Оглавление

[**1** **Работа в сети** 1](#_Toc490665451)

[1.1 Пример 1 URL 2](#_Toc490665452)

[1.2 Пример 2 UDP 3](#_Toc490665453)

[1.3 Пример 3 TCP/IP 4](#_Toc490665454)

[1.4 TCP/IP сервер с множеством подключений 6](#_Toc490665455)

[1.5 TCP/IP прерываемые сокеты на клиенте SocketChannel. 7](#_Toc490665456)

[1.6 Пример 4 RMI 8](#_Toc490665457)

[1.7 Отправка, используя формат HTTP-запроса «**multipart»**. 10](#_Toc490665458)

[1.8 Отправка, используя формат HTTP-запроса «**Json».** 11](#_Toc490665459)

[**2** **Сервлет** 11](#_Toc490665460)

[**3** **EJB + сервлет** 13](#_Toc490665461)

[3.1 Создание библиотеки классов. 13](#_Toc490665462)

[3.2 Создание EE-приложения (корпоративного) и модуля ejb. 13](#_Toc490665463)

[3.3 Создание компонента сеанса 14](#_Toc490665464)

[3.4 Добавление бизнес-метода в сеансовый компонент ejb 16](#_Toc490665465)

[3.5 Развертывание корпоративного приложения 16](#_Toc490665466)

[3.6 Создание клиентского приложения уровня предприятия. 17](#_Toc490665467)

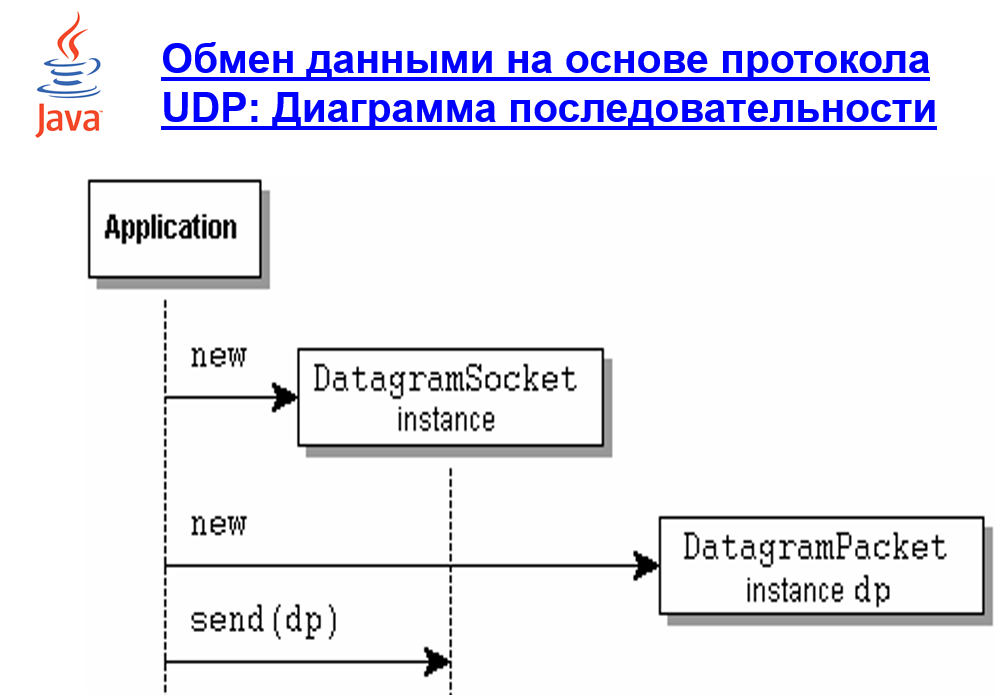
[**4** **JDBC** 19](#_Toc490665468)

[4.1 Пример 1 19](#_Toc490665469)

[4.2 Предварительная подготовка команды 20](#_Toc490665470)

# **Работа в сети**





TCP:

* Задать тайм-аут сокетов:

|  |
| --- |
| Socket s = **new** Socket(**"localhost"**,80); s.setSoTimeout(5000); |
| Можно и так:  Socket s = **new** Socket(); **try** {  s.connect(**new** InetSocketAddress(**"localhost"**, 8075), 5000); }**catch**(Exception e){  System.***err***.println(**"Exception: "**+e); } |

Тайм-аут выбрасывает исключение SocketTimeoutException

* Получить IP по имени:

|  |
| --- |
| InetAddress adr = InetAddress.*getByName*(**"jurfed.ru"**); System.***out***.println(adr);  System.***out***.println(adr.getHostAddress()); |
| jurfed.ru/81.177.174.9  81.177.174.9 |

Получить все IP – адреса:

InetAddress[] addresses = InetAddress.*getAllByName*(host);

## Пример 1 URL

N:\semestr4\razrab\_raspred\_priloz\_java\lab1:

|  |
| --- |
| *//1. На основе класса URL написать и отладить приложение, выводящее на консоль текст страницы, адрес которой //передается в качестве параметра вызова приложения.* **public class** Test {   **public static void** main(String[] args) **throws** MalformedURLException, IOException {  URL url = **new** URL(**"http://jurfed.ru/"**);  InputStream input = url.openStream();  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(input));  String line = **""**;  String finishLines = **""**;  **while** ((line = br.readLine()) != **null**) {  finishLines += line + **"\n"**;  }  br.close();  input.close();   System.***out***.println(finishLines);  } } |

## Пример 2 UDP

**UDP** N:\semestr4\razrab\_raspred\_priloz\_java\lab1:

Сериалимзовать класс Demo и отправить серверу. Сервером прочитать данный класс, сделать его новый экземпляр и отправить обратно клиенту.

Главные классы: **DatagramSocket** и **DatagramPacket**

|  |
| --- |
| **public class** Demo **implements** Serializable {   **private int last**;  **private int id**;  **private** String **name**;  **private** String **message**;  **private** Date **date**;  **private transient int temp**;   **public** Demo() {  **last** = 1;  **id** = 1;  **name** = **"Server class"**;  **message** = **"Hello? I'm server"**;  **date** = **new** Date();  **temp** = 34;  }   **public** String toString() {  String msg = getClass().getCanonicalName() + **"\n"**;  **return** msg + **"last="** + **last** + **" id="** + **id** + **" name="** + **name** + **" message="** + **message** + **" date="** + **date**;   } } |
| **public class** Client {   **public static void** main(String[] args) **throws** IOException, ClassNotFoundException {  Demo inDemo = **new** Demo(**"From Client"**); *// инициализировать сокет дейтаграмм* DatagramSocket socket = **new** DatagramSocket();  *// послать запрос* InetAddress address = InetAddress.*getByName*(**"localhost"**);  *//----------------------- сформировать запрос с клиентским объектом Demo серверу и отправить его:* ByteArrayOutputStream bos = **new** ByteArrayOutputStream();  ObjectOutput out = **new** ObjectOutputStream(bos);  out.writeObject(inDemo);  **byte**[] yourBytes = bos.toByteArray();  *//отправить серверу* DatagramPacket packet = **new** DatagramPacket(yourBytes, yourBytes.**length**, address, 4445);  socket.send(packet);  *//-----------------------* socket.send(packet);  *// получить ответ от сервера с серверным объектом Demo* **byte**[] buf = **new byte**[1000];  packet = **new** DatagramPacket(buf, buf.**length**);  socket.receive(packet);  *// вывести ответ от сервера  //десериализация* ByteArrayInputStream bis = **new** ByteArrayInputStream(buf);  ObjectInput in = **new** ObjectInputStream(bis);  inDemo= (Demo) in.readObject();  System.***out***.println(**"Client:\n"**+inDemo);  socket.close();  } } |
| **package** demo;   **import** java.io.\*;  **import** java.net.DatagramPacket;  **import** java.net.DatagramSocket;  **import** java.net.InetAddress;  **public class** Server {  */\*\*  \** ***@param args*** *the command line arguments  \*/* **public static void** main(String[] args) **throws** IOException, ClassNotFoundException, InterruptedException {  Server s = **new** Server();  }   **public** Server() **throws** IOException, ClassNotFoundException, InterruptedException {  *//указываем порт* DatagramSocket socket = **new** DatagramSocket(4445);  **try** { *// получить запрос от клиента и записать в buf* **byte**[] buf = **new byte**[256];  DatagramPacket packet = **new** DatagramPacket(buf, buf.**length**);  socket.receive(packet); *//------------------------------Прочитать полученные данных от клиента и десериализовать их* ObjectInput in = **new** ObjectInputStream(**new** ByteArrayInputStream(buf));  *//дессириализуем класс Demo* Demo inDemo = (Demo) in.readObject();  System.***out***.println(**"Server:\n"**+inDemo);  *//--------------------------------* Thread.*sleep*(5000); *// Сформировать ответ и отправить клиенту на полученный адрес и порт* Demo serverDemo = **new** Demo(**"From Server"**);  ByteArrayOutputStream bos = **new** ByteArrayOutputStream();  ObjectOutput out = **new** ObjectOutputStream(bos);  out.writeObject(serverDemo);  **byte**[] myBytes = bos.toByteArray();  InetAddress address = packet.getAddress();  packet = **new** DatagramPacket(myBytes, myBytes.**length**, address, packet.getPort());  socket.send(packet);  } **catch** (IOException e) {  e.printStackTrace();  }  socket.close();  } } |

## Пример 3 TCP/IP

**TCP/IP** N:\semestr4\razrab\_raspred\_priloz\_java\lab2:

Последовательно послать серверу приложения строки «date» и «time». Сервер в ответ на первый запрос должен вернуть текущую дату, на второй текущее время.

Главные классы:

* + **Socket**
  + **ServerSocket**

|  |
| --- |
| **На сервере:**  ServerSocket s = **new** ServerSocket(8189); *// wait for client connection* Socket incoming = s.accept();  InputStream inStream = incoming.getInputStream();//входной поток OutputStream outStream = incoming.getOutputStream();//выходной поток  Scanner in = **new** Scanner(inStream); PrintWriter out = **new** PrintWriter(outStream, **true** */\* autoFlush \*/*); |
| **На клиенте:**  *//Создать экземпляр класса Socket* Socket **soc** = **new** Socket(**"localhost"**,8030);  *//ссылка на выходной поток класса socket* PrintStream **ps** = **new** PrintStream(**soc**.getOutputStream()); *//ссылка на входной поток класса socket* BufferedReader **br** = **new** BufferedReader(**new** InputStreamReader(**soc**.getInputStream())); |

|  |
| --- |
| **Клиент:**  **public class** TCP2 {   **public static void** main(String[] args) **throws** IOException {  *//Создать экземпляр класса Socket* Socket soc = **new** Socket(**"localhost"**, 8030);   *//ссылка на выходной поток класса socket* PrintStream ps = **new** PrintStream(soc.getOutputStream());  *//ссылка на входной поток класса socket* BufferedReader br = **new** BufferedReader(**new** InputStreamReader(soc.getInputStream()));   **while** (**true**) {  *//проимзвести запись в выходной поток и отправить данные* ps.println(**"date"**);  ps.flush();   *//Прочитать данные из входного потока* String msg = br.readLine();  System.***out***.println(msg);   *//проимзвести запись в выходной поток и отправить данные* ps.println(**"time"**);  ps.flush();   *//Прочитать данные из входного потока* msg = br.readLine();  System.***out***.println(msg);   *//Закрыть входной, выходной потоки и сокета* soc.close();  ps.close();  br.close();   System.*exit*(0);  }   } } |
| **Сервер:**  **public class** TPC1 { **public static void** main(String[] args) **throws** IOException, InterruptedException {   *//Создать экземпляр класса ServerSocket* ServerSocket server = **new** ServerSocket(8030);  *//Получить ссылку на экземпляр класса Socket* Socket s = server.accept();   **while** (**true**) {  *//ссылка на выходной поток класса* PrintStream ps = **new** PrintStream(s.getOutputStream());  *//ссылка на входной поток класса* BufferedReader br = **new** BufferedReader(**new** InputStreamReader(s.getInputStream()));   String msg = br.readLine();*//читаем сообщение от клиента* **if** (msg == **null**) {*//закрываем соединение, если сообщение от ликета пустое* ps.close();  s.close();  **return**;  }  System.***out***.println(msg);    GregorianCalendar gc = **new** GregorianCalendar(TimeZone.*getTimeZone*(**"GMT+4"**));  *//если от клиента получили date* **if** (msg.equals(**"date"**)) {  SimpleDateFormat sdf = **new** SimpleDateFormat(**"dd.MM.yyyy"**);  String responce = sdf.format(gc.getTime());  ps.println(responce);  ps.flush();*//отрпавляем данные клиенту* }  Thread.*sleep*(500);   msg = br.readLine();  System.***out***.println(msg);    **if** (msg.equals(**"time"**)) {*//если от клиента получили time* SimpleDateFormat sdf = **new** SimpleDateFormat(**"hh:mm"**);  String responce = sdf.format(gc.getTime());  ps.println(responce);  ps.flush();*//отрпавляем данные клиенту* }    }  } } |

## TCP/IP сервер с множеством подключений

Запуск ожиданий подключений в отдельных потоках

|  |
| --- |
| **public class** ThreadedEchoServer {  **public static void** main(String[] args) **throws** IOException {   **int** i = 1;  ServerSocket s = **new** ServerSocket(8189);   **while** (**true**) {  **Socket** incoming = s.accept();  System.***out***.println(**"Spawning "** + i++);  **Runnable** r = **new** ThreadedEchoHandler(incoming);  **Thread** t = **new** Thread(r);  t.**start**();  }  } }  **class** ThreadedEchoHandler **implements** Runnable {  **private** Socket **incoming**;   **public** ThreadedEchoHandler(Socket i) {  **incoming** = i;  }   **public void** run() {  **try** {  **try** {  InputStream inStream = **incoming**.getInputStream();  OutputStream outStream = **incoming**.getOutputStream();   Scanner in = **new** Scanner(inStream);  PrintWriter out = **new** PrintWriter(outStream, **true** */\* autoFlush \*/*);   out.println(**"Hello! Enter BYE to exit."**);   *// echo client input* **boolean** done = **false**;  **while** (!done && in.hasNextLine()) {  String line = in.nextLine();  out.println(**"Echo: "** + line);  **if** (line.trim().equals(**"BYE"**))  done = **true**;  }  } **finally** {  **incoming**.close();  }  } **catch** (IOException e) {  e.printStackTrace();  }  } } |

## TCP/IP прерываемые сокеты на клиенте SocketChannel.

**SocketChannel** channel = SocketChannel.*open*(**new InetSocketAddress**(**"localhost"**, 8189));

Пример: для прерывания нужно нажать на Cancel

|  |
| --- |
| */\*\*  \* клиент, устанавливающий прерываемое соединение с сервером  \*/* **public void** connectInterruptibly() **throws** IOException {  **messages**.append(**"Interruptible:\n"**);  **SocketChannel** channel = **SocketChannel.*open***(**new InetSocketAddress**(**"localhost"**, 8189));  **try** {  **in** = **new** Scanner(channel);  **while** (!Thread.*currentThread*().isInterrupted()) {*//пока поток не прерван* **messages**.append(**"Reading "**);  **if** (**in**.hasNextLine()) {*//чтение ответа от сервера* String line = **in**.nextLine();  **messages**.append(line);  **messages**.append(**"\n"**);  }  }  } **finally** {  channel.close();  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  **messages**.append(**"Channel closed\n"**);  **interruptibleButton**.setEnabled(**true**);  }  });  } } |
| */\*\*  \* Серверный класс, обрабатывающий данные, полученные от 1-ого клиента.  \*/* **class** TestServerHandler **implements** Runnable {  **private** Socket **incoming**;  **private int counter**;   **public** TestServerHandler(Socket i) {  **incoming** = i;  }   **public void** run() {  **try** {  OutputStream outStream = **incoming**.getOutputStream();  PrintWriter out = **new** PrintWriter(outStream, **true** */\* autoFlush \*/*);  **while** (**counter** < 100) {  **counter**++;  **if** (**counter** <= 10) out.println(**counter**);  Thread.*sleep*(200);  }  **incoming**.close();  **messages**.append(**"Closing server\n"**);  } **catch** (Exception e) {  **messages**.append(**"\nTestServerHandler.run: "** + e);  }  } } |

Весь пример:



## Пример 4 RMI

**RMI N:\semestr4\razrab\_raspred\_priloz\_java\lab2**

Позволяет делать программы, распределенными между несколькими JVM.

Доступ к удаленному объекту осуществляется через переменную-ссылку на объект, указывающую на реализацию замещающей заглушки удаленного интерфейса. Ссылка на объект содержит сетевой адрес машины и порт (конечную точку) сервера.

|  |
| --- |
| 1. Интерфейс должен прямо или косвенно расширять интерфейс java.rmi.Remote  2. В сигнатуре каждого метода должно быть указание на возможность генерации исключения RemoteException |

**RMI: запуск приложения:**

1. Запуск реестра RMI :

* В среде Unix:

**rmiregistry &**

* В среде Windows:

**start rmiregistry**

2. Запуск сервера

3. Запуск клиента

|  |
| --- |
| **public interface** DateTime **extends** Remote {   String getDate () **throws** RemoteException;  String getTime () **throws** RemoteException;  **void** myclose() **throws** RemoteException;  } |
| **public class** MyServer **implements** DateTime {   **static** Registry *registry*;  **static** Remote *stub*;  **public static** String *servName* = **"MyDateTimeService"**;   **public** Object close(String name) {  **return ""**;  }   **public static void** main(String... args) **throws** Exception {   *registry* = LocateRegistry.*createRegistry*(2099); *//регистрация в реестре с указанием порта* DateTime service = **new** MyServer();  *stub* = UnicastRemoteObject.*exportObject*(service, 0);*//экспорт удаленного объекта и получаем заглушку (2-ой параметр -порт)  registry*.bind(*servName*, *stub*);*//помещение удаленного объекта в реестре с именем MyDateTimeService* System.***out***.println(**"OK..."**); *// synchronized (stub) { // stub.wait(); // UnicastRemoteObject.unexportObject(stub, true);//удаление заглушки // } // System.out.println("End.");* }   @Override  **public** String getDate() **throws** RemoteException {*//отправка даты* GregorianCalendar gc = **new** GregorianCalendar(TimeZone.*getTimeZone*(**"GMT+4"**));  SimpleDateFormat sdf = **new** SimpleDateFormat(**"dd.MM.yyyy"**);  String responce = sdf.format(gc.getTime());  **return** responce;  }   @Override  **public** String getTime() **throws** RemoteException {*//отправка времени* GregorianCalendar gc = **new** GregorianCalendar(TimeZone.*getTimeZone*(**"UTC+4"**));  SimpleDateFormat sdf = **new** SimpleDateFormat(**"dd.MM.yyyy"**);   sdf = **new** SimpleDateFormat(**"hh:mm"**);  String responce = sdf.format(gc.getTime());  **return** responce;  }   @Override  **public void** myclose() **throws** RemoteException {  **try** {  *registry*.unbind(*servName*);*//удаление из рееста удаленного объекта* } **catch** (NotBoundException ex) {  Logger.*getLogger*(MyServer.**class**.getName()).log(Level.***SEVERE***, **null**, ex);  }  System.***out***.println(**"End close"**);  } } |
| **public class** MyClient {   **public static void** main(String... args) **throws** Exception {  *//Находим удалённый регистратор и запрашиваем у него объект-заглушку, связанную с именем "MyDateTimeService"* Registry registry = LocateRegistry.getRegistry(**"localhost"**, 2099);*//получение удаленного объекта* DateTime service = (DateTime) registry.lookup(**"MyDateTimeService"**);   System.***out***.println(**"Send to server"**);  System.***out***.println(service.getDate());*//вызываем удаленный метод на сервере* Thread.*sleep*(2000);  System.***out***.println(service.getTime());*//вызываем другой удаленный метод на сервере* System.***out***.println(**"Enter 1 for exit"**);   service.myclose();  *// registry.unbind("MyDateTimeService");* System.***out***.println(**"End."**);  } } |

## Отправка, используя формат HTTP-запроса «**multipart»**.

Для отправки нужна apache – библиотекой «httpclient»( N:\dplm\programms\httpcomponents-client-4.5.2-bin). Библиотеки указываем в build.gradle, в разделе **dependencies:**

|  |
| --- |
| version **'1.0-SNAPSHOT'** apply **plugin**: **'java'** sourceCompatibility = 1.8  repositories {  mavenCentral() }  dependencies { **'N:\\подготовка\\java\\horstmann\\CoreJavaBook\\v1ch10\\AppletApplication\\lib'**) } |

|  |
| --- |
| **public class** Test {   **public static void** main(String args[]){  *//формируем http - клиента без проверки подписи сервера* CloseableHttpClient **httpclient** = HttpClients.*custom*().  setHostnameVerifier(**new** AllowAllHostnameVerifier()).  setSslcontext(**new** SSLContextBuilder().loadTrustMaterial(**null**, **new** TrustStrategy() {  **public boolean** isTrusted(X509Certificate[] arg0, String arg1) **throws** CertificateException {  **return true**;  }  }).build()).build();   HttpPost **httppost** = **new** HttpPost(**"https://jurfed.ru/planets"**);   MultipartEntityBuilder **builder** = MultipartEntityBuilder.*create*();  builder.setMode(HttpMultipartMode.***BROWSER\_COMPATIBLE***);  builder.addTextBody(**"clientId"**, **"138860177375"**);  FileBody fileBody = **new** FileBody(**new** File(**"C:\\reestrs\\test2.csv.gz"**));  builder.addPart(**"file"**, fileBody);  httppost.setEntity(builder.build());  HttpResponse **response** = httpclient.execute(httppost);*//отправка запроса* HttpEntity **resEntity** = response.getEntity();  String responseString = EntityUtils.*toString*(resEntity, **"UTF-8"**);*//запись ответа от сервера в строку resEntity* } } |

## Отправка, используя формат HTTP-запроса «**Json».**

Для отправки нужна apache – библиотекой «httpclient»

|  |
| --- |
| StringEntity params = **new** StringEntity(**"{param1:values1,"** + **"\""** + **"{param2:values2,"** + **"\""** + **"{param2:values2,"** + **"\"} "**);  HttpPost httppost = **new** HttpPost(“https://jurfed.ru/planets”); httppost.addHeader(**"content-type"**, **"application/json"**); httppost.setEntity(params); HttpResponse response = httpclient.execute(httppost); HttpEntity resEntity = response.getEntity(); String responseString = EntityUtils.*toString*(resEntity, **"UTF-8"**); httpclient.close(); |

# **Сервлет**

**Сервлет (Servlet)** — Java-класс, который на стороне сервера динамически обрабатывает запрос клиента и формирует на него ответ.

Для обеспечения работы по протоколу HTTP определены следующие основные классы и интерфейсы :

**HttpServlet**

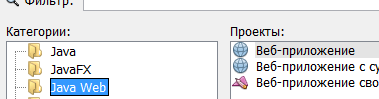
**ServletConfig**

**ServletContext**

**HttpServletRequest**

**HttpServletResponse**

Создаем web – приложение:



Настраиваем index.html для того, чтобы использовался метод get: 

Пример:

|  |
| --- |
| **package** servler;  **import** java.io.IOException; **import** java.io.PrintWriter; **import** java.util.Random; **import** java.util.regex.Matcher; **import** java.util.regex.Pattern; **import** javax.servlet.ServletException; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **public class** lab3Servler **extends** HttpServlet {   String **hello**, **answer**;  Random **rand**;   @Override  **public void** init() {*//блок инициализации. Вызывается первым* **rand** = **new** Random();  **hello** = **"Мой сервлет"**;  }  @Override  **protected void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  *//настрйока кодировки* request.setCharacterEncoding(**"Cp1251"**);  response.setContentType(**"text/html;charset=Cp1251"**);   PrintWriter out = response.getWriter();  out.println(**"<html><head>"**);  out.println(**"<title>hello</title></head><body>"**);  *//указание метода отправки следующего request-запроса* out.println(**"<form method=\"post\" action=\""** + request.getContextPath() + **"/lab3Servler\">"**);  out.println(**"<table border=\"1\"><tr><td align=\"top\">"**);*//таблица для элементов с толщиной 1* out.println(**"Enter the String</td> <td valign=\"top\">"**);  out.println(**"<input type=\"text\" name=\"textPole\" size=\"20\">"**);*//поле для ввода текста с именем "textPole"* out.println(**"</td></tr><tr><td valign=\"top\">"**);*//переход в следующую строку в таблице* out.println(**"<input type=\"submit\" value=\"Отправить\"></td></tr>"**);*//кнопка отправки* out.println(**"</table></form>"**);  out.println(**"</body></html>"**);  } @Override  **protected void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {   request.setCharacterEncoding(**"Cp1251"**);  response.setContentType(**"text/html;charset=Cp1251"**);  PrintWriter out = response.getWriter();   out.println(**"<html><head>"**);  out.println(**"<title>Отправленные параметры</title></head><body>"**);   String parName = request.getParameter(**"textPole"**);*//получаем параметр, введенный пользователем* **if** ((parName.trim()) == **""**) {  response.setContentType(**"text/html;charset=Cp1251"**);  out = response.getWriter();  out.println(**"<html><head>"**);  out.println(**"<title>hello</title></head><body>"**);  out.println(**"<form method=\"post\" action=\""** + request.getContextPath() + **"/lab3Servler\">"**);  out.println(**"<table border=\"0\"><tr><td align=\"top\">"**);  out.println(**"Enter the String</td> <td valign=\"top\">"**);  out.println(**"<input type=\"text\" name=\"textPole\" size=\"20\">"**);  out.println(**"</td></tr><tr><td valign=\"top\">"**);  out.println(**"<input type=\"submit\" value=\"Отправить\"></td></tr>"**);  out.println(**"</table></form>"**);  out.println(**"</body></html>"**);  **answer** = **"Please enter the string againg"**;  } **else if** (checkString(parName)) {  **rand** = **new** Random();  **int** i = Integer.*parseInt*(parName);  **answer** = **""** + (i + (**rand**.nextInt(100) + 1));  } **else** {  **final** String WORD = **"[a-zA-Zа-яА-Я0-9]+"**;   **int** i = 0;  Pattern pattern = Pattern.*compile*(WORD);  CharSequence externalText;  Matcher matcher = pattern.matcher(parName);*//указываем свой текст* **while** (matcher.find()) {  i++;  }  out.println(**"<strong>"** + **"Original string: "** + parName + **"</strong>"**);  out.println(**"<br />"**);  **answer** = **" total words: "** + i;  }   out.println(**"<strong>"** + **answer** + **"</strong>"**);  out.println(**"<br />"**);  out.println(**"</body></html>"**);  }   *//является ли строка числом* **public boolean** checkString(String string) {  **try** {  Integer.*parseInt*(string.trim());  } **catch** (Exception e) {  **return false**;  }  **return true**;  } } |

Весь проект:



# **EJB + сервлет**

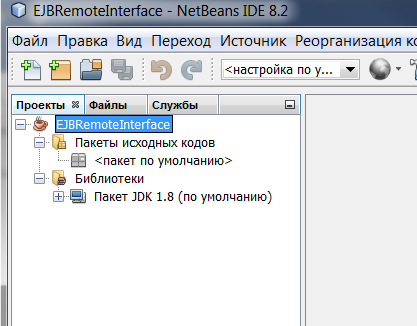
**EJB**(Enterprise Java Beans) — это фреймворк для построение бизнес-логики приложения.

**Session beans**— используется для построения бизнес-логики, которая может быть вызвана программным клиентом

## Создание библиотеки классов.

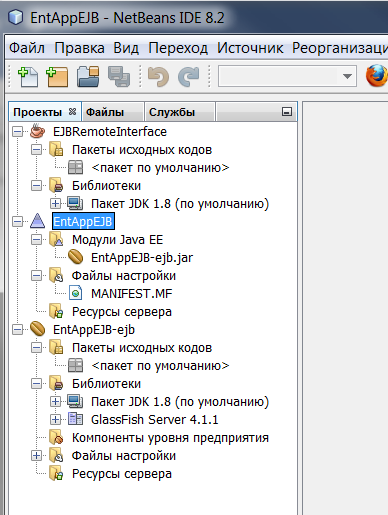
Библиотека классов содержит интерфейс для удаленного доступа к ejb. В дальнейшем эта библиотека в виде jar – архива рассылается клиентам для доступа к ejb. Также в дальнейшем в самом компоненте ejb будет использоваться этот же jar – архив для реализации этого интерфейса.

1. "Файл > Новый проект" затем "Библиотека классов Java" в категории "Java". Нажмите кнопку "Далее", введите **EJBRemoteInterface** в качестве имени проекта, нажмите кнопку "Завершить".



## Создание EE-приложения (корпоративного) и модуля ejb.

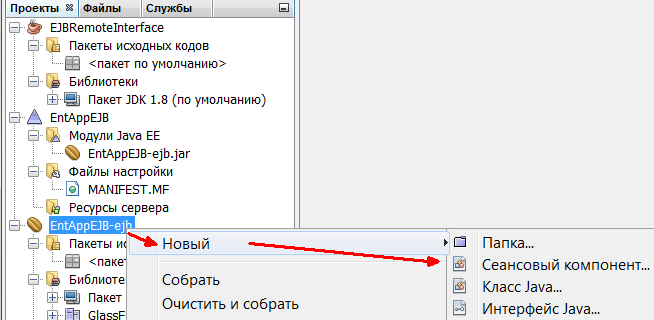
1. Выберите "Файл" > "Создать проект" > Enterprise Application ("Корпоративное приложение") в категории Java EE. Нажмите кнопку "Далее". Введите **EntAppEJB** как имя проекта. Нажмите кнопку "Далее".Выберите GlassFish Server в качестве сервера. Убедитесь, что выбрано Create EJB Module ("Создать модуль EJB") и отмените выбор Create Web Application ("Создать модуль веб-приложения"). Нажмите кнопку "Завершить".



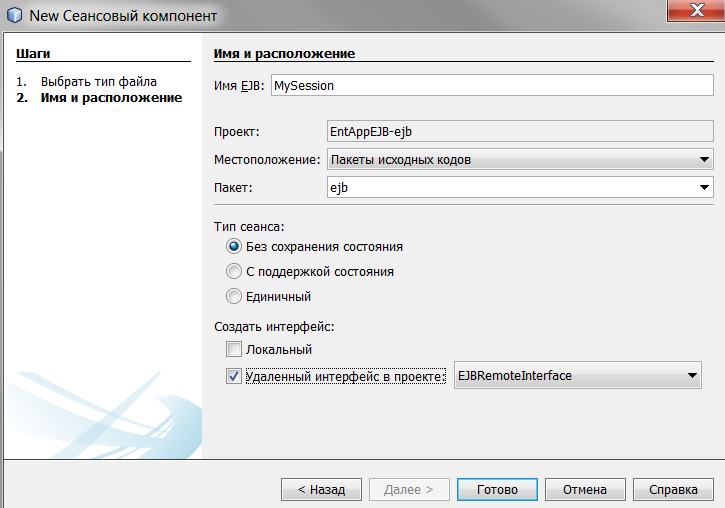
Теперь в окне проектов перечислены три типа проектов: библиотека классов, корпоративное приложение и модуль EJB.

## Создание компонента сеанса

Выбрать проект модуля ejb и создать сеансовый компонент.

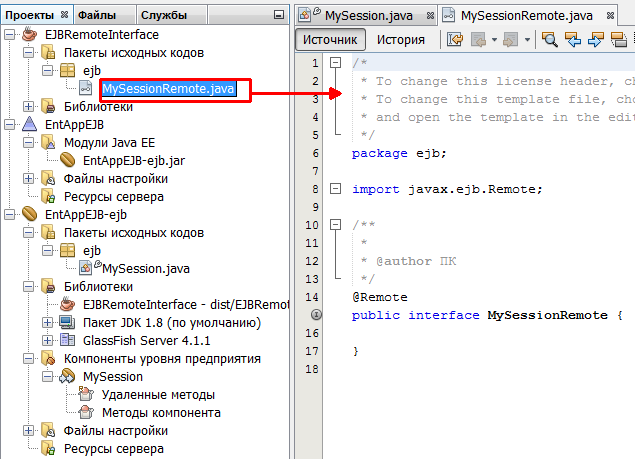


Далее заполнить параметры: ввести имя сеансового компонента MySession, указать пакет ejb, выбрать «без поддержки состояния», выбрать создание удаленного интерфейса с указанием созданного ранее проекта библиотеки классов(библиотека пока ничего не содержит):

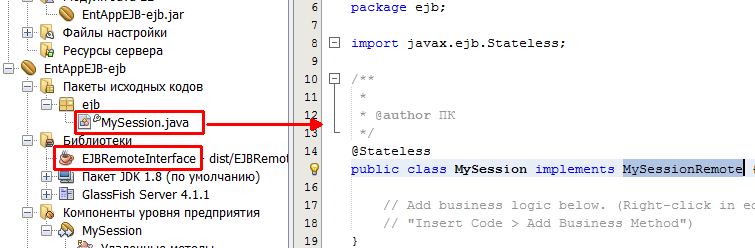


При этом будет создано следующее:

* Удаленный интерфейс под названием MySessionRemote в пакете ejb проекта библиотеки классов EJBRemoteInterface:

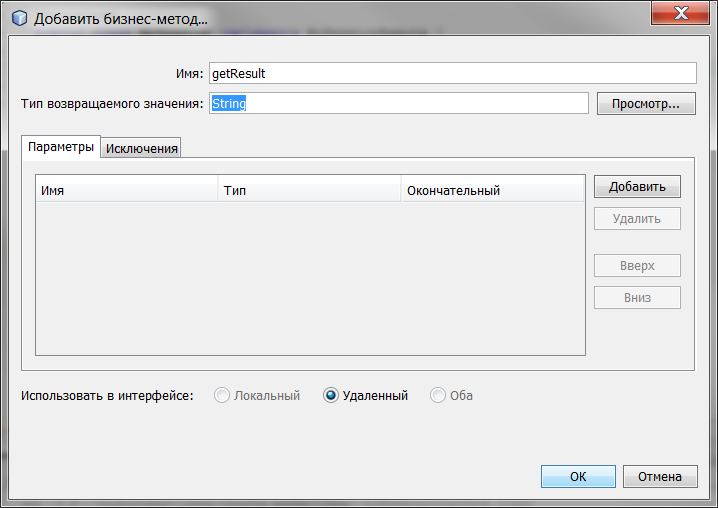


* Класс – «сеансовый компонент» (MySession) в пакете ejb модуля EJB, реализующий интерфейс MySessionRemote.  и файл JAR EJBRemoteInterface (проект-библиотека классов с интерфейсом) был добавлен как библиотека модуля EJB. **Stateful** может хранить какое-либо состояние между запросами, соответственно контейнер имеет пул таких бинов и для каждой сессии выделяет отдельный экземпляр. **Stateless** никакого состояния не хранит, а следовательно может использоваться один и тот же экземпляр. Вроде же все понятно.



## Добавление бизнес-метода в сеансовый компонент ejb

1. Щелкните правой кнопкой мыши в тексте класса MySession и выберите Insert Code ("Вставить код"), затем выберите Add Business Method ("Добавить бизнес-метод"). Введите **getResult** в поле Method Name ("Имя метода") и String ("Строка") в поле Return Type ("Тип возврата"). Нажмите кнопку "ОК".



При этом и в интерфейс MySessionRemote и в класс MySession будет добавлен данный метод.

1. Внесите нижеприведенные изменения, чтобы метод getResult возвращал строку.

@Stateless

public class MySession implements MySessionRemote {

public String getResult() {

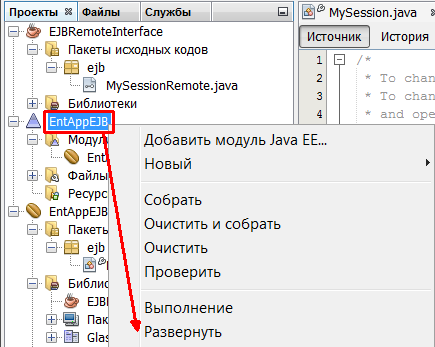
return **"This is My Session Bean"**;

}

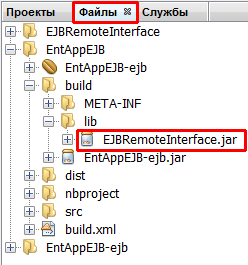
}

## Развертывание корпоративного приложения

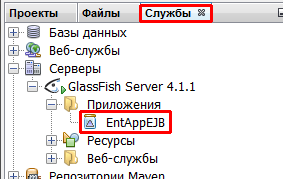
1. Щелкните правой кнопкой мыши корпоративное приложение EntAppEJB и выберите Deploy ("Развернуть"). Среда IDE развернет архив EAR на сервере.



Если взглянуть в окно Files ("Файлы"), можно заметить, что файл JAR EJBRemoteInterface развернут вместе с приложением:



Если развернуть узел 'Приложения' сервера GlassFish в окне 'Службы', можно увидеть, что компонент EntAppEJB развернут:



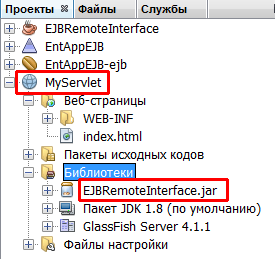
## Создание клиентского приложения уровня предприятия.

При создании клиента приложения проекту нужна библиотека классов Java EJBRemoteInterface в качестве библиотеки, чтобы установить ссылку на EJB.

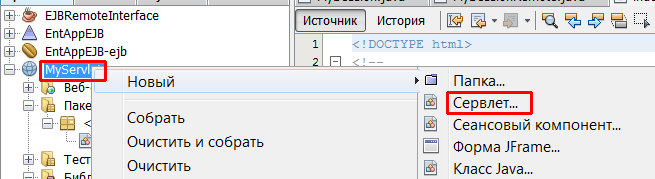
Создаем сервлет:

|  |  |
| --- | --- |
|  |  |
| 3 |  |

Добавить библиотеку классов к проекту:



Далее создаем класс-сервлет:



Импортируем нашу библиотеку, указываем аннотацию @EJB, добавляем кнопку в веб-интерфейс «out.println("<input type=\"submit\" value=\"Отправить\"></td></tr>");»

|  |
| --- |
| **public class** FirstServlet **extends** HttpServlet {  @EJB  **private** MySessionRemote **mySession**;  String [] **str**;  String **parName**=**""**, **parName2**=**""**, **parName3**=**""**, **parName4**=**""**;   **public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** IOException {  request.setCharacterEncoding(**"Cp1251"**);  response.setContentType(**"text/html;charset=Cp1251"**);   PrintWriter out = response.getWriter();  out.println(**"<html><head>"**);  out.println(**"<title>Мой сервлет</title></head><body>"**);  out.println(**"<form method=\"post\" action=\""** + request.getContextPath() + **"/FirstServlet\">"**);  out.println(**"<table border=\"0\"><tr><td align=\"top\">"**);   out.println(**"Enter you name</td> <td valign=\"top\">"**);  out.println(**"<input type=\"text\" name=\"you\_name\" size=\"20\">"**);  out.println(**"</td></tr><tr><td valign=\"top\">"**);    out.println(**"<input type=\"submit\" value=\"Отправить\"></td></tr>"**);  out.println(**"</table></form>"**);  out.println(**"</body></html>"**);  }   @Override  **public void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** IOException {  request.setCharacterEncoding(**"Cp1251"**);  response.setContentType(**"text/html;charset=Cp1251"**);   PrintWriter out = response.getWriter();  String str = request.getParameter(**"you\_name"**);*//записать введенное имя в переменную* String str2 = **mySession**.getResult(str);*//послать введенное имя в на сервер и получить ответ.* out.println(**"<html><head>"**);  out.println(**"<title>Мой сервлет</title></head><body>"**);  out.println(**"<form method=\"post\" action=\""** + request.getContextPath() + **"/FirstServlet\">"**);  out.println(**"<table border=\"0\"><tr><td align=\"top\">"**);  out.println(**"Enter you name</td> <td valign=\"top\">"**);  out.println(**"</td></tr><tr><td valign=\"top\">"**);  out.println(**"<input type=\"text\" name=\"you\_name\" size=\"20\">"**);  out.println(**"</td></tr><tr><td valign=\"top\">"**);  out.println(str2+**"</td> <td valign=\"top\">"**);*//вывести ответ от сервера* out.println(**"<input type=\"submit\" value=\"Отправить\"></td></tr>"**);  out.println(**"</table></form>"**);  out.println(**"</body></html>"**);  } } |

Правим Index.html

|  |
| --- |
| <html>  <head>  <title>TODO supply a title</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  </head>  <body>  <form action="FirstServlet" method="GET">  <h1>Welcome!</h1>  <input type="submit" value="Start"/>  </form>  </body>  </html> |

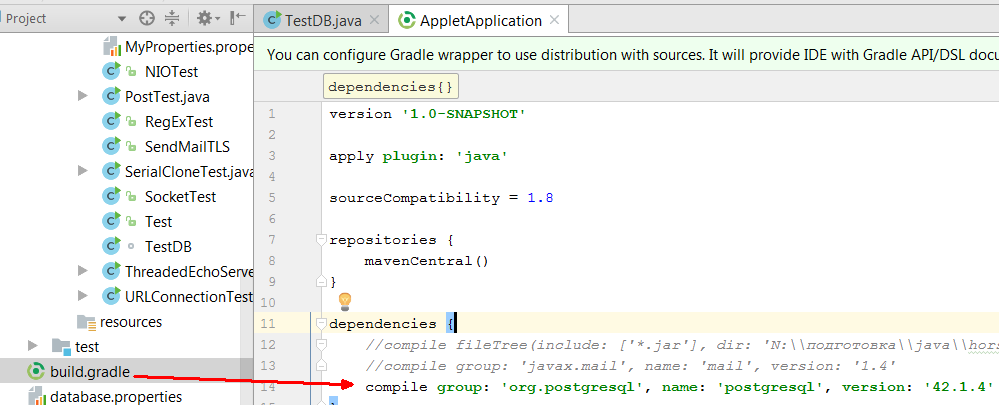
Весь проект:



# **JDBC**

## Пример 1

1. Указываем драйвер:



1. Создаем файл с настройками подключения «database.properties»:

|  |
| --- |
| url=jdbc:postgresql://localhost:5432/postgres  username=postgres  password=123456 |

1. Код:

|  |
| --- |
| **class** TestDB {  **public static void** main(String args[]) **throws** IOException, SQLException {  Properties props = **new** Properties();*//объект с настройками*  InputStream in = **new** FileInputStream(**"database.properties"**);  props.load(in);  in.close();   String url = props.getProperty(**"url"**);  String username = props.getProperty(**"username"**);  String password = props.getProperty(**"password"**);   **Connection conn = DriverManager.*getConnection***(url, username, password);   **Statement stat = conn.createStatement**();   **stat.executeUpdate**(**"CREATE TABLE Greetings (Message CHAR(20))"**);  stat.executeUpdate(**"INSERT INTO Greetings VALUES ('Hello, World!')"**);   **ResultSet result = stat.executeQuery**(**"SELECT \* FROM Greetings"**);  **if** (**result.next**())  System.***out***.println(**result.getString**(1));*//результат по номеру колонки* System.***out***.println(result.getString(**"Message"**));*//результат по имени колонки* result.**close**();  stat.executeUpdate(**"DROP TABLE Greetings"**);  } } |



## Предварительная подготовка команды

|  |
| --- |
| **class** TestDB {  **public static void** main(String args[]) **throws** IOException, SQLException {  Properties props = **new** Properties();*//объект с настройками* InputStream in = **new** FileInputStream(**"database.properties"**);  props.load(in);  in.close();   String url = props.getProperty(**"url"**);  String username = props.getProperty(**"username"**);  String password = props.getProperty(**"password"**);   Connection conn = DriverManager.*getConnection*(url, username, password);   String str = **"select books.price, books.title from books where Publisher\_Id=? and ISBN=?"**;  **PreparedStatement prepare = conn.prepareStatement(str);**   **prepare**.setString(1, **"07434"**);  **prepare**.setString(2, **"0-7434-1146-3"**);   ResultSet result = **prepare.executeQuery()**;  **if** (result.next()) {  System.***out***.println(result.getString(1) + **" "** + result.getString(2));  }    result.close();  } } |

## Навигация вперед/назад и обновление данных.

***TYPE\_SCROLL\_INSENSITIVE*** – с прокруткой, но без учета изменений в бд

***CONCUR\_UPDATABLE*** – с редактированием и обновлением БД

Другие параметр в томе 2 стр 265

Нужно обязательно запрашивать первичный ключ, чтобы далее выполнить обновление даже другого столбца.

|  |
| --- |
| String query = **"SELECT isbn, price FROM Books"**;**Statement stat = conn.createStatement(ResultSet.*TYPE\_SCROLL\_INSENSITIVE*,ResultSet.*CONCUR\_UPDATABLE*);**  ResultSet rs=stat.executeQuery(query); **while**(rs.next()){  System.***out***.println(rs.getDouble(**"price"**));  **if** (rs.getDouble(**"price"**)==70.0){  **rs.updateDouble**(**"price"**,77.77);   **rs.updateRow(); - непосредственно обновляет строку в бд**  } } **rs.previous(); rs.previous();** System.***out***.println(rs.getDouble(**"price"**)); |