

Работа в сети

Пакет `java.net`

Адреса в сети

- Классы
 - URI (uniform resource *identifier*)
 - URL (uniform resource *locator*)
 - InetAddress

InetAddress

```
InetAddress address = InetAddress.getLocalHost();
System.out.println("1-"+address );
InetAddress addrList[ ] = InetAddress.getAllByName("yandex.ru");
for (InetAddress a: addrList){
    System.out.println("2-"+a);
}
InetAddress addr = InetAddress.getByName(null);
System.out.println("3-"+address );
addr=InetAddress.getByName("localhost");
System.out.println("4-"+address );
addr= InetAddress.getByName("127.0.0.1");
System.out.println("5-"+address );
```

Исключение

UnknownHostException

URI и URL

- A URI is a uniform resource *identifier* while a URL is a uniform resource *locator*

URL

Для того, чтобы извлечь реальную информацию, адресуемую данным URL, необходимо на основе URL создать объект `URLConnection`, воспользовавшись для этого методом `openConnection()`

URL

```
URL hp = new URL("http://edu.mmcs.sfedu.ru/course/view.php?id=356");
URLConnection hpCon = hp.openConnection();
System.out.println("Date : " + hpCon.getDate ());
System.out.println("Type : " + hpCon.getContentType ());
System.out.println("Length: " + hpCon.getContentLength());
System.out.println( "HeaderField: " + hpCon.getHeaderField(0));
```

Исключения

MalformedURLException

IOException

URI

//Задание 4 (HashSet && TreeSet)

URI lk=

```
new URI("https://edu.mmcs.sfedu.ru/mod/assign/view.php?id=26670");  
System.out.println(lk.getHost());  
System.out.println(lk.getPath());  
System.out.println(lk.getQuery());  
System.out.println(lk.getScheme());
```

Исключение
URISyntaxException

Сокеты

- Класс Socket
 - Socket (String host, int port)
 - InputStream getInputStream()
 - OutputStream getOutputStream()
- Информация о текущем соединении
 - getInetAddress()
 - getPort()
 - getLocalPort()

Пример

```
try {
    Socket s = new Socket(" time-a.timefreq.bldrdoc.gov ", 13);
    // сервер, который поддерживает протокол Daytime
    try {
        InputStream inStream = s.getInputStream();
        Scanner in = new Scanner(inStream);
        while (in.hasNextLine()) {
            String line = in.nextLine();
            System.out.println(line);           // 61174 26-05-14 19:41:44 50 0 0 903.1 UTC(NIST) *
        }
    } finally { s.close(); }
} catch (IOException e) {System.out.println("нет доступа к ресурсу");
}
```

Сокеты

- Класс `ServerSocket`

`ServerSocket(int port)`

`ServerSocket(int port, int count)`

Объект `ServerSocket` ожидает, пока к нему кто-нибудь не подключится.

При подключении он создает объект `Socket`

```
Socket socket = s.accept();
```

После подключения идет взаимодействие «сокет-сокет»

Клиент и сервер равноправны

Пример (EchoServer)

```
try(ServerSocket s = new ServerSocket(8189);  
{  
    System.out.println("Server start") ;  
    // wait for client connection  
    Socket incoming = s.accept();  
    try {  
        InputStream inStream =  
            incoming.getInputStream();  
        OutputStream outputStream =  
            incoming.getOutputStream();  
        Scanner in = new Scanner(inStream);  
        PrintWriter out =  
            new PrintWriter(outputStream, true );
```

```
        out.println("Hello! Enter BYE to exit.");  
        // echo client input  
        boolean done = false;  
        while (!done && in.hasNextLine()) {  
            String line = in.nextLine();  
            System.out.println("client tell-"+line);  
            out.println("Echo: " + line);  
            if (line.trim().equals("BYE")) done = true;  
        }  
    } finally { incoming.close(); }  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Пример

Проверяем соединение через putty

Пример клиента – EchoClient

Пробуем доступ с другого компьютера

Пробуем присоединиться несколькими клиентами к одному серверу

Взаимодействие в отдельном потоке

```
class ServerToOne extends Thread {  
    //для одного клиента  
    private Socket socket;  
    private Scanner in;  
    private PrintWriter out;  
    public ServeToOne(Socket s) throws IOException {  
        socket = s;  
        in = ...  
        out = ...  
        start(); // Вызывает run()  
    }  
}
```

ЧТО ВЫПОЛНИТЬ В ПОТОКЕ

```
public void run() {  
    try {  
        boolean done=true;  
        while (done) {  
            String str = in.readLine();  
            if (str.equals("BYE")) {  
                out.println("close");  
                done=false;  
                Thread.currentThread().interrupt();  
            }  
            else{  
                System.out.println("Echoing: " + str);  
                out.println("ECHO:"+str);  
            }  
        }  
    }  
}
```

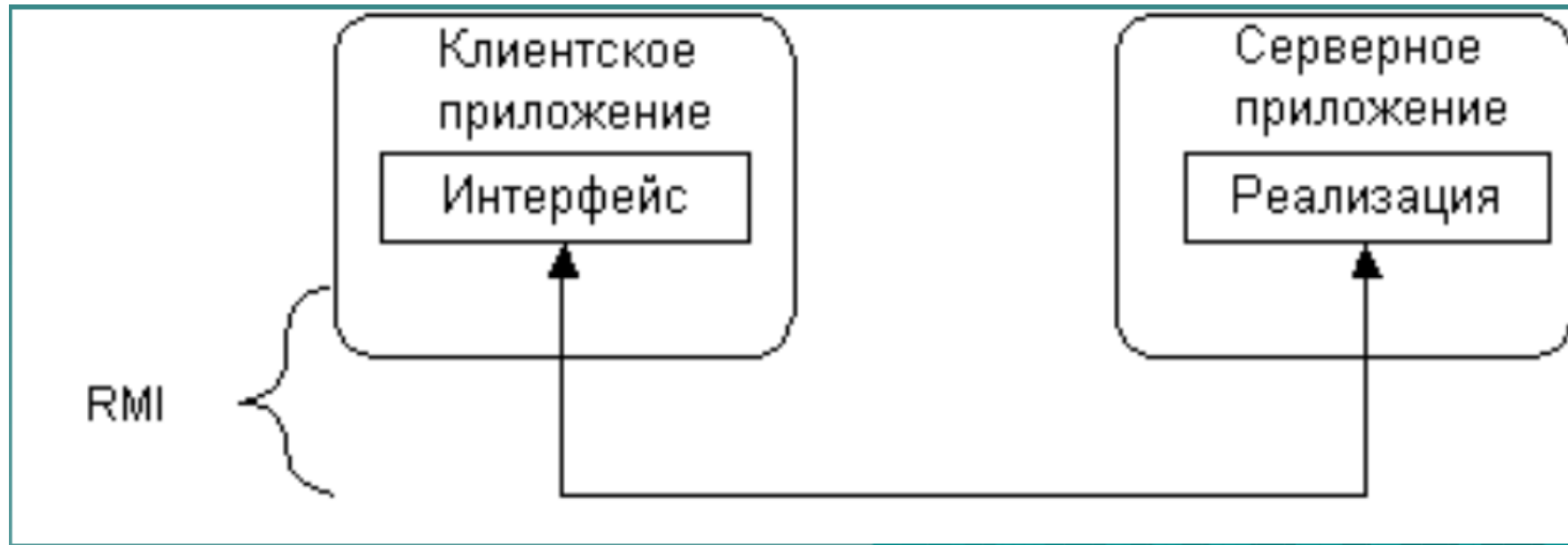
```
} catch(IOException e) {  
    System.err.println("IO Exception");  
} finally {  
    try {  
        System.out.println("closing...");  
        socket.close();  
    } catch(IOException e) {  
        System.err.println("Socket not closed");}  
    }  
}}
```

Многопоточный сервер

```
public class MultiServer {  
    static final int PORT = 8189;  
    public static void main(String[] args)  
        throws IOException {  
        ServerSocket s = new ServerSocket(PORT);  
        System.out.println("Server Started");  
        try {  
            while(true) {  
                // ждем подключение  
                Socket socket = s.accept();
```

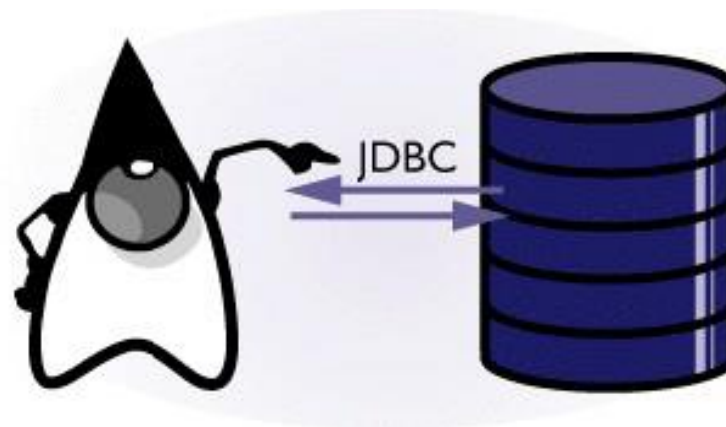
```
        try {  
            new ServerToOne (socket);  
        } catch(IOException e) {  
            // Если неудача - закрываем сокет,  
            // в противном случае нить закроет его:  
            socket.close();  
        }  
    } finally {  
        s.close();  
    }  
}
```

RMI



Работа с базами данных

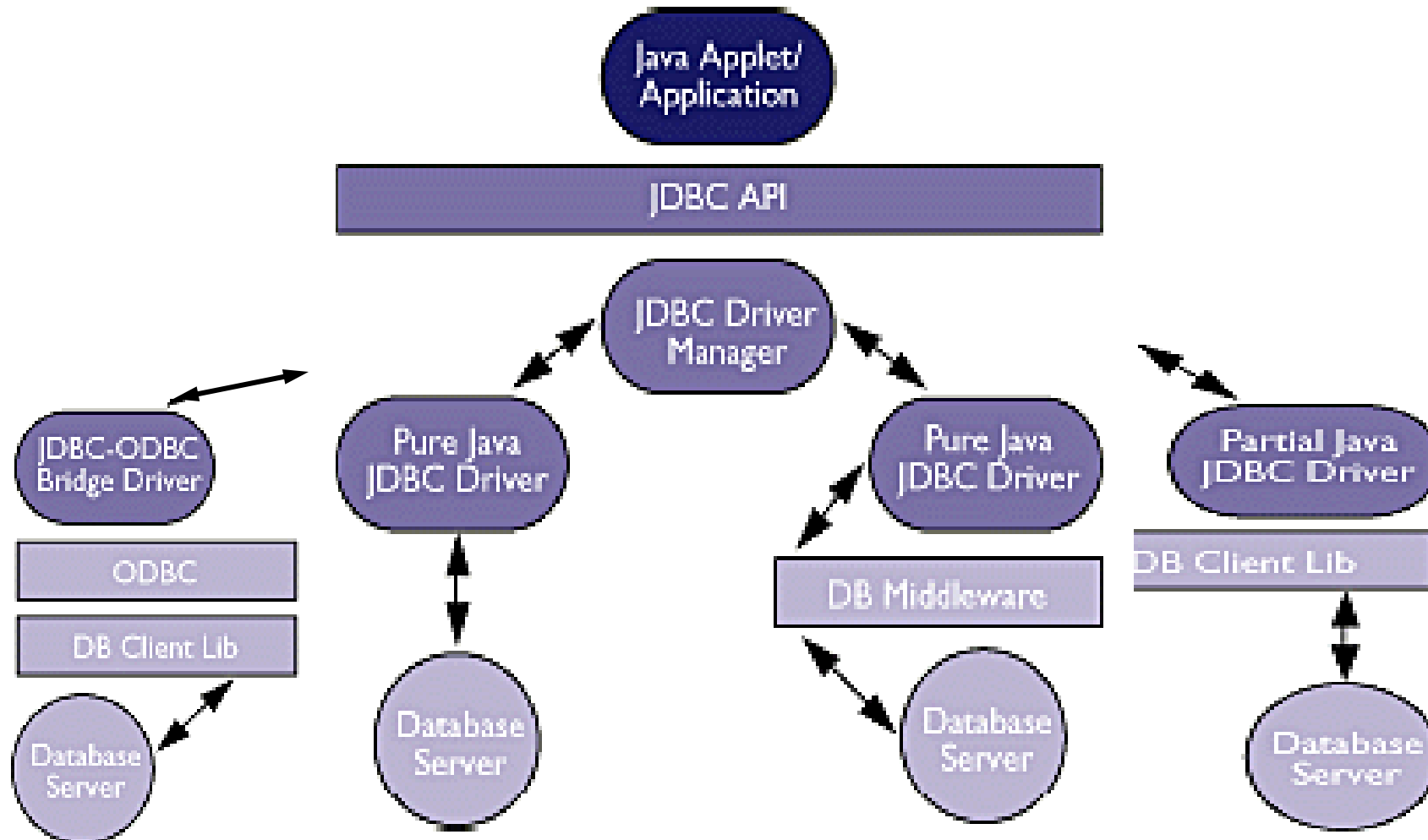
JDBC



JDBC

- JDBC, также как и Microsoft ODBC и Borland DataBase Engine (BDE), базируется на X/Open SQL CLI (Call Level Interface).
- Авторы спецификации обращают особое внимание на то, что их основная задача состоит в описании основных абстракций и концепций, определенных в X/Open CLI, в виде натуральных ("родных") интерфейсов Java.

Архитектура JDBC



Основы взаимодействия Java и DB

- JDBC предполагает передачу запроса к базе данных в виде строки
- Базовым требованием JDBC, является удовлетворение входного уровня ANSI-стандарта SQL-92

Основные интерфейсы

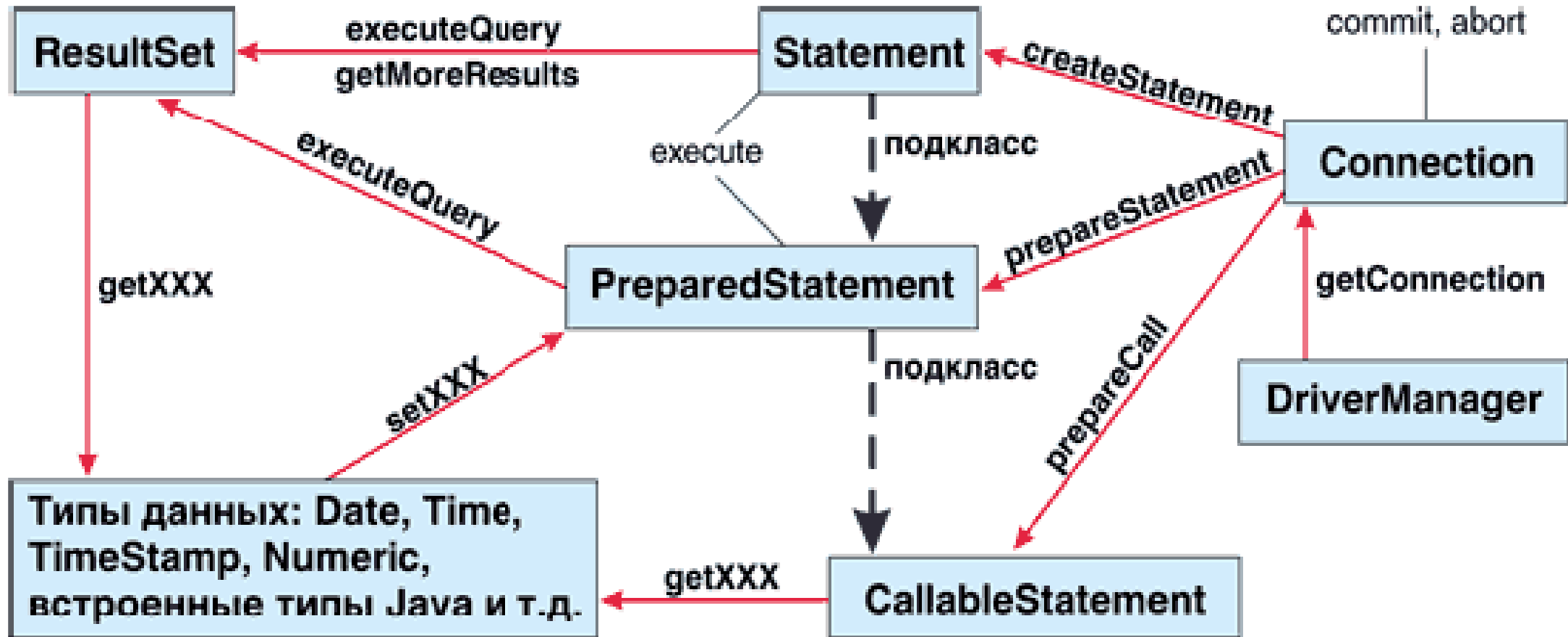


Схема работы с JDBC

- загрузить драйвер в память виртуальной машины;
- получить соединение (Connection) с базой данных;
- подготовить выражение-запрос (Statement);
- выполнить запрос;
- разобрать полученные данные (ResultSet).

JDBC драйверы поставляются в виде библиотек – архивов

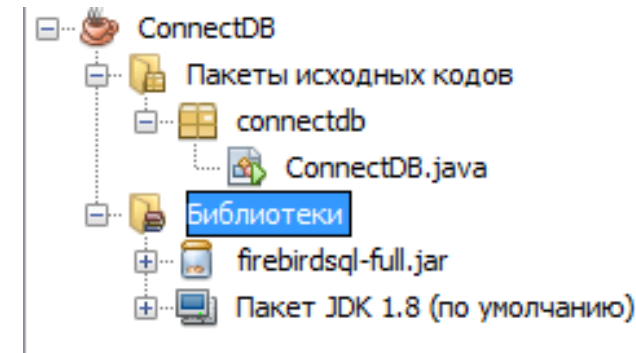
Например:

pg74jdbc3.jar

firebirdsql-full.jar

Доступ к библиотеке драйвера

- Изменить значение переменной CLASSPATH
set classpath=.; C:\Drivers\firebirdsql-full.jar
- Поместить архив-файл в каталог jre/lib/ext
- Поместить архив в каталог проекта



- При запуске приложения, использующего JDBC драйвер указать путь к библиотеке драйвера

```
java -classpath .; C:\Drivers\firebirdsql-full.jar <имя класса>
```

Регистрация драйвера

- Класс DriverManager отвечает за выбор нужного драйвера базы данных и создание нового соединения с базой данных
- Классы драйверов обычно имеют имена вида
org.postgresql.Driver
org.firebirdsql.jdbc.FBDriver

Регистрация драйвера

- Системное свойство `jdbc.drivers` содержит имена классов для тех драйверов, которые диспетчер должен зарегистрировать при запуске
- Имена драйверов можно задать в командной строке
`java -Djdbc.drivers=org.firebirdsql.jdbc.FBDriver <класс>`
- Приложение может установить системное свойство
`System.setProperty("jdbc.drivers", "org.firebirdsql.jdbc.FBDriver")`
- Драйвер можно зарегистрировать явно, загрузив его класс
`Class.forName("org.firebirdsql.jdbc.FBDriver");`

```
public class ConnectDB {  
    public static void main(String[] args) {  
        String driverName = "org.firebirdsql.jdbc.FBDriver";  
        try{  
            Class.forName(driverName);  
            System.out.println("OK !!!!");  
        }catch(ClassNotFoundException e){  
            System.out.println("Fireberd JDBC driver not found");  
        }  
        finally{  
            System.out.println("Exit");  
            return;  
        }  
    }  
}
```

Соединение с БД

- Получение соединения с конкретной базой данных возложено на менеджер драйверов. Его метод `getConnection()` принимает ссылку на БД, имя пользователя и его пароль:

```
Connection con =  
    DriverManager.getConnection(url, "Login", "Password");
```

В ответ на такое обращение менеджер драйверов возвратит ссылку на экземпляр класса `Connection`, представляющий собой соединение с базой данных.

Если в процессе соединения необходимо передать дополнительные параметры, можно использовать вариант метода `getConnection()`, в котором все параметры соединения, включая имя пользователя и пароль передаются через коллекцию свойств

```
Properties props = new Properties();  
Connection c = DriverManager.getConnection(databaseURL, props);
```

Формат URL для соединения

jdbc: название_подпротокола: другие_сведения

Например

`jdbc:firebirdsql:localhost/3050:c:/DB/employee.fdb`

Добавляем описания (1)

```
String databaseURL =  
    "jdbc:firebirdsql:servername/3050:/path/employee.fdb";  
String user = "LOGIN";  
String password = "password";  
Connection c = null;
```

Добавляем операторы в блок try (2)

```
c = DriverManager.getConnection(databaseURL,user,password);  
System.out.println("Connect");
```

Добавляем обработку ошибки (3)

```
}catch(SQLException e){  
    System.out.println("SQLException" +e.getMessage());  
}
```

Добавляем финализацию (4)

```
try{ if (c!=null) c.close();  
} catch(SQLException e){System.out.println("Error!!!");}
```

Выполнение статических запросов

Statement - интерфейс предназначен для разового выполнения статических запросов

Добавляем

(1) `Statement s = null;`

(2) `s=c.createStatement();`

(4) `try{ if (s!=null) s.close();} catch(SQLException e){}`

Выполнение статических запросов

ResultSet - интерфейс для получения результата в виде множества данных

```
(1) ResultSet rs =null;
```

```
(2) rs = s.executeQuery(  
    "select full_name from employee where salary<50000");
```

```
(4) try{  
    if (rs!=null) rs.close();  
} catch(SQLException e){. . .}
```

Результаты запроса

Класс результата - ResultSet

Методы для получения возвращаемых результатов:

```
boolean next()  
String getString(int column)  
String getString(String name)  
и т.д. для всех типов данных
```

```
while (rs.next()) {  
    System.out.println(rs.getString("full_name"));  
}
```

Если не знаем запрос

```
String query ;  
// получили запрос  
rs = s.executeQuery( query);  
ResultSetMetaData rsM=rs.getMetaData();  
System.out.println("columns="+rsM.getColumnCount() );  
for (int i=0; i< rsM.getColumnCount(); i++){  
    System.out.print(rsM.getColumnName(i+1)+" - "  
        +rsM.getColumnTypeName(i+1) + " | ");  
}
```

```
while (rs.next()) {  
    for (int i=0; i< rsM.getColumnCount(); i++)  
        System.out.print(rs.getString(i+1)+" | ");  
    System.out.println();  
}
```

Или сделать переключатель, в зависимости от типа колонки и читать данные методами

getString()

getDouble()

getInt()

getDate()

...

```
select dept_no, full_name, salary  
  from employee  
 where salary<50000
```

При сборке проекта в jar-файл проверить в настройках, чтобы проект включал зависимые библиотеки

