Skripta za prvi kolokvijum iz projektovanja softvera

Svrha skripte je pretstavljanje kostura klasa, interfejsa i njihove implementacije

# Domenski objekti – rs.ac.bg.fon.ps.domain

public class **DomenskiObjekat**{

private Long id;

//ostali atributi

//Konstruktor

//Geteri, seteri

//Override equals i to String metoda

}

# IRepository(najvisi nivo apstrakcije) – rs.ac.bg.fon.ps.repository

public interface **IRepository** <T, K> {

public void connect() throws Exception;

public void disconnect() throws Exception;

public void add(T t) throws Exception;

public void update(T t) throws Exception;

public void update(T t,K k) throws Exception;

public void delete(T t) throws Exception;

public List<T> getAll() throws Exception;

public T get(K k) throws Exception;

}

# Repository database – rs.ac.bg.fon.ps.repository.db

## Repository database-DbConnectionFactory

public class **DbConnectionFactory** {

private static DbConnectionFactory instance;

private Connection connection;

private DbConnectionFactory() {

}

public static DbConnectionFactory getInstance() {

if (instance == null) {

instance = new DbConnectionFactory();

}

return instance;

}

public Connection getConnection() throws SQLException {

if (connection == null || connection.isClosed()) {

String url = "jdbc:mysql://localhost:3306/softveri";

String user = "root";

String password = "";

connection = DriverManager.getConnection(url, user, password);

}

return connection;

}

}

## Repository database-IDbRepository

public interface **IDbRepository**<T, K> extends **IRepository**<T, K> {

@Override

public default void connect() throws Exception {

DbConnectionFactory.getInstance().getConnection();

}

@Override

public default void disconnect() throws Exception {

DbConnectionFactory.getInstance().getConnection().close();

}

}

# Repository database implementacija – rs.ac.bg.fon.ps.repository.db.impl

public class RepositoryDomenskiObjekat implements IDbRepository<Domenski objekat,Long>{

//Konstruktor

### GET ALL METODA

public List<DomenskiObjekat> **getAll**() throws Exception {

try {

String query = "SELECT id, name,......,**domenskiObjekat2ID**"

+ " FROM domenskiobjekat";

System.out.println(query);

List<DomesnkiObjekat> domenskiObjekti = new ArrayList<>();

Connection connection = DbConnectionFactory.getInstance().getConnection();

Statement statement = connection.createStatement();

ResultSet rs = statement.executeQuery(query);

while (rs.next()) {

**DomenskiObjekat2 do2=Controller.getInstance().getDomesnkiObjekat2(rs.getLong(“domenskiObjekat2ID”));**

DomenskiObjekat do = new DomenskiObjekat (rs.getLong("id"), rs.getString("name"),rs.get...,**do2**);

domenskiObjekti.add(do);

}

rs.close();

statement.close();

return domenskiObjekti;

} catch (Exception e) {

e.printStackTrace();

throw new Exception("Get all domesnkiObjekti error. \n" + e.getMessage());

}

}

**Ako domenski objekat ima atribut koji je tipa nekog drugog domenskog objekta, na primer Predmet ima atribut koji Profesor ga predaje koji je tipa Profesor. onda je potrebno ubaciti gore boldovan tekst. domenskiObjekat2ID je spoljni kljuc domenskog objekta 1i takodje je veza u sql bazi sa primarnim kljucem domenskog objekta 2.**

### GET METODA

public DomesnkiObjekat **get**(Long k) throws Exception {

try {

String query = "SELECT id,username,firstName,lastName,password,...."

+ " FROM domesnkiObjekat"+" WHERE id="+k;

System.out.println(query);

DomenskiObjekat do=null;

Connection connection = DbConnectionFactory.getInstance().getConnection();

Statement statement = connection.createStatement();

ResultSet rs = statement.**executeQuery**(query);

if (rs.next()) {

do = new DomenskiObjekat(rs.getLong("id"), rs.getString("username"),rs.getString("firstName"),rs.getString("lastName"),rs.getString("password"),......);

}

rs.close();

statement.close();

return do;

} catch (Exception e) {

e.printStackTrace();

throw new Exception("Get DomenskiObjekat error. \n" + e.getMessage());

}

}

### ADD METODA

public void **add**(DomenskiObjekat t) throws Exception {

try {

String query = "INSERT INTO domenskiObjekat (student\_id,subject\_id,grade,date) VALUES (?, ?, ?, ?)";

System.out.println(query);

Connection connection = DbConnectionFactory.getInstance().getConnection();

PreparedStatement statement = connection.prepareStatement(query);

statement.setString(1, t.getStudent().getIndex());

statement.setLong(2, t.getSubject().getId());

statement.setInt(3, t.getGrade());

statement.setDate(4, **new java.sql.Date(t.getDate().getTime())**);

statement.executeUpdate();

statement.close();

} catch (Exception e) {

e.printStackTrace();

throw new Exception("Insert Application error");

}

}

**KONSTRUKTOR new java.sql.Date(t.getDate().getTime())) prebacuje iz Jave atribut tipa Date u sqlDate tip koji moze da se ucita unutar sqla.**

**UMESTO ovakovog upita sa place holderima mogao se koristiti i** INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

**Posto je query string,voditi racuna o tome da Datum i neke varchar vrednosti u sql-u staviti pod navodnicima, npr:**

**String query = ”INSERT INTO domenskiObjekat (ime)”+” VALUES (‘t.getIme()’) “ ;**

**Ovo ne vazi samo kod add tj insert metode, vec kod svih upita**

**Ako dodajemo u sql tabelu vreme - new java.sql.Timestamp(ld.getD().getTime()));**

Ako se datum iz neke tekst komponente prebacuje u sql bazu, to se radi ovako:

**String date=txtDate.getText();**

**Date date2=new SimpleDateFormat(“dd/mm/yyyy”).parse(date);**

ds

### UPDATE METODA sa 2 parametra

public void **update**(DomesnkiObjekat t, Long id) throws Exception {

try{

String query = "UPDATE domenskiobjekat"+

" SET student\_id='"+String.valueOf(t.getStudent().getIndex())+"',subject\_id="+String.valueOf(t.getSubject().getId())

+",grade="+String.valueOf(t.getGrade()) +",date='"+new java.sql.Timestamp(t.getDate().getTime())+

"' WHERE id="+id;

System.out.println(query);

Connection connection = DbConnectionFactory.getInstance().getConnection();

Statement statement = connection.createStatement();

statement.executeUpdate(query);

statement.close();

}catch(Exception e){

e.printStackTrace();

throw new Exception("Update domesnkiobjekat error. \n" + e.getMessage());

}

}

**Upit se moze realizovati i koriscenjem placeHoldera;**

### DELETE METODA

public void delete(DomenskiObjekat t) throws Exception {

try{

String query = "DELETE FROM domesnkiobjekat"

+ " WHERE id=" + t.getId().toString();

System.out.println(query);

Connection connection = DbConnectionFactory.getInstance().getConnection();

Statement statement = connection.createStatement();

statement.**executeUpdate**(query);

statement.close();

}catch(Exception e){

e.printStackTrace();

throw new Exception("Delete domenskiObjekat error. \n" + e.getMessage());

}

}

Kod metoda koje ne vracaju nista, tj su tipa void, navodi se **executeUpdate umesto executeUpdate.**

}

# Repository memory – rs.ac.bg.fon.ps.repository.memory

## Repository memory-IMemoryRepository

@Override

public default void connect() throws Exception{

}

@Override

public default void disconnect() throws Exception{

}

//ovde nam ne treba konekcija sa bazom

# Repository memory impl – rs.ac.bg.fon.ps.repository.memory.impl

public class **RepositoryMemoryDomenskiObjekat** implements IMemoryRepository<DomenskiObjekat, Long> {

private List<DomenskiObjekat> domenskiObjekti;

public RepositoryMemoryDomenskiObjekat () {

domenskiObjekti = new ArrayList<>();

domenskiObjekti.add(new DomenskiObjekat (1l, "do1"));

domenskiObjekti.add(new DomenskiObjekat (2l, "do2"));

domenskiObjekti.add(new DomenskiObjekat (3l, "do3"));

}

public List< DomenskiObjekat > getAll() throws Exception{

return domenskiObjekti;

}

@Override

public void add(DomenskiObjekat do) throws Exception {

if (!domenskiObjekti.contains(do)) {

domenskiObjekti.add(do);

} else {

throw new Exception("DomenskiObjekat already exists.");

} }

@Override

public void save(Manufacturer t) throws Exception {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public void delete(Manufacturer t) throws Exception {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public Manufacturer get(Long k) throws Exception {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

# Controller – rs.ac.bg.fon.ps.controller

public class **Controller** {

private RepositoryMemoryPoreskaStopa repositoryMemoryPoreskaStopa;

private RepositoryMemoryProizvod repositoryMemoryProizvod;

private RepositoryMemoryProizvodjac repositoryMemoryProizvodjac;

private RepositoryDbProfessor repositoryDbProfesor;

private RepositoryDbPredmet repositoryDbPredmet;

private static Controller instance;

public Controller() {

repositoryMemoryPoreskaStopa=new RepositoryMemoryPoreskaStopa();

repositoryMemoryProizvod=new RepositoryMemoryProizvod();

repositoryMemoryProizvodjac=new RepositoryMemoryProizvodjac();

repositoryDbProfesor=new RepositoryDbProfessor();

repositoryDbPredmet=new RepositoryDbPredmet();

}

private Controller(){

}

public static Controller getInstance(){

if(instance==null){

instance=new Controller();

}

return instance;

}

public List<Proizvod> getAllProizvods() throws Exception{

List<Proizvod> proizvodi=new ArrayList<>();

proizvodi=repositoryMemoryProizvod.getAll();

return proizvodi;

}

public List<Proizvodjac> getAllProizvodjacs() throws Exception{

List<Proizvodjac> proizvodjaci=new ArrayList<>();

proizvodjaci=repositoryMemoryProizvodjac.getAll();

return proizvodjaci;

}

public List<PoreskaStopa> getAllPoreskaStopas() throws Exception{

List<PoreskaStopa> poreskeStope=new ArrayList<>();

poreskeStope=repositoryMemoryPoreskaStopa.getAll();

return poreskeStope;

}

public List<Profesor> getAllProfesors() throws Exception {

List<Profesor> profesori=null;

try{

repositoryDbProfesor.connect();

profesori=repositoryDbProfesor.getAll();

}catch(Exception e){

e.printStackTrace();

}finally{

repositoryDbProfesor.disconnect();

}

return profesori;

}

public Profesor getProfesor(Long id) throws Exception{

repositoryDbProfesor.connect();

Profesor p=null;

try{

p=(Profesor)repositoryDbProfesor.get(id);

}catch(Exception e){

e.printStackTrace();

throw e;

}finally{

//repositoryDbProfesor.disconnect();

}

return p;

}

public void updateProfesor(Profesor p) throws Exception {

try{

repositoryDbProfesor.connect();

repositoryDbProfesor.update(p, p.getId());

}catch(Exception e){

e.printStackTrace();

}finally{

repositoryDbProfesor.disconnect();

}

}

public List<Predmet> getAllPredmets() throws Exception {

List<Predmet> angazovanja=null;

try{

repositoryDbPredmet.connect();

angazovanja=repositoryDbPredmet.getAll();

}catch(Exception e){

e.printStackTrace();

}finally{

repositoryDbPredmet.disconnect();

}

return angazovanja;

}

**Disconnect zakomentarisi ako dobijes error rs.statement close..**

# ViewController – rs.ac.bg.fon.ps.view.controller

public class **ViewController** {

private FrmMain frmMain;

private FrmZadatak1 frmZadatak1;

private FrmZadatak2 frmZadatak2;

private FrmEditProfesor frmEditProfesor;

private FrmZadatak3 frmZadatak3;

private FrmAngazovanja frmAngazovanja;

private static ViewController instance;

private static List<Object> izabraniProfesori;

private Map<String, Object> paramMap = new HashMap<>();

public static List<Object> getIzabraniProfesori() {

return izabraniProfesori;

}

public static void setIzabraniProfesori(List<Object> izabraniProfesori) {

ViewController.izabraniProfesori = izabraniProfesori;

}

private ViewController() {

}

public static ViewController getInstance() {

if (instance == null) {

instance = new ViewController();

}

return instance;

}

public void startApp() {

frmMain = new FrmMain();

frmMain.setExtendedState(Frame.MAXIMIZED\_BOTH);

frmMain.setVisible(true);

frmMain.setLocationRelativeTo(null);

frmMain.setTitle("Main form");

}

public void startZadatak1() {

frmZadatak1 = new FrmZadatak1(frmMain, true);

frmZadatak1.setLocationRelativeTo(null);

frmZadatak1.setVisible(true);

frmZadatak1.setTitle("Zadatak 1");

}

public void setParamMap(Proizvod product) {

paramMap.put("PROFESOR\_DETAILS", profesor);

}

public void setParamMapProfesor(Profesor profesor) {

paramMap.put("PROFESOR\_DETAILS", profesor);

}

public Map<String, Object> getParamMap() {

return paramMap;

}

public void openZadatak2() {

frmZadatak2 = new FrmZadatak2(frmMain, true);

frmZadatak2.setLocationRelativeTo(null);

frmZadatak2.setVisible(true);

frmZadatak2.setTitle("Zadatak 2");

}

public void openEditProfesorForm() {

frmEditProfesor = new FrmEditProfesor(frmMain, true);

frmEditProfesor.setLocationRelativeTo(null);

frmEditProfesor.setVisible(true);

frmEditProfesor.setTitle("Edit professor form");

}

public void openZadatak3(){

frmZadatak3 = new FrmZadatak3(frmMain, true);

frmZadatak3.setLocationRelativeTo(null);

frmZadatak3.setVisible(true);

frmZadatak3.setTitle("Zadatak 3");

}

public void startFrmAngazovanja() {

frmAngazovanja=new FrmAngazovanja(frmMain, true);

frmAngazovanja.setLocationRelativeTo(null);

frmAngazovanja.setVisible(true);

frmAngazovanja.setTitle("Angazovanja");

}

} **private Map<String, Object> paramMap = new HashMap<>();**

private static ViewController instance;

private ViewController() {

}

public static ViewController getInstance() {

if (instance == null) {

instance = new ViewController();

}

return instance;

}

public void startApp() {

new FormLogin().setVisible(true);

}

public void openMain(User loggedUser) {

mainForm = new FormMain();

mainForm.setVisible(true);

this.loggedUser = loggedUser;

}

public void openViewProducts() {

formProducts = new FormProducts(mainForm, true);

formProducts.setLocationRelativeTo(mainForm);

formProducts.setVisible(true);

}

public void openViewProductsTM() {

formProductsTableModel = new FormProductsTableModel(mainForm, true);

formProductsTableModel.setLocationRelativeTo(mainForm);

formProductsTableModel.setVisible(true);

}

public void openProductForm(Product product){

paramMap.put("PRODUCT\_FORM\_DETAILS", product);

new FormProduct(mainForm, true, FormMode.FORM\_DETAILS).setVisible(true);

}

**public Map<String, Object> getParamMap() {**

**return paramMap;**

**}**

public void refreshProductsView() {

if(formProductsTableModel != null) formProductsTableModel.refreshProductData();

}

}

# UTILS – MOD OTVARANJA FORME rs.ac.bg.fon.ps.view.utils

public enum FormMode {

FORM\_ADD,

FORM\_DETAILS,

}

# CUSTOM MODEL TABELE - rs.ac.bg.fon.ps.view.componenets.table

public class ProductTableModel extends AbstractTableModel {

private final List<Product> products;

private String[] columnNames = new String[]{"ID", "NAME", "DESCRIPTION", "UNIT", "MANUFACTURER", "PRICE"};

private Class[] columnClasses = new Class[]{Long.class, String.class, String.class, MeasurmentUnit.class, Manufacturer.class, BigDecimal.class};

public ProductTableModel(List<Product> products) {

this.products = products;

}

@Override

public int getRowCount() {

if (products == null) {

return 0;

}

return products.size();

}

@Override

public int getColumnCount() {

return columnNames.length;

}

@Override

public Class<?> getColumnClass(int columnIndex) {

if (columnIndex > columnClasses.length) {

return null;

}

return columnClasses[columnIndex];

}

@Override

public Object getValueAt(int rowIndex, int columnIndex) {

Product product = products.get(rowIndex);

switch (columnIndex) {

case 0:

return product.getId();

case 1:

return product.getName();

case 2:

return product.getDescription();

case 3:

return product.getMeasurmentUnit();

case 4:

return product.getManufacturer();

case 5:

return product.getPrice();

default:

return "n/a";

}

}

@Override

public String getColumnName(int column) {

if (column > columnNames.length) {

return "n/a";

}

return columnNames[column];

}

@Override

public boolean isCellEditable(int rowIndex, int columnIndex) {

return true;

}

@Override

public void setValueAt(Object aValue, int rowIndex, int columnIndex) {

Product product = products.get(rowIndex);

switch (columnIndex) {

case 0:

product.setId((Long) aValue);

break;

case 1:

product.setName(aValue.toString());

break;

case 2:

product.setDescription(aValue.toString());

break;

case 3:

product.setMeasurmentUnit((MeasurmentUnit) aValue);

break;

case 4:

product.setManufacturer((Manufacturer) aValue);

break;

case 5:

product.setPrice(new BigDecimal(aValue.toString()));

break;

}

}

public void insertProduct(Product p) {

this.products.add(p);

// fireTableDataChanged();

fireTableRowsInserted(products.size() - 1, products.size() - 1);

}

public Product deleteProduct(int row) {

Product p = products.remove(row);

fireTableRowsDeleted(row, row);

return p;

}

public Product getProductAt(int index) {

return products.get(index);

}

public void refreshProducts() {

fireTableDataChanged();

}

}

# MAIN-rs.ac.bg.fon.ps.main

public class Main {

public static void main(String[] args) {

ViewController.getInstance().startApp();

}

}

# FORME-rs.ac.bg.fon.ps.view.forms

//kod svake forme ispod init components ima prepare view fja

Punjenje comboBoxVrednosti sa domenskim objektima

private void fillCbDomenskiObjekat() throws Exception {

cbDomenskiObjekat.removeAllItems();

List<DomenskiObjekat> dos = Controller.getInstance().getAllDomesnkiObjekti();

for (DomenskiObjekat do : dos) {

cbDomenskiObjekat.addItem(do);

}

}

Punjenje comboBoxVrednosti sa enumom

Ako imamo enum measurment unit

private void fillCbMeasurmentUnit() {

cbMeasurmentUnit.removeAllItems();

for (MeasurmentUnit mu : MeasurmentUnit.values()) {

cbMeasurmentUnit.addItem(mu);

}

}

Punjenje tabele sa customm modelom i podacima

private void fillTableDomenskiObjekti() throws Exception {

List<DomenskiObjekat> dos = Controller.getInstance().getAllDomesnkiObjekti();

DomenskiObjekatTableModel model = new DomenskiObjekatTableModel (dos);

tblDo.setModel(model);

//punjenje polja cb sa domesnkim objektima manufacturers

List<Manufacturer> manufacturers = Controller.getInstance().getAllManufacturers();

JComboBox cbManufacturers = new JComboBox(manufacturers.toArray());

TableColumn tc = tblProducts.getColumnModel().getColumn(4);

tc.setCellEditor(new DefaultCellEditor(cbManufacturers));

//punjenje polja cb sa enumom

JComboBox cbMeasurmentUnit = new JComboBox(MeasurmentUnit.values());

TableColumn tc2 = tblProducts.getColumnModel().getColumn(3);

tc2.setCellEditor(new DefaultCellEditor(cbMeasurmentUnit));

}

**//Dugmici za Add new Row, Add,Edit,Delete**

private void **btnAddEmptyRowActionPerformed**(java.awt.event.ActionEvent evt) {

ApplicationTableModel model = (ApplicationTableModel) tblApplications.getModel();

model.insertApplication(new Appllication()); //insertuje prazan red

}

private void **btnSubmitActionPerformed**(java.awt.event.ActionEvent evt) {

ApplicationTableModel model = (ApplicationTableModel) tblApplications.getModel();

int selectedRow = tblApplications.getSelectedRow();

Appllication app = model.getApplicationAt(selectedRow);

if (app.getId() != null && app.getGrade() != 0 && app.getStudent() != null && app.getSubject() != null && app.getDate() != null) {

try {

Controller.getInstance().saveApplication(app, app.getStudent(), app.getSubject());

JOptionPane.showMessageDialog(this, "Application added sucesfully", "Application", JOptionPane.INFORMATION\_MESSAGE);

} catch (Exception ex) {

ex.printStackTrace();

}

} else {

JOptionPane.showMessageDialog(this, "Please fill all fields before adding applicaiton", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void **btnDeleteActionPerformed**(java.awt.event.ActionEvent evt) {

ApplicationTableModel model = (ApplicationTableModel) tblApplications.getModel();

int selectedRow = tblApplications.getSelectedRow();

Appllication app = model.getApplicationAt(selectedRow);

try {

Controller.getInstance().deleteApplication(app);

model.deleteApplicaiton(selectedRow);

} catch (Exception ex) {

JOptionPane.showMessageDialog(this, "Error deleting application", "Delete Error", JOptionPane.ERROR\_MESSAGE);

ex.printStackTrace();

}

}

private void **btnEditActionPerformed**(java.awt.event.ActionEvent evt) {

int selectedRow = tblApplications.getSelectedRow();

if (selectedRow >= 0) {

ApplicationTableModel model = (ApplicationTableModel) tblApplications.getModel();

Appllication app = model.getApplicationAt(selectedRow);

ViewController.getInstance().openFrmApplication(app);

} else {

JOptionPane.showMessageDialog(this, "You must select application", "Edit product error", JOptionPane.ERROR\_MESSAGE);

}

}

**//Fillovanje liste**

private void fillListProfssors() throws Exception {

listProfesori.removeAll();

List<Profesor> profesor = Controller.getInstance().getAllProfesors();

DefaultListModel listaProfesora = new DefaultListModel();

listProfesori.setModel(listaProfesora);

for (Profesor profesor1 : profesor) {

listaProfesora.addElement(profesor1);

}

}