МИНИСТЕРСТВО образования РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

УНИВЕРСИТЕТ информатики и радиоэлектроники»

Факультет компьютерного проектирования

Индивидуальная практическая работа №2

По дисциплине: «Реализация EJB- приложения c использованием session компонентов»

Тема: «Учет выплат по кредитам»

Выполнил студент 4-го курса факультета инновационного непрерывного образования Группа [894351](https://iis.bsuir.by/#/schedule;groupName=493551)

Галкин Илья Викторович

Зачетная книжка №6535021

Дата сдачи: 13.01.2022

Проверил: Унучек Е.Н.

 Минск 2022

**Оглавление**

[1. Цель 3](#_Toc91872395)

[2. Выполнение работы 3](#_Toc91872396)

# Краткие теоретические сведения

**EJB**(Enterprise Java Beans) – это фреймворк для построение бизнес-логики приложения.

Сервер приложений J2EE состоит из двух основных элементов:

WEB-Container– (JSP, JSF и т.д.) все что дает конечный вид пользователю, а точней пользовательский интерфейс.

**EJB-Container –** используется для написания бизнес-логики.

С точки зрения EJB – это технология, предоставляющая множество готовых решений (управление транзакциями, безопасность, хранение информации и т.п.) для вашего приложения.

### EJB делится на три типа компонентов

**1. Session beans**– используется для построения бизнес-логики, которая может быть вызвана программным клиентом через локальный, удаленный или веб-интерфейс обслуживания клиентов.

Для доступа к приложению, развернутого на сервере, клиент вызывает методы сессионного компонента. Сессионный компонент выполняет работу для своего клиента, защищая его от сложности, выполняя бизнес-задач внутри сервера.

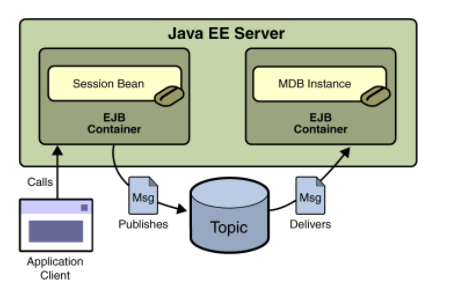


Рисунок 1 – принцип работы Session Bean

Существует 2 типа session-beans: **stateless** и **stateful**.

**Stateful**– автоматически сохраняют свое состояние между разными клиентскими вызовами.

**Stateless**– используются для реализации **бизнесс-процессов**, которые могут быть завершены за одну операцию.

**2. Message-Driven beans**– компонент является корпоративным компонентом, который позволяет Java EE приложениям обрабатывать сообщения асинхронно.

Этот тип бинов обычно действует в качестве слушателя JMS-сообщения, который похож на слушателя событий, но получает JMS-сообщений вместо событий. Сообщения могут быть отправлены на любой компонент Java EE (клиентское приложение, другой компонент, или веб-компонент) или JMS приложение или систему, которая не использует Java EE технологий.

Message-Driven beans может обрабатывать не только JMS сообщения но и других видов сообщений.

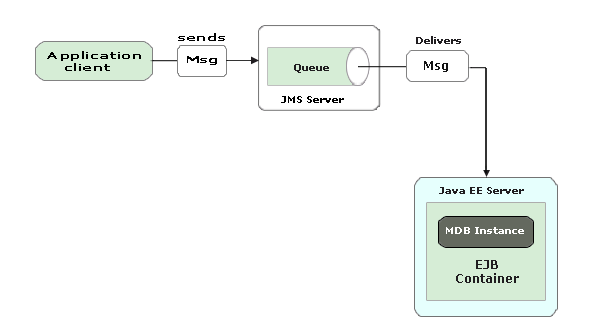


Рисунок 2 – принцип работы **Message-Driven beans**

На схеме выше можно наблюдать общение между приложением и сервером с помощью очереди куда поступают сообщения.

**3. Entities** – это сущности каких-то объектов и в EJB оно является хранилищем данных на период жизненного цикла Entity.

Entities является свое-родным отображением таблиц в БД.

Одним из главным достоинством EJB3 стал новый механизм работы с persistence, он дает возможность автоматически сохранять объекты в реляционной БД используя технологию ORM.

Для работы с entity был создан JPA (Java Persistence API).

**JPA определяет стандарт для**:

1) конфигурации маппинга сущностей приложения и их отображения в таблицах БД;

2) EntityManager API – позволяет выполнять CRUD (create, read, update, delete) операции над сущностями;

3) Java Persistence Query Language (JPQL) – для поиска и получения данных приложения;

# Скриншоты работы программы

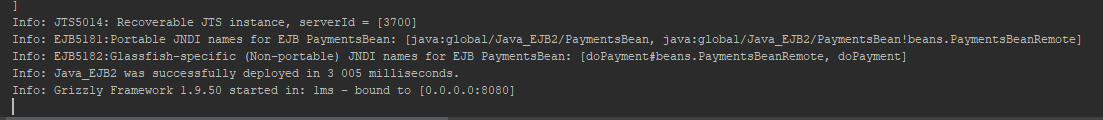


Рисунок 3 – запуск сервера

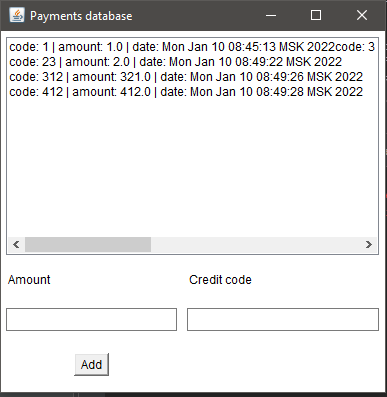


Рисунок 4 – запуск клиента

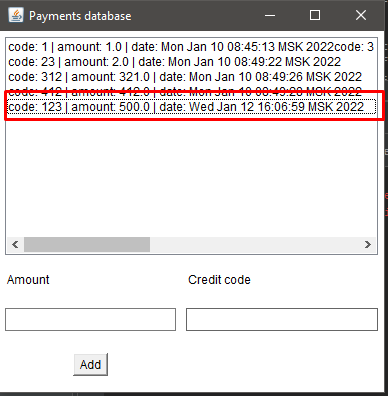


Рисунок 5 – добавление записи об оплате

# Листинг программного кода

**PaymentBean.java**

**package** beans;

**import** **beans.PaymentsBeanRemote**;

**import** **common.infrastructure.ISerializable**;

**import** **common.settings.Constants**;

**import** **javax.ejb.Stateless**;

**import** **java.io.\***;

**import** **java.util.ArrayList**;

**@Stateless**(mappedName = "doPayment")

**public** **class** **PaymentsBean** **implements** PaymentsBeanRemote {

**@Override**

**public** **void** **addPayment**(ISerializable model) **throws** IOException {

FileWriter fw = **new** FileWriter(Constants.FileName, **true**);

BufferedWriter bw = **new** BufferedWriter(fw);

bw.write(model.getFileRepresentation());

bw.close();

}

**@Override**

**public** ArrayList<String> **getAllPayments**() **throws** IOException {

ArrayList<String> result = **new** ArrayList<String>();

ensureFileExist();

FileReader fr = **new** FileReader(Constants.FileName);

BufferedReader br = **new** BufferedReader(fr);

**while** (br.ready()) {

result.add(br.readLine());

}

br.close();

fr.close();

**return** result;

}

**private** **void** **ensureFileExist**() **throws** IOException

{

File yourFile = **new** File(Constants.FileName);

yourFile.createNewFile(); // if file already exists will do nothing

}

}

**GUI.java**

**package** client.view;

**import** **client.ServerContext**;

**import** **java.awt.\***;

**import** **java.awt.event.WindowAdapter**;

**import** **java.awt.event.WindowEvent**;

**import** **java.util.ArrayList**;

**public** **class** **GUI** **extends** Frame {

**public** **GUI**(String title) {

**super**(title);

configureWindow();

configureServerContext();

configureComponents();

}

**private** ServerContext serverContext;

**private** List list;

**private** **void** **configureWindow**() {

setLayout(**new** GridBagLayout());

setSize(**400**,**400**);

setPreferredSize(getSize());

addWindowListener(**new** WindowClose());

setVisible(**true**);

}

**private** **void** **configureServerContext**() {

serverContext = **new** ServerContext();

}

**private** **void** **configureComponents**() {

GridBagConstraints gbc = **new** GridBagConstraints();

gbc.insets = **new** Insets(**5**,**5**,**5**,**5**);

list = **new** List();

gbc.fill = GridBagConstraints.BOTH;

gbc.weighty = **1**;

gbc.weightx = **1**;

gbc.gridheight = **3**;

gbc.gridwidth = **5**;

gbc.gridx = **0**;

gbc.gridy = **0**;

add(list, gbc);

Label valueLabel = **new** Label("Amount");

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridx = **0**;

gbc.gridy = **3**;

gbc.gridheight = **1**;

gbc.gridwidth = **2**;

gbc.weighty = **0.05**;

add(valueLabel, gbc);

TextField valueInput = **new** TextField();

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridx = **0**;

gbc.gridy = **4**;

gbc.gridheight = **1**;

gbc.gridwidth = **2**;

add(valueInput, gbc);

Label codeLabel = **new** Label("Credit code");

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridx = **3**;

gbc.gridy = **3**;

gbc.gridheight = **1**;

gbc.gridwidth = **2**;

add(codeLabel, gbc);

TextField codeInput = **new** TextField();

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridx = **3**;

gbc.gridy = **4**;

gbc.gridheight = **1**;

gbc.gridwidth = **2**;

add(codeInput, gbc);

Panel leftPanel = **new** Panel(**new** FlowLayout());

gbc.gridx = **0**;

gbc.gridy = **5**;

gbc.gridheight = **1**;

gbc.gridwidth = **2**;

add(leftPanel, gbc);

Button addBtn = **new** Button("Add");

addBtn.addActionListener(**new** AddButtonActionListener(list, valueInput, codeInput, serverContext));

leftPanel.add(addBtn);

SeedData(list);

}

**private** **void** **SeedData**(List list) {

list.removeAll();

ArrayList<String> newList = serverContext.getPaymentsList();

**for** (String s : newList) {

list.add(s);

}

}

**private** **static** **class** **WindowClose** **extends** WindowAdapter {

**public** **void** **windowClosing**(WindowEvent we) {

System.exit(**0**);

}

}

}

**AddButtonActionListener.java**

**package** client.view;

**import** **client.ServerContext**;

**import** **java.awt.\***;

**import** **java.awt.event.ActionEvent**;

**import** **java.awt.event.ActionListener**;

**import** **java.util.ArrayList**;

**public** **class** **AddButtonActionListener** **implements** ActionListener {

**private** **final** List \_list;

**private** **final** TextField \_valueInput;

**private** **final** TextField \_codeInput;

**private** **final** ServerContext \_remoteObject;

**public** **AddButtonActionListener**(List list, TextField valueInput, TextField codeInput, ServerContext remoteObject) {

**super**();

\_list = list;

\_valueInput = valueInput;

\_codeInput = codeInput;

\_remoteObject = remoteObject;

}

**@Override**

**public** **void** **actionPerformed**(ActionEvent event) {

String valueText = \_valueInput.getText();

String codeText = \_codeInput.getText();

**if**(!valueText.trim().equals("") && !codeText.trim().equals("")) {

\_remoteObject.insertPayment(valueText, codeText);

\_list.clear();

ArrayList<String> newList = \_remoteObject.getPaymentsList();

**for** (String s : newList) {

\_list.add(s);

}

\_valueInput.setText("");

\_codeInput.setText("");

}

}

}

**ServerContext.java**

**package** client;

**import** **beans.PaymentsBeanRemote**;

**import** **common.models.Payment**;

**import** **javax.naming.InitialContext**;

**import** **javax.naming.NamingException**;

**import** **java.io.IOException**;

**import** **java.util.ArrayList**;

**import** **java.util.Properties**;

**public** **class** **ServerContext** {

PaymentsBeanRemote paymentsBean;

**public** **ServerContext**() {

**try** {

Thread.currentThread().setContextClassLoader(getClass().getClassLoader());

Properties properties = **new** Properties();

properties.setProperty("java.naming.factory.initial", "com.sun.enterprise.naming.SerialInitContextFactory");

properties.setProperty("java.naming.factory.url.pkgs", "com.sun.enterprise.naming");

properties.setProperty("java.naming.factory.state", "com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl");

InitialContext initialContext = **new** InitialContext(properties);

paymentsBean = (PaymentsBeanRemote) initialContext.lookup("doPayment");

} **catch** (NamingException e) {

e.printStackTrace();

}

}

**public** ArrayList<String> **getPaymentsList**() {

**try** {

**return** paymentsBean.getAllPayments();

}

**catch** (IOException e) {

System.err.println("DB error");

e.printStackTrace();

**return** **new** **ArrayList**();

}

}

**public** **void** **insertPayment**(String value, String code) {

**try** {

**double** parsedValue = Double.parseDouble(value);

paymentsBean.addPayment(**new** Payment(code, parsedValue));

System.out.println("Payment info was successfully added");

} **catch**(NumberFormatException e) {

System.err.println("Incorrect data");

e.printStackTrace();

}

**catch** (IOException e) {

System.err.println("Error during data insert");

e.printStackTrace();

}

}

}