelZal', 6, TZritelZalList));

Result := TZritelZalList(BoldMembers[6]);

end;

procedure TRaspisanieList.Add(NewObject: TRaspisanie);

begin

if Assigned(NewObject) then

AddElement(NewObject);

end;

function TRaspisanieList.IndexOf(anObject: TRaspisanie): Integer;

begin

result := IndexOfElement(anObject);

end;

function TRaspisanieList.Includes(anObject: TRaspisanie) : Boolean;

begin

result := IncludesElement(anObject);

end;

function TRaspisanieList.AddNew: TRaspisanie;

begin

result := TRaspisanie(InternalAddNew);

end;

procedure TRaspisanieList.Insert(index: Integer; NewObject: TRaspisanie);

begin

if assigned(NewObject) then

InsertElement(index, NewObject);

end;

function TRaspisanieList.GetBoldObject(index: Integer): TRaspisanie;

begin

result := TRaspisanie(GetElement(index));

end;

procedure TRaspisanieList.SetBoldObject(index: Integer; NewObject: TRaspisanie);

begin;

SetElement(index, NewObject);

end;

{ TZritelZal }

function TZritelZal.\_Get\_M\_idZritelZal: TBAInteger;

begin

assert(ValidateMember('TZritelZal', 'idZritelZal', 0, TBAInteger));

Result := TBAInteger(BoldMembers[0]);

end;

function TZritelZal.\_GetidZritelZal: Integer;

begin

Result := M\_idZritelZal.AsInteger;

end;

procedure TZritelZal.\_SetidZritelZal(const NewValue: Integer);

begin

M\_idZritelZal.AsInteger := NewValue;

end;

function TZritelZal.\_Get\_M\_nazvaniezala: TBAString;

begin

assert(ValidateMember('TZritelZal', 'nazvaniezala', 1, TBAString));

Result := TBAString(BoldMembers[1]);

end;

function TZritelZal.\_Getnazvaniezala: String;

begin

Result := M\_nazvaniezala.AsString;

end;

procedure TZritelZal.\_Setnazvaniezala(const NewValue: String);

begin

M\_nazvaniezala.AsString := NewValue;

end;

function TZritelZal.\_Get\_M\_idMesto: TBAInteger;

begin

assert(ValidateMember('TZritelZal', 'idMesto', 2, TBAInteger));

Result := TBAInteger(BoldMembers[2]);

end;

function TZritelZal.\_GetidMesto: Integer;

begin

Result := M\_idMesto.AsInteger;

end;

procedure TZritelZal.\_SetidMesto(const NewValue: Integer);

begin

M\_idMesto.AsInteger := NewValue;

end;

function TZritelZal.\_GetMesto: TMestoList;

begin

assert(ValidateMember('TZritelZal', 'Mesto', 5, TMestoList));

Result := TMestoList(BoldMembers[5]);

end;

procedure TZritelZalList.Add(NewObject: TZritelZal);

begin

if Assigned(NewObject) then

AddElement(NewObject);

end;

function TZritelZalList.IndexOf(anObject: TZritelZal): Integer;

begin

result := IndexOfElement(anObject);

end;

function TZritelZalList.Includes(anObject: TZritelZal) : Boolean;

begin

result := IncludesElement(anObject);

end;

function TZritelZalList.AddNew: TZritelZal;

begin

result := TZritelZal(InternalAddNew);

end;

procedure TZritelZalList.Insert(index: Integer; NewObject: TZritelZal);

begin

if assigned(NewObject) then

InsertElement(index, NewObject);

end;

function TZritelZalList.GetBoldObject(index: Integer): TZritelZal;

begin

result := TZritelZal(GetElement(index));

end;

procedure TZritelZalList.SetBoldObject(index: Integer; NewObject: TZritelZal);

begin;

SetElement(index, NewObject);

end;

function GeneratedCodeCRC: String;

begin

result := '1485239007';

end;

procedure InstallObjectListClasses(BoldObjectListClasses: TBoldGeneratedClassList);

begin

BoldObjectListClasses.AddObjectEntry('BusinessClassesRoot', TBusinessClassesRootList);

BoldObjectListClasses.AddObjectEntry('Mesto', TMestoList);

BoldObjectListClasses.AddObjectEntry('Raspisanie', TRaspisanieList);

BoldObjectListClasses.AddObjectEntry('ZritelZal', TZritelZalList);

end;

procedure InstallBusinessClasses(BoldObjectClasses: TBoldGeneratedClassList);

begin

BoldObjectClasses.AddObjectEntry('BusinessClassesRoot', TBusinessClassesRoot);

BoldObjectClasses.AddObjectEntry('Mesto', TMesto);

BoldObjectClasses.AddObjectEntry('Raspisanie', TRaspisanie);

BoldObjectClasses.AddObjectEntry('ZritelZal', TZritelZal);

end;

var

CodeDescriptor: TBoldGeneratedCodeDescriptor;

initialization

CodeDescriptor := GeneratedCodes.AddGeneratedCodeDescriptorWithFunc('BusinessClasses', InstallBusinessClasses, InstallObjectListClasses, GeneratedCodeCRC);

finalization

GeneratedCodes.Remove(CodeDescriptor);

end.