Степанов Е.О., к.ф.-м.н.
Ярцев Б.М.

Учебно-методическое пособие по дисциплине «Архитектуры и технологии разработки распределенного программного обеспечения»

Описание практических работ студентов (ЛП)

Оглавление

Технология CORBA

В данном разделе содержатся задания для лабораторных работ по теме технология CORBA. Для их выполнения следует ознакомиться с примером «Служба мгновенных сообщений» из соответствующего раздела УМП.

Лабораторная работа 1: Использование POA

В данной работе надо будет написать клиента, как в примере «Служба мгновенных сообщений», используя объектный адаптер – *Portable Object Adapter* (*POA*). В связи с этим в реализации сервера (сервер из примера «Служба мгновенных сообщений») произошли следующие изменения:

- За инициализацией *ORB* следует получение ссылки на корневой объектный адаптер `rootPOA`. Корневой адаптер всегда существует, ссылку на него можно получить с помощью метода *ORB* `resolve_initial_references`**.**
- Далее создается массив политик, который после передается в качестве одного из параметров конструктору нового объектного адаптера `myPOA`. Адаптер `myPOA` создается внутри корневого объектного адаптера (*POA* можно вкладывать друг в друга).
- После создания серванта имя будущего *CORBA*-объекта представляется в виде набора байт. Далее объектному адаптеру дается команда активировать *CORBA*-объект с заданным именем и ссылкой на сервант.
- В корневом объектном адаптере активируется менеджер объектных адаптеров.
- Для ожидания подключений в данном примере используется метод *ORB* `run()`. В предыдущих примерах использовался альтернативный способ – присоединение сервера к некоторому потоку.

Класс сервера:

```
import org.omg.PortableServer.*;

public class Server {

     public static void main(String[] args) {
          try {
               org.omg.CORBA.ORB orb =
               org.omg.CORBA.ORB.init(args,null);
               POA rootPOA = POAHelper.narrow(
               orb.resolve_initial_references("RootPOA"));
```

```
                org.omg.CORBA.Policy[] policies = {
                     rootPOA.create_lifespan_policy(
                     LifespanPolicyValue.PERSISTENT)
                };

                POA myPOA = rootPOA.create_POA(
                     "messaging_poa", rootPOA.the_POAManager(),
                     policies );
                MessagingServiceImpl messagingServant =

                new MessagingServiceImpl();
                byte[] messagingId =
                "MessagingService".getBytes();

                myPOA.activate_object_with_id(
                     messagingId, messagingServant);
                rootPOA.the_POAManager().activate();

                System.out.println(
                     myPOA.servant_to_reference(messagingServant)
                     + " is ready.");
                orb.run();
           } catch (Exception e) {
                e.printStackTrace();
           }
     }
}
```

Класс серванта теперь наследуется от скелетона `Message.MessagingServicePOA`. Здесь нет необходимости указывать имя *CORBA*-объекта в качестве параметра конструктора родителя.

Класс серванта:

```
public class MessagingServiceImpl extends
Message.MessagingServicePOA {
     private HashMap nameToId;
     private ArrayList users;
     public MessagingServiceImpl() {
          System.out.println("Constructing
          MessagingServiceImpl");
          nameToId = new HashMap();
          users = new ArrayList();
     }

     public int registerUser (String userName, String password) {
          if (nameToId.containsKey(userName)) {
               System.out.println("User " + userName +
               " already registered");
               return 0;
          } else {
```

```
                int uid = nameToId.size();
                nameToId.put(userName, new Integer(uid));
                users.add(new User(userName, password));
                notifyAllUsers(++uid, false);
                System.out.println("User " + userName +
                     " registered successfully, user ID is " +
                     (new Integer(uid)).toString());
                return uid;
           }
      }
      ...
      private void notifyAllUsers(int uid, boolean online)
      {
           String userName = ((User)users.get(uid - 1)).name;
           for (int i = 0; i < users.size(); ++i) {
                if (i != uid - 1) {
                     User buddy = (User)users.get(i);
                     if (buddy.online) {
                          buddy.receiver.userStatusNotification(
                          uid, userName, online);
                     }
                }
           }
      }
      class User {...}
      class Msg  {...}
}
```

Задание:

Написать класс клиента.

Указания для написания класса клиента:

- После получения ссылки на корневой объектный адаптер создаются политики и адаптер `callbackPOA`, в котором будет размещен *CORBA*-объект получатель сообщений (`MessageReceiver`).
- Далее с помощью механизма `bind` получаем ссылку на сервер.
- Создание пользовательского интерфейса.
- Активация получателя сообщений и менеджера объектных адаптеров.
- Получение ссылки на `MessageReceiver` для последующей передачи серверу.

Лабораторная работа 2: Использование Сервиса имен

В данной лабораторной работе будет рассмотрен еще один способ получения ссылки на *CORBA*-объект – использование сервиса имен. Этот способ более удобен, чем прямое задание *IOR*, и, в отличие от механизма `bind`, **является стандартным**.

Класс серванта:

```
public class MessagingServiceImpl extends
Message.MessagingServicePOA {
     private HashMap nameToId;
     private ArrayList users;
     public MessagingServiceImpl() {
          System.out.println("Constructing
          MessagingServiceImpl");
          nameToId = new HashMap();
          users = new ArrayList();
     }

     public int registerUser (String userName, String password) {
          if (nameToId.containsKey(userName)) {
               System.out.println("User " + userName +
               " already registered");
               return 0;
          } else {
               int uid = nameToId.size();
               nameToId.put(userName, new Integer(uid));
               users.add(new User(userName, password));
               notifyAllUsers(++uid, false);
               System.out.println("User " + userName +
                    " registered successfully, user ID is " +
                    (new Integer(uid)).toString());
               return uid;
          }
     }
     ...
     private void notifyAllUsers(int uid, boolean online)
     {
          String userName = ((User)users.get(uid - 1)).name;
          for (int i = 0; i < users.size(); ++i) {
               if (i != uid - 1) {
                    User buddy = (User)users.get(i);
                    if (buddy.online) {
                         buddy.receiver.userStatusNotification(
                         uid, userName, online);
                    }
               }
          }
     }
```

```
     class User {...}
     class Msg  {...}
}
```

Класс клиента:

```
public static void main(String[] args) {
     try {
          ORB orb = ORB.init(args,null);
          org.omg.CORBA.Object rootObj =
               orb.resolve_initial_references("NameService");
          NamingContext root =
          NamingContextHelper.narrow(rootObj);

          NameComponent[] path =
               {
                    new NameComponent("MessagingService", "")};

          org.omg.CORBA.Object msgObj = root.resolve(path);
          Message.MessagingService service =
               Message.MessagingServiceHelper.narrow(msgObj);
          MessagingFrame mf = new MessagingFrame(service);
          MessageReceiverImpl mr = new MessageReceiverImpl(mf);
          POA rootPOA =
          (POA)orb.resolve_initial_references("RootPOA");
          rootPOA.the_POAManager().activate();
          Message.MessageReceiver receiver =
               Message.MessageReceiverHelper.narrow(
          rootPOA.servant_to_reference(mr));
          LoginFrame loginFrame =
               new LoginFrame(service, mf,
               orb.object_to_string(receiver));
          loginFrame.show();
     } catch (Throwable t) {
          t.printStackTrace();
     }
}
```

При использовании сервиса имен необходимо использовать дополнительные параметры командам запуска сервера и клиента. Кроме того, необходимо запустить сам сервис имен. Это делается следующим образом:

```
     start                    nameserv                    -J-
Dvbroker.se.iiop_tp.scm.iiop_tp.listener.port=<port> NameService
     start          vbj          Server          -ORBInitRef
NameService=iioploc://<host>:<port>/NameService
     start          vbj          Client          -ORBInitRef
NameService=iioploc://<host>:<port>/NameService
```

Здесь `host` – идентификатор узла, на котором запущен сервис имен. Порт может быть любым свободным, но должен согласовываться при запуске сервиса имен, клиента и сервера.

Задание:

Написать класс сервера.

Указания по написанию класса сервера:

- Подключение пакета `CosNaming`
- Получение ссылки на сервис имен
- Создание пути размещения объекта в виде последовательности компонентов имени (в данном случае имя состоит из одного компонента, однако в более сложных случаях удобно использовать иерархическую структуру)
- Установление соответствия между полным именем объекта и ссылкой на него. Метод `rebind` контекста именования отличается от его метода `bind` тем, что при конфликте имен он будет разрешен в пользу нового сопоставления.

Технология *Enterprise Java Beans*

В данном разделе содержатся задания для лабораторных работ по теме *Enterprise Java Beans*. Для выполнения следующих двух лабораторных работ следует ознакомиться с примером компонента, предназначенного для преобразования курсов валют, описанного в следующем разделе.

Пример “Конвертор валют”

В данном примере мы покажем разработку сеансового компонента без состояния (*Stateless Session Bean*) для конвертации денежных сумм из одной валюте в другую. В качестве валют будут использоваться доллары, евро и йены. Компонент будет переводить сумму в долларах в сумму в евро или йенах.

Удаленный интерфейс

В удаленном интерфейсе (*Remote Interface*) определим два метода – `dollarToEuro(double)` и `dollarToYena(double)`. Эти методы, получив в качестве параметра сумму в долларах, переведут эту сумму по некоторому постоянному курсу и вернут сумму в евро или йенах соответственно.

```
public interface CurrencyRemote extends EJBObject {
      public double dollarToEuro(double dollars) throws RemoteException;
      public double dollarToYena(double yena) throws RemoteException;
}
```

Домашний интерфейс

Домашний интерфейс (*Home Interface*) состоит из одного метода.

```
public interface CurrencyHome extends EJBHome {
      public CurrencyRemote create() throws RemoteException, CreateException;
}
```

Класс компонента

В классе компонента заданы для простоты два постоянных курса перевода валют (задание этих констант, а также основных используемых извне методов выделены жирным шрифтом).

```
public class CurrencyBean implements SessionBean {
      private static final long serialVersionUID = 8897567469393327031L;

      private double dollarToYenaRate = 100.25;
      private double dollarToEuroRate = 0.83;

      public void ejbCreate() {

      }
      public double dollarToEuro(double dollars) {
            return dollars * dollarToEuroRate;
      }
      public double dollarToYena(double dollars) {
            return dollars * dollarToYenaRate;
      }

      public void ejbActivate() throws EJBException, RemoteException {
      }

      public void ejbPassivate() throws EJBException, RemoteException {
      }

      public void ejbRemove() throws EJBException, RemoteException {
      }

      public void setSessionContext(SessionContext arg0) throws EJBException,
            RemoteException {
      }
}
```

Дескриптор развертывания

Дескриптор развертывания для этого компонента ничем принципиально не отличается от дескриптора развертывания компонента из предыдущего примера.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar id="ejb-jar_ID" version="2.1"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
```

```
      <description> Second session bean </description>
      <display-name>CurrencyConvertorBean</display-name>

      <enterprise-beans>
            <session>
                  <ejb-name>CurrencyBean</ejb-name>
                  <home>currencyConvertorBean.CurrencyHome</home>
                  <remote>currencyConvertorBean.CurrencyRemote</remote>
                  <ejb-class>currencyConvertorBean.CurrencyBean</ejb-class>
                  <session-type>Stateless</session-type>
                  <transaction-type>Container</transaction-type>
            </session>
      </enterprise-beans>
</ejb-jar>
```

Создание проекта для компонента

В этом примере не будет рассматриваться создание простого *Java*-проекта для компонента, ограничимся *EJB*-проектом. Создаем *EJB*-проект *CurrencyConvertorBean*. Также в каталоге ejbModule создаем интерфейсы и классы, вставляем дескриптор развертывания. После этого добавляем проект к контейнеру *JBoss*.

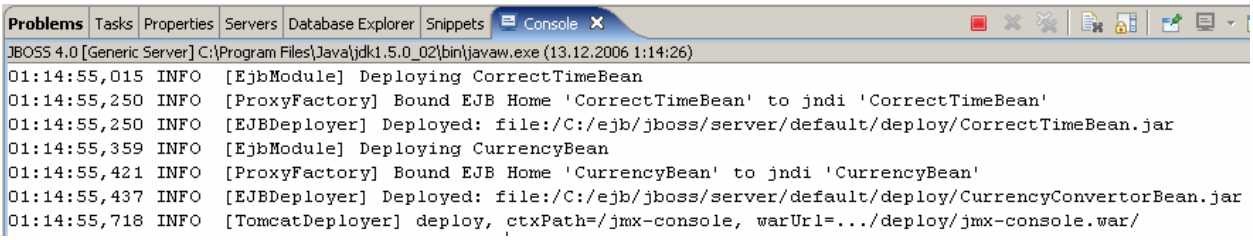

Рис. 1. Теперь их два

Если все было сделано правильно – то при запуске *JBoss* появится сообщение об успешном развертывании двух компонентов (компонента из предыдущего примера и текущего компонента) (рис. 1).

Лабораторная работа 1: Создание проекта для клиентского приложения

В данной работе от вас требуется написание небольшого приложения, которое будет работать с компонентом *CurrencyConvertorBean*. Это будет графическое приложение, в котором пользователь сможет ввести наименования валют, и сумму денег в одной валюте. После этого приложение выведет сумму денег в другой валюте.

Примерный внешний вид приложения, которое вам следует создать представлен на следующих рисунках.

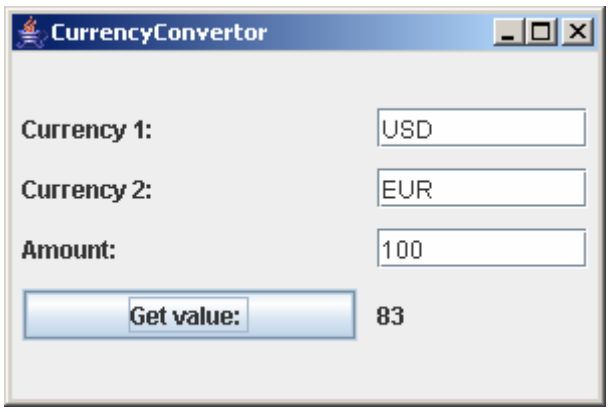

Рис. 2. Перевод долларов в евро

Введем в качестве первой валюты доллар, в качестве второй – евро, а в качестве суммы – 100 долларов. Щелкнем кнопку Get value**:**, и появится сумма уже в евро – 83 евро.

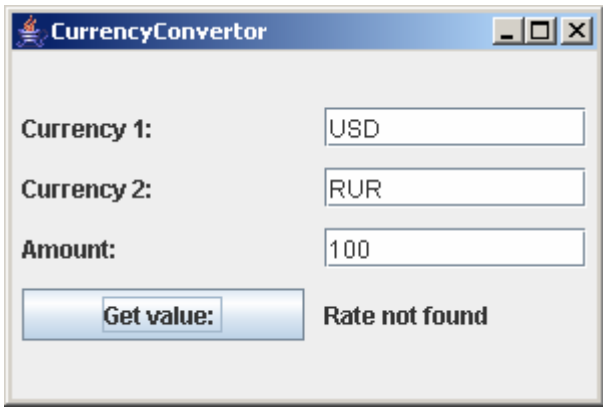

Рис. 3. Курс перевода не найден

Если же в качестве валют для перевода указывается что-либо отличное от пар USD -> EUR или USD -> YEN, то тогда выводится сообщение Rate not found (рис. 3).

Лабораторная работа 2: Получение котировок валют из базы данных

В данной работе требуется разработать компонент с функциональностью, похожей на функциональность компонента из предыдущего примера. Но теперь компонент будет брать информацию о курсах валют не из констант классов (как это было в предыдущем примере), а из базы данных. В качестве базы данных используется *Oracle 9i*.

Удаленный интерфейс

В удаленном интерфейсе (*Remote Interface*) определим главный метод, реализующий требуемую функциональность:

```
double convert(String cur1, String cur2, double amount) throws
RemoteException, CurrencyRateNotFoundException
```

Этот метод, получив на вход сумму в одной валюте, описываемой строкой `cur1`, переведет ее в сумму в другой валюте, описываемой строкой `cur2`. Если одна или обе из валют не найдены, то генерируется исключение `CurrencyRateNotFoundException`.

```
public interface DBCurrencyRemote extends EJBObject {
     public double convert(String cur1, String cur2, double
     amount)
          throws RemoteException, CurrencyRateNotFoundException;
}
```

Домашний интерфейс

Домашний интерфейс (*Home Interface*) по сути дела ничем не отличается от домашних интерфейсов (*Home Interface*) компонентов из предыдущих примеров.

```
public interface DBCurrencyHome extends EJBHome {
     public DBCurrencyRemote create() throws RemoteException,
          CreateException;
}
```

Создание таблицы в базе данных

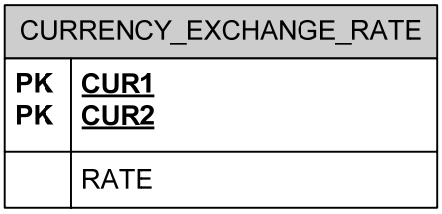

Рис. 4. Инфологическая модель базы данных

Теперь необходимо создать таблицу курсов валют в базе данных, в которой будут храниться строки следующего вида: CUR1, CUR2, RATE. Строка уникальным образом определяется упорядоченной парой идентификаторов типов валют: (CUR1, CUR2), то есть для двух заданных валют CUR1 и CUR2 может быть задан только один курс перевода RATE валюты CUR1 в валюту CUR2 (паре (CUR2, CUR1) естественно, соответствует другой курс). Инфологическая модель этой простой базы данных представлена на рис. 4.

Создадим такую таблицу, и введем значения курсов для некоторых пар валют. Ниже приводится *SQL*-код, который создает необходимую таблицу и заполняет ее информацией.

```
create table CURRENCY_EXCHANGE_RATE
(
  CUR1 varchar(30),
```

```
  CUR2 varchar(30),
  RATE float,
  primary key (CUR1, CUR2)
);

insert into CURRENCY_EXCHANGE_RATE
(CUR1, CUR2, RATE) values ('EUR', 'USD', 1.3);

insert into CURRENCY_EXCHANGE_RATE
(CUR1, CUR2, RATE) values ('USD', 'EUR', 0.8);

insert into CURRENCY_EXCHANGE_RATE
(CUR1, CUR2, RATE) values ('EUR', 'RUR', 34.4);

insert into CURRENCY_EXCHANGE_RATE
(CUR1, CUR2, RATE) values ('USD', 'RUR', 26.5);

insert into CURRENCY_EXCHANGE_RATE
(CUR1, CUR2, RATE) values ('RUR', 'EUR', 0.33);

insert into CURRENCY_EXCHANGE_RATE
(CUR1, CUR2, RATE) values ('RUR', 'USD', 0.45);
```

Если таблица с таким названием уже существует в базе данных, то перед выполнением приведенного выше скрипта необходимо выполнить следующую *SQL*-команду:

```
drop table CURRENCY_EXCHANGE_RATE
```

Задание

Реализуйте модифицированную версию компонента Конвертор Валют, которая использует для преобразования курсов валют данные из базы, структура которой описана в предыдущем разделе.

Лабораторная работа 3: Разработка корзины с покупками для Интернет-магазина

Для выполнения следующей лабораторной работы (лабораторная работа содержит 3 задания, поэтому ее надо разбить на 3 занятия) рассмотрим следующий пример. В нем будет продемонстрирована работа такого компонента, имеющегося у большинства интернет магазинов как корзина с покупками. Пользователь, зайдя на страницу нашего магазина сможет залогиниться, введя свое имя, просмотреть список доступных товаров, добавить необходимые ему товары в корзину, сделать заказ, пересчитать количество отдельных товаров в заказе, а также просмотреть свои предыдущие заказы.

Торговать интернет магазин будет сигаретами.

Создание таблиц в базе данных

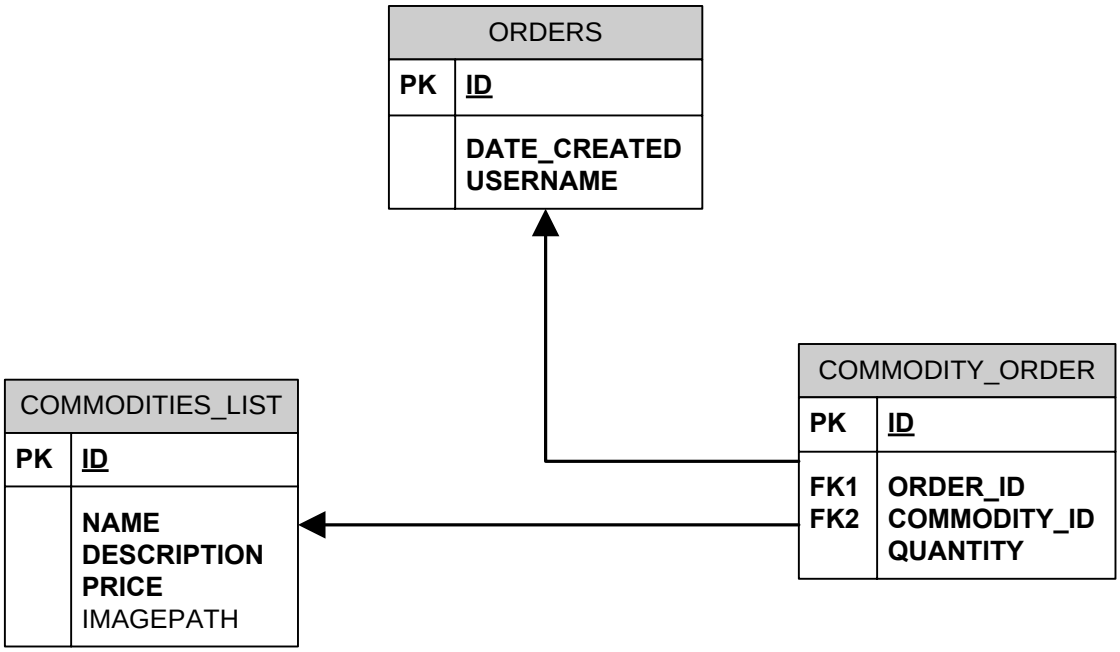

Рис. 5. Инфологическая модель базы данных

Список товаров, сделанные заказы и их состав будут храниться в базе данных, инфологическая модель которой представлена на рис. 5. Эта база данных состоит из трех основных таблиц:

- Таблицы COMMODITIES_LIST, в которой хранится информация о всех доступных товарах в магазине. Поле NAME – название товара, DESCRIPTION – его описание, PRICE – цена в долларах США, а необязательное поле IMAGEPATH – идентификатор изображения, соответствующее товару (используется при отображении списка товаров в интернете).
- Таблицы ORDERS, в которой хранится информация о заказе. DATE_CREATED – дата, когда заказ был сделан и USERNAME – имя пользователя, сделавшего заказ.
- Таблицы COMMODITY_ORDER, которая описывает товары, из которых состоит заказ. ORDER_ID – идентификатор заказа, COMMODITY_ID – идентификатор товара, QUANTITY – количество экземпляров этого товара в заказе. ORDER_ID И COMMODITY_ID связаны с таблицами COMMODITIES_LIST и ORDERS по внешнему ключу.

Для того, чтобы создать таблицы в базе данных *Oracle 9i*, необходимо выполнить следующий *SQL*-код. Помимо создания таблиц в базе данных, данный *SQL*-код заполняет таблицы некоторым набором значений по умолчанию. Для каждой таблицы определен триггер, автоматически генерирующий поле ID для каждой добавленной строки таблицы. Вначале создается пакет ORDER_ID_PKG, в котором определена переменная,

хранящая целое значение, и функция ее возвращающая. Триггер, генерирующий поле ID для таблицы ORDERS, сохраняет новое сгенерированное значение в поле LAST_ID пакета ORDER_ID_PKG. Это сделано для того, чтобы получить ID последнего добавленного заказа и использовать его при добавлении строк в таблицу COMMODITY_ORDER.

```
create or replace package ORDER_ID_PKG as
  LAST_ID integer;
  function GET_LAST_ID return integer;
end ORDER_ID_PKG;
/

create or replace package body ORDER_ID_PKG as
  function GET_LAST_ID return integer is
    begin
      return LAST_ID;
    end GET_LAST_ID;

end ORDER_ID_PKG;
/
create table COMMODITIES_LIST
(
  ID integer not null,
  NAME varchar(20) not null,
  DESCRIPTION varchar(50) not null,
  PRICE float not null,
  IMAGEPATH varchar(30),
  primary key(ID)
);

create sequence COMMODITIES_SEQ
start with 1
increment by 1
nomaxvalue;

create trigger COMMODITIES_TRIGGER
before insert on COMMODITIES_LIST
for each row
begin
select COMMODITIES_SEQ.nextval into :new.id from dual;
end;
/

insert into COMMODITIES_LIST(NAME, DESCRIPTION, PRICE,
IMAGEPATH)
values('Kent 1', 'Normal cigarettes', 1.99, 'img/kent1.jpg');

insert into COMMODITIES_LIST(NAME, DESCRIPTION, PRICE,
IMAGEPATH)
values('Kent 4', 'Light cigarettes', 1.99, 'img/kent4.jpg');
```

```
insert into COMMODITIES_LIST(NAME, DESCRIPTION, PRICE,
IMAGEPATH)
values('Kent 8', 'Very light cigarettes', 1.99,
'img/kent8.jpg');

insert into COMMODITIES_LIST(NAME, DESCRIPTION, PRICE,
IMAGEPATH)
values('Lucky Strike', 'Pure American Taste', 2.30,
'img/lucky.jpg');

insert into COMMODITIES_LIST(NAME, DESCRIPTION, PRICE,
IMAGEPATH)
values('Portugas', 'Fine Cuban Cigar', 10.95,
'img/portugas.jpg');

select * from COMMODITIES_LIST;
/

create table ORDERS
(
  ID integer not null,
  DATE_CREATED DATE not null,
  USERNAME varchar(20) not null,
  primary key(ID)
);

create sequence ORDERS_SEQ
start with 1
increment by 1
nomaxvalue;

create trigger ORDERS_TRIGGER
before insert on ORDERS
for each row
begin
select ORDERS_SEQ.nextval into :new.id from dual;
end;
/

create table COMMODITY_ORDER
(
  ID integer not null,
  ORDER_ID integer not null,
  COMMODITY_ID integer not null,
  QUANTITY integer not null,
  primary key(ID)
);

alter table COMMODITY_ORDER
add constraint FK_ORDER
foreign key (ORDER_ID)
references ORDERS(ID);
```

```
alter table COMMODITY_ORDER
add constraint FK_COMMODITIES
foreign key(COMMODITY_ID)
references COMMODITIES_LIST(ID);

create sequence COMMODITY_ORDER_SEQ
start with 1
increment by 1
nomaxvalue;

create trigger COMMODITY_ORDER_TRIGGER
before insert on COMMODITY_ORDER
for each row
begin
select COMMODITY_ORDER_SEQ.nextval into :new.id from dual;
end;
/
```

Если таблицы уже созданы в базе данных, то необходимо предварительно выполнить следующий *SQL*-код, который удалит эти таблицы и последовательности при помощи которых генерировалось автоматически значение основного ключа строки в таблице.

```
drop table COMMODITY_ORDER;
drop table ORDERS;
drop table COMMODITIES_LIST;
drop sequence COMMODITY_ORDER_SEQ;
drop sequence COMMODITIES_SEQ;
drop sequence ORDERS_SEQ;
```

Стоит сделать несколько замечаний – для таблицы COMMODITY_ORDER выполняется операция `drop` в самом начале кода. Это сделано из-за того, что таблица COMMODITY_ORDER связана внешним ключом с двумя другими таблицами. БД *Oracle* не даст нам выполнить операцию drop для двух других таблиц, пока связь посредством внешнего ключа с таблицей COMMODITY_ORDER.

Для того чтобы соотнести объекты из базы данных с *Java*-классами был создан класс абстрактный класс `DataItem`. В нем есть поле `id`, которое соответствует ID объекта в базе данных (как было описано выше, у любого объекта в этой базе данных есть свой ID). Класс находится в пакете `dataObjects`.

```
public abstract class DataItem implements Serializable {
     protected int id = -1;

     public int getId() {
          return id;
     }

     public void setId(int id) {
          this.id = id;
```

```
    }
}
```

От класса `DataItem` наследуются три других класса – `Commodity`, `CommodityOrder` и `Order`. `Commodity` соответствует товару из таблицы COMMODITY_LIST, `CommodityOrder` – товару из заказа в таблице COMMODITY_ORDERS, `Order` – заказу из таблицы ORDERS.

Исходный код класса `Commodity` представлен далее:

```
public class Commodity extends DataItem {

    private static final long serialVersionUID = -
7004067843847157531L;

    private String name;

    private String description;

    private String imagePath;

    private double price;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getImagePath() {
        return imagePath;
    }

    public void setImagePath(String imagePath) {
        this.imagePath = imagePath;
```

```
    }

    public String toString() {
        return "ID=" + id + "; NAME=" + name + "; DESCRIPTION"
        + description + "; PRICE=" + price +
            "; IMAGEPATH=" + imagePath;
    }
}
```

Исходный код класса `CommodityOrder` представлен далее:

```
public class CommodityOrder extends DataItem {

    private static final long serialVersionUID = -
592875988844838475L;

    private int commodityId;

    private int commodityQuantity;

    public int getCommodityId() {
        return commodityId;
    }

    public void setCommodityId(int commodityId) {
        this.commodityId = commodityId;
    }

    public int getCommodityQuantity() {
        return commodityQuantity;
    }

    public void setCommodityQuantity(int commodityQuantity) {
        this.commodityQuantity = commodityQuantity;
    }
}
```

Исходный код класса `Order` представлен далее:

```
public class Order extends DataItem {

    private static final long serialVersionUID = -
3276613990865210869L;

    private String name;
    private Date date;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
```

```
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

Компоненты

Для реализации заявленной функциональности будут использоваться три сеансовых компонента *EJB*: *CommoditiesListBean*, *OrdersListBean* и *ShoppingBasketBean*. *CommoditiesListBean* и *OrdersListBean* являются сеансовыми компонентами без состояния, а *ShoppingBasketBean* – компонентом с состоянием (*stateful*).

CommoditiesListBean используется для поиска товара в базе данных по ID товара, а также для получения всех товаров в базе данных.

OrdersListBean используется для составления списка сделанных заказов по имени пользователя, а также для получения списка товаров, входящих в заказ, описываемый при помощи ID.

ShoppingBasketBean используется для реализации логики корзины в интернет магазине. При помощи него можно добавлять товары в корзину, пересчитывать количество товаров и делать заказ.

Удаленный интерфейс *CommoditiesListBean*

В удаленном интерфейсе (*Remote Interface*) определим два метода:

- `public List getAvailableCommodities() throws RemoteException, SQLException`. Этот метод возвращает список из объектов класса `Commodity`, которые соответствуют записям в таблице COMMODITIES_LIST.
- `public Commodity getCommodityById(int id) throws RemoteException, SQLException`. Этот метод по `id` возвращает объект класса `Commodity`, соответствующий товару в базе данных.

```
public interface CommoditiesListRemote extends EJBObject {
    public List getAvailableCommodities() throws
    RemoteException, SQLException;

    public Commodity getCommodityById(int id) throws
    RemoteException, SQLException;
}
```

Домашний интерфейс класса *CommoditiesListBean*

Этот интерфейс аналогичен домашним интерфейсам из предыдущих примеров.

```
public interface CommoditiesListHome extends EJBHome {
     public CommoditiesListRemote create() throws
     RemoteException, CreateException;
}
```

Класс компонента *CommoditiesListBean*

В классе компонента заданы два основных метода – `List getAvailableCommodities()` и `Commodity getCommodityById(int id)`. В первом методе выполняется SELECT запрос в базе данных, после чего на основе сгенерированного `ResultSet` создается список из объектов класса `Commodity`. Выполнение запроса к базе данных, а также формирование объектов `Commodity` выделено далее жирным шрифтом.

```
public class CommoditiesListBean implements SessionBean {

     private static final long serialVersionUID = -
8641939595462379799L;

     private DataSource dataSource = null;

     public List getAvailableCommodities() throws SQLException {
          if (dataSource == null)
               throw new SQLException("Datasource not loaded");

          Connection conn = dataSource.getConnection();

          PreparedStatement p = conn.prepareStatement("SELECT *
          FROM " +  " COMMODITIES_LIST");

          ResultSet r = p.executeQuery();

          List arrayList = new ArrayList();

          while (r.next()) {
               Commodity commodity = new Commodity();
               commodity.setId(r.getInt("ID"));
               commodity.setName(r.getString("NAME"));

     commodity.setDescription(r.getString("DESCRIPTION"));
               commodity.setPrice(r.getDouble("PRICE"));
               commodity.setImagePath(r.getString("IMAGEPATH"));
               arrayList.add(commodity);
          }

          conn.close();
          return arrayList;
     }
```

Во втором методе выполняется поиск товара по его ID. Если такой товар найден, то возвращается созданный по `ResultSet` объект `Commodity`. Выполнение запроса к базе данных и формирование объекта `Commodity` выделено далее жирным шрифтом.

```
     public Commodity getCommodityById(int id) throws
SQLException {
          if (dataSource == null)
               throw new SQLException("Datasource not loaded");

          Connection conn = dataSource.getConnection();

          PreparedStatement p = conn.prepareStatement("SELECT *
FROM " +
               " COMMODITIES_LIST WHERE ID=?");

          p.setInt(1, id);
          ResultSet r = p.executeQuery();

          Commodity commodity = new Commodity();

          if (r.next()) {
               commodity.setId(r.getInt("ID"));
               commodity.setName(r.getString("NAME"));

     commodity.setDescription(r.getString("DESCRIPTION"));
               commodity.setPrice(r.getDouble("PRICE"));
               commodity.setImagePath(r.getString("IMAGEPATH"));
          }

          conn.close();
          return commodity;
     }

     public void ejbCreate() {
     }

     public void ejbActivate() throws EJBException,
     RemoteException {
     }

     public void ejbPassivate() throws EJBException,
     RemoteException {
     }

     public void ejbRemove() throws EJBException, RemoteException
     {
     }
```

```
    public void setSessionContext(SessionContext arg0) throws
    EJBException,
              RemoteException {
    try {
         InitialContext ctx = new InitialContext();
         if (dataSource == null)
              dataSource = (DataSource)
    ctx.lookup("java:/OracleDS");
    } catch (NamingException ex) {
         throw new EJBException(ex);
         }
    }
}
```

Задание:

Написать тестовое консольное приложение, для проверки работоспособности компонента `CommoditiesListBean`. Должен выполняться вызов обоих методов компонента, а результат выводиться в консоль.

Удаленный интерфейс *OrdersListBean*

В удаленном интерфейсе определим два метода:

- `List getOrdersForUser(String username) throws RemoteException, SQLException`. Этот метод возвращает список из объектов Order для пользователя с именем `username`.
- `List getCommodityOrdersForOrder(int orderId) throws RemoteException, SQLException`. Этот метод возвращает список из объектов `CommodityOrder` для заказа с ID = `orderId`.

```
public interface OrdersListRemote extends EJBObject {
    public List getOrdersForUser(String username) throws
RemoteException,
         SQLException;
    public List getCommodityOrdersForOrder(int orderId) throws
         RemoteException, SQLException;
}
```

Домашний интерфейс *OrdersListBean*

```
public interface OrdersListHome extends EJBHome {
    public OrdersListRemote create() throws RemoteException, CreateException;

}
```

Класс компонента *OrdersListBean*

Выполнение запроса к базе данных и формирование объектов `Order` и `CommodityOrder` выделено жирным шрифтом.

```
public class OrdersListBean implements SessionBean {

    private static final long serialVersionUID = 2767604097827994556L;

    private DataSource dataSource = null;

    public List getOrdersForUser(String username) throws SQLException {
        if (dataSource == null)
            throw new SQLException("Datasource not loaded");

        Connection conn = dataSource.getConnection();

        PreparedStatement p = conn.prepareStatement("SELECT * FROM " +
            " ORDERS WHERE USERNAME=?");
        p.setString(1, username);

        ResultSet r = p.executeQuery();
        List arrayList = new ArrayList();

        while(r.next()) {
            Order order = new Order();
            order.setId(r.getInt("ID"));
            order.setName(r.getString("USERNAME"));
            order.setDate(r.getDate("DATE_CREATED"));
            arrayList.add(order);
        }

        conn.close();
        return arrayList;
    }

    public List getCommodityOrdersForOrder(int orderId)
        throws SQLException {

        if (dataSource == null)
            throw new SQLException("Datasource not loaded");

        Connection conn = dataSource.getConnection();

        PreparedStatement p = conn.prepareStatement("SELECT * FROM " +
            " COMMODITY_ORDER WHERE ORDER_ID=?");
        p.setInt(1, orderId);
        ResultSet r = p.executeQuery();

        List arrayList = new ArrayList();

        while(r.next()) {
            CommodityOrder order = new CommodityOrder();
```

```
                    order.setId(r.getInt("ID"));
                    order.setCommodityId(r.getInt("COMMODITY_ID"));
                    order.setCommodityQuantity(r.getInt("QUANTITY"));
                    arrayList.add(order);
            }
            conn.close();
            return arrayList;
    }

    public void ejbCreate() {
    }

    public void ejbActivate() throws EJBException, RemoteException {
    }

    public void ejbPassivate() throws EJBException, RemoteException {
    }

    public void ejbRemove() throws EJBException, RemoteException {
    }

    public void setSessionContext(SessionContext arg0) throws EJBException,
                    RemoteException {
    try {
            InitialContext ctx = new InitialContext();
            if (dataSource == null)
                    dataSource = (DataSource) ctx.lookup("java:/OracleDS");
                    System.out.println("Data source aquired");
            } catch (NamingException ex) {
                    throw new EJBException(ex);
            }
    }

}
```

Задание:

Написать тестовое консольное приложение, для проверки работоспособности компонента `OrdersListBean`. Должен выполняться вызов обоих методов компонента, а результат выводиться в консоль.

Удаленный интерфейс *ShoppingBasketBean*

Интерфейс этого сеансового компонента с состоянием отражает те возможные опреации, которые пользователь может производить с корзиной. Эти методы перечислены далее:

- `public List getCurrentCommoditiesList() throws RemoteException`. Этот метод возвращает список из объектов CommodityOrder, соответствующих товарам в корзине.

- `public boolean isEmpty() throws RemoteException`. Этот метод возвращает булевское значение true, если корзина пуста.
- `public addCommodity(int commodityId) throws RemoteException`. Этот метод добавляет одну единицу товара с ID = `commodityId` в корзину.
- `public recalculateCommodity(int commodityId, int newCount) throws RemoteException`. Этот метод изменяет количество товаров с ID = `commodityId` в корзине на `newCount`.
- `public deleteCommodity(int commodityId) throws RemoteException`. Этот метод удаляет товар из корзины.
- `public void processOrder() throws RemoteException, SQLException`. Этот метод создает новый объект в таблице ORDERS базы данных, а затем добавляет записи с ORDER_ID, соответствущему только что добавленному заказу, в таблицу COMMODITY_ORDERS.

Как вы уже, наверное, успели убедиться, вызов каждого из этих методов может менять состояние компонента *ShoppingBasketBean*.

```
public interface ShoppingBasketRemote extends EJBObject {
     public List getCurrentCommoditiesList() throws
     RemoteException;
     public boolean isEmpty() throws RemoteException;
     public void addCommodity(int commodityId) throws
     RemoteException;
     public void recalculateCommodity(int commodityId, int
     newCount)
          throws RemoteException;
     public void deleteCommodity(int commodityId) throws
     RemoteException;
     public void processOrder() throws RemoteException,
     SQLException;
}
```

Домашний интерфейс *OrdersListBean*

Этот интерфейс отличается от домашних интерфейсов предыдущих компонентов. Компонент *ShoppingBasketBean* создается отдельно для каждого пользователя, поэтому удобно в него передать имя пользователя. Имя пользователя будет использоваться при выполнении запроса в методе `processOrder()`. В метод `create(String username)` домашнего интерфейса передается имя пользователя.

```
public interface ShoppingBasketHome extends EJBHome {
     public ShoppingBasketRemote create(String username) throws
          RemoteException, CreateException;
}
```

Класс компонента *ShoppingBasketBean*

Компонент *ShoppingBasketBean* должен постоянно поддерживать текущий список товаров. В классе компонента это делается посредством списка из объектов класса `CommodityOrder`. Все операции по добавлению, пересчету, и удалению товаров из списка текущих товаров в корзине сводятся к операциям поиска, удаления и добавления соответствующего объекта `CommodityOrder` в этом списке. Определен дополнительный вспомонательный метод `private CommodityOrder findOrderById(int commodityId)`. Он ищет соответствующий объект в списке и возвращает либо его, либо значение `null`, если он найден не был. Код метода, осуществляющий поиск по `commodityId` представлен далее.

```
Iterator iter = commodityOrderList.iterator();
CommodityOrder order = null;
while (iter.hasNext()) {
     CommodityOrder currentOrder = (CommodityOrder) iter.next();
     if (currentOrder.getCommodityId() == commodityId)  {
          order = currentOrder;
          break;
     }
}
return order;
```

Так как теперь компонент при создании должен получать имя пользователя, метод `public void ejbCreate(String username)` также несколько изменился. Теперь он сохраняет переданное имя пользователя во внутреннее поле `String username`, а также создает список, в котором в дальнейшем будет храниться текущий список товаров в корзине. Код этого метода представлен далее.

```
this.username = username;
commodityOrderList = new ArrayList();
```

В методе `public void processOrder() throws SQLException` выполняется создание заказа в базе данных по товарам, которые есть в корзине на данный момент. Если в корзине нет товаром, то этот метод не делает ничего. В противном случае он сначала создает объект в таблице ORDERS посредством следующего запроса:

```
PreparedStatement p = conn.prepareStatement("insert into "
     "ORDERS(DATE_CREATED, USERNAME) values (?, ?)");
p.setDate(1, new java.sql.Date(System.currentTimeMillis()));
p.setString(2, username);
p.executeUpdate();
```

Теперь необходимо получить ID этого объекта. В Интернете много рекомендаций на тему того, что это необходимо делать путем выборки максимального ID из таблицы в базе данных.

```
select max(ID) from ORDERS where USERNAME = 'BORIS'
```

Не используйте этот подход! Во-первых, он будет работать только тогда, когда генерируемый индекс получается путем увеличения предыдущего, а во-вторых, если один пользователь попробует оформить два заказа одновременно, то может быть выбран ID другого оформляемого в этот момент заказа, и список товаров далее будет добавлен не к тому заказу.

Поэтому в базе данных был создан пакет, в который посредством триггера сохранялось значение поля ID последнего добавленного объекта в таблицу ORDERS. Стоит отметить, что пакеты в СУБД *Oracle* сохраняют свое состояние для каждой сессии (от создания подключения до его закрытия), поэтому можно не опасаться, что в пакете окажется ID другого заказа.

```
p = conn.prepareStatement("select ORDER_ID_PKG.GET_LAST_ID from
dual");
ResultSet r = p.executeQuery();
int orderId = -1;
if (r.next()) {
     orderId = r.getInt(1);
}
```

То есть выбирается максимальное значение ID для текущего пользователя. Учитывая, что в нашем магазине не может быть создано двух корзин одновременно для одного пользователя, такое решение будет корректно работать.

Затем для каждого товара в корзине добавляется по записи в таблице COMMODITY_ORDER.

```
while (iter.hasNext()) {
     CommodityOrder currentOrder = (CommodityOrder) iter.next();

     p = conn.prepareStatement("insert into
COMMODITY_ORDER(ORDER_ID, " +
          "COMMODITY_ID, QUANTITY) values (?, ?, ?)");
     p.setInt(1, orderId);
     p.setInt(2, currentOrder.getCommodityId());
     p.setInt(3, currentOrder.getCommodityQuantity());
     p.execute();
}
```

Метод `void deleteCommdity(int commodityId)` сводится к вызову метода `recalculateCommodity(commodityId, 0)`.

Метод `void addCommodity(int commodityId)` вначале проверяет, есть ли товар с ID=`commodityId` в корзине, и либо добавляет новый объект `CommodityOrder` в список, либо просто увеличивает количество экземпляров в корзине товара на один.

```
CommodityOrder order = findOrderById(commodityId);
```

```
if (order == null) {
     order = new CommodityOrder();
     order.setCommodityId(commodityId);
     order.setCommodityQuantity(1);
     commodityOrderList.add(order);
} else {
     order.setCommodityQuantity(order.getCommodityQuantity() + 1);
}
```

Метод void recalculateCommodity(int commodityId, int newCount) изменяет количество экземпляров товара с ID=commodityId на newCount. Если newCount = 0, то товар удаляется из списка. Если товара не было в списке, то он туда добавляется.

```
CommodityOrder order = findOrderById(commodityId);

if (order == null) {
     if (newCount > 0) {
          order = new CommodityOrder();
          order.setCommodityId(commodityId);
          order.setCommodityQuantity(1);
          commodityOrderList.add(order);
     }
} else {
     if (newCount > 0) {
          order.setCommodityQuantity(newCount);
     } else {
          commodityOrderList.remove(order);
     }
}
```

Код класса компонента *ShoppingBasketBean* приводится далее.

```
public class ShoppingBasketBean implements SessionBean {

     private static final long serialVersionUID =
     8173942031447022589L;

     private String username = null;

     private DataSource dataSource = null;

     private List commodityOrderList;

     public void ejbCreate(String username) {
          this.username = username;
          commodityOrderList = new ArrayList();
     }

     private CommodityOrder findOrderById(int commodityId) {
          Iterator iter = commodityOrderList.iterator();
          CommodityOrder order = null;
          while (iter.hasNext()) {
```

```
            CommodityOrder currentOrder = (CommodityOrder)
            iter.next();
            if (currentOrder.getCommodityId() == commodityId)
            {
                order = currentOrder;
                break;
            }
        }
        return order;
    }

    public List getCurrentCommoditiesList() {
        return commodityOrderList;
    }

    public boolean isEmpty()  {
        return (commodityOrderList.size() == 0);
    }

    public void addCommodity(int commodityId) {
        CommodityOrder order = findOrderById(commodityId);

        if (order == null) {
            // Creating new order
            order = new CommodityOrder();
            order.setCommodityId(commodityId);
            order.setCommodityQuantity(1);
            commodityOrderList.add(order);
        } else {

    order.setCommodityQuantity(order.getCommodityQuantity() +
                1);
        }
    }

    public void recalculateCommodity(int commodityId, int
    newCount) {
        CommodityOrder order = findOrderById(commodityId);

        if (order == null) {
            if (newCount > 0) {
                // Creating new order
                order = new CommodityOrder();
                order.setCommodityId(commodityId);
                order.setCommodityQuantity(1);
                commodityOrderList.add(order);
            }
        } else {
            if (newCount > 0) {
                order.setCommodityQuantity(newCount);
            } else {
                commodityOrderList.remove(order);
            }
```

```
        }
    }

    public void deleteCommodity(int commodityId) {
        recalculateCommodity(commodityId, 0);
    }

    public void processOrder() throws SQLException {
        if (dataSource == null)
            throw new SQLException("Datasource not loaded");
        Connection conn = dataSource.getConnection();

        if (commodityOrderList.size() == 0)
            return;

        PreparedStatement p = conn.prepareStatement("insert
        into " +  "ORDERS(DATE_CREATED, USERNAME) " +
            "values (?, ?)");

        p.setDate(1, new
        java.sql.Date(System.currentTimeMillis()));
        p.setString(2, username);
        p.executeUpdate();

        p = conn.prepareStatement("select max(ID) from ORDERS
        where " + "USERNAME = ?");
        p.setString(1, username);

        ResultSet r = p.executeQuery();

        int orderId = -1;
        if (r.next()) {
            orderId = r.getInt(1);
        }

        Iterator iter = commodityOrderList.iterator();
        while (iter.hasNext()) {
            CommodityOrder currentOrder = (CommodityOrder)
        iter.next();

        p = conn.prepareStatement("insert into
        COMMODITY_ORDER(" + " ORDER_ID, COMMODITY_ID, QUANTITY)
        " + "values (?, ?, ?)");

            p.setInt(1, orderId);
            p.setInt(2, currentOrder.getCommodityId());
            p.setInt(3, currentOrder.getCommodityQuantity());
            p.execute();
        }

        commodityOrderList.clear();
        conn.close();
    }
```

```
    public void ejbActivate() throws EJBException,
    RemoteException {
    }

    public void ejbPassivate() throws EJBException,
    RemoteException {
    }

    public void ejbRemove() throws EJBException, RemoteException
    {
    }

    public void setSessionContext(SessionContext arg0) throws
    EJBException,  RemoteException {
         try {
              InitialContext ctx = new InitialContext();
              if (dataSource == null)
                   dataSource = (DataSource)
                        ctx.lookup("java:/OracleDS");
         System.out.println("Data source aquired");
         } catch (NamingException ex) {
              throw new EJBException(ex);
         }
    }

}
```

Задание:

Написать тестовый клиент для *ShoppingBasketBean.* Он должен создавать один экземпляр компонента и делать тестовый заказ.