Тема 10. Взаимодействие на Java с базите от данни (продължение)

Съдържание

- Използване на Java Class за търсене на драйвер;
- Взаимодействие на езика с базата от данни;
- Настройка на връзката с базата от данни;
- Получаване на адреса на данновия източник (DSN)

Използване на Java Class за търсене на драйвер

Класовете в Java по време на изпълнение се представят от класа java.lang.Class;

- За всеки обект съществува обект от класа **Class, който може да се получи по време на изпълнение;**
- Обект от този клас позволява:
  - от референция да се определи класа;
  - да се получат неговите методи и други свойства;
  - да се определи структурата на обекта;
  - да се изпълнят неговите методи по време на изпълнение.

Процесът на получаване на информация по време на изпълнение се нарича "**рефлексия**"

Използване на Java Class за търсене на драйвер

Примерен клас за илюстрация на рефлексия:

```
import java.lang.reflect.Field;
public class Reflection {
    public int intField = 1;
    public String strField = "String value";
    public Reflection() {     }
```

(продължава)

Използване на Java Class за търсене на драйвер

```
public static void main(String[] args) {
// Търсене на клас по име. forName () е статичен метод,
// който връща обект от класа Class по дадено име като String
    try {
      Reflection objReflection = new Reflection();
      Class classReflection = objReflection.getClass();
      Class.forName("Reflection");
      // Class.forName("ReflectionBad");
      Field[] fields = classReflection.getFields( );
      for(int i=0; i<fields.length; i++) {
            System.out.println("field : "+fields[i].toString());
      }//field : public int Reflection.intField
    }
    catch ( ClassNotFoundException e ) {
// не може да намери класа
          System.out.println("No found class : "+e.toString());
// No found class : java.lang.ClassNotFoundException: ReflectionBad
        }
     }
}
```

Взаимодействие на езика с базата от данни

Основните стъпки при взаимодействието се изразяват в следното:

- Инициализация на базата;
  - Проверка за наличие на драйвер (**рефлексия**);
  - Установяване на връзка (предоставен от клас DriverManager);
  - Установяване на параметри (при необходимост).
- Изпълнение на заявки;
- Затваряне на базата;

Взаимодействие на езика с базата от данни

- **Инициализация на базата**

Създава се връзка с базата от данни.

Пакети:

java.sql.Connection - за връзката с базата

java.sql.DriverManager-за драйверите към различни бази

java.sql.PreparedStatement-за заявките

java.sql.SQLException-за изключенията

java.sql.ResultSet-за резултантното множество

Пример на рамков код, с който се прави достъпа към базата от данни посредством драйвер JDBC от съответната библиотека на Java:

Взаимодействие на езика с базата от данни

- **Инициализация на базата:**

Пример на рамков код, с който се прави достъпа към базата от данни посредством драйвер JDBC от съответната библиотека на Java. Основни класове (обекти):

```
// обект от клас Connection - представя връзката
Connection db;
// статичен низ за името на данновия източник
// пример: "jdbc:odbc:testDSN" – следва пояснение
static String url = < име на данновия източник >
// класове за командите за създаване и манипулация на
// данни-три класа за представяне, следва пояснение
PreparedStatement statement; //Statement, CallableStatement
```

Взаимодействие на езика с базата от данни
(Класове за командите за създаване и манипулация на данни)

Три класа за представяне:

- **Statement;**
- **PreparedStatement ;**
- **CallableStatement**

Взаимодействие на езика с базата от данни
(Класове за командите за създаване и манипулация на данни)

Statement-основни характеристики и приложение:

- Използва се за изпълнение на константни (непараметризирани) SQL команди.
- Не могат да се предават параметри към SQL заявката по време на изпълнение - чрез променливи.
- Предимство при изпълняване на конкретна SQL заявка еднократно, обикновено за DDL команди, напр. CREATE , ALTER , DROP и др.
- Производителността е по-ниска в сравнение с другите два интерфейса-няма предварителна подготовка-парсиране, компилация, компресиране…

Взаимодействие на езика с базата от данни

(Класове за командите за създаване и манипулация на данни)

Statement-пример:

```
//създаване на референция
 Statement stmt = con.createStatement();
//изпълнение
stmt.executeUpdate("CREATE TABLE StudentRegister(
StudentFN CHAR(8) NOT NULL PRIMARY KEY,
NAME CHAR(30) NOT NULL,
PersonSex CHAR(3) NOT NULL,
PersonTitle CHAR(10) NOT NULL)“);
```

Взаимодействие на езика с базата от данни
(Класове за командите за създаване и манипулация на данни)

PreparedStatement -основни характеристики и приложение:

- Използва се за изпълнение на динамични или параметризирани SQL команди.
- Могат да се предават параметри към SQL заявката по време на изпълнение - чрез променливи.
- Предимство при многократно изпълняване на SQL команди, обикновено за DML команди, напр. SELECT, UPDATE , DELETE и др.
- Производителността е основана на предварителна подготовка (еднократно)-парсиране, компилация…

Взаимодействие на езика с базата от данни
(Класове за командите за създаване и манипулация на данни)

PreparedStatement-пример:

```
//създаване на референция
 PreparedStatement pstmt = con.prepareStatement("update StudentRegister set NAME = ? where StudentFN = ?");
//свързване на данни
pstmt.setString(1, "MyName");   // "MyName" в първи холдер
pstmt.setString (2, "61662223");     // " 61662223 "  във втори
//изпълнение
pstmt.executeUpdate();
```

Взаимодействие на езика с базата от данни
(Класове за командите за създаване и манипулация на данни)

CallableStatement -основни характеристики и приложение:

- Използва се за изпълнение на съхранените процедури.
- Могат да се предават три типа параметри на съхранени процедури - чрез променливи:
  - IN - използва се за предаване на стойностите на съхранена процедура;
  - OUT - използва се за съхраняване на резултата, върнат от съхранената процедура,
  - IN OUT - действа като IN и OUT параметър.
- Предимство най-висока производителност (сървърни) съхранени процедури-парсиране, компилация…

Взаимодействие на езика с базата от данни
(Класове за командите за създаване и манипулация на данни)

CallableStatement-пример:

```
//създаване на референция
 CallableStatement cstmt = con.prepareCall("{call anyProcedure(?, ?, ?)}");
// cstmt.setter() предава към IN параметри
// cstmt.registerOutParameter() регистрира OUT параметри
//изпълнение
cstmt.execute();
//cstmt.getter() за получаване на резултата, върнат от съхранената процедура
```

Взаимодействие на езика с базата от данни
(Инициализация на базата-имплементация)

Основни действия-алгоритъм,пример:

- Зареждане на драйвера;
- Получаване на връзката с базата от данни;
- Подготовка на механизъма за съхранение;
- (Незадължително) създаване на референция към обекти за командите-вижте предните обяснения.
  - Алтернативата е тези референции да се създават и унищожават в локален обхват;
  - Предимства-следва пример?
  - Недостатъци и избор?

Взаимодействие на езика с базата от данни

Инициализация на базата (примерен код)

```
void initDB() {
    try {
        // зареждане на драйвера JDBC/ODBC вариант
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        // зареждане на драйвера ORACLE вариант
        // Class.forName ("oracle.jdbc.driver.OracleDriver");
        // получаване на връзката JDBC/ODBC вариант
        db = DriverManager.getConnection(url);
        // изисква експлицитно изпълнение на Commit
        db.setAutoCommit(false);
        statement = db.createStatement(); // незадължително
    } catch (Exception e) {
        System.out.println("Could not initialize the database.");
        e.printStackTrace();
    }
}
```

Взаимодействие на езика с базата от данни

Изпълнение на заявки. Реализация на добавяне на данни в базата данни:

```
private void insert(<контейнер >, <индекс>) throws SQLException {
// обект от PreparedStatement, представящ командата
      PreparedStatement st = null;
      try {
         st = db.prepareStatement(<SQL команда за добавяне>);
// запис на параметрите на заявката за добавяне
         st.executeUpdate();  // изпълнение на команда
         db.commit();// възприемане на промените
         st.close();// затваряне на команда
    } catch (SQLException e) {
// обработка на изключение
    }
}
```

Взаимодействие на езика с базата от данни

Пример: Реализация на четене от базата и запис в колекция

```
    private void loadFromDB(<контейнер >) {
     ResultSet rs;
     try {
// обект от PreparedStatement, представящ командата, изпълнение
          PreparedStatement st = db.prepareStatement("Select * from
                                                    StudentRegister");
// получаване на резултатите
           ResultSet rs = st.getResultSet();
           if( rs != null){
             while (rs.next()) {     // установяване на курсор към запис
// четене на данни и запис в <контейнер>
            }
           }else{/* няма данни */}
           st.close();
       } catch (Exception e) {
// обработка на изключението
    }
}
```

Взаимодействие на езика с базата от данни

//Пример за манипулиранее на данни в таблица DBSamples.java

```
package dbTest;
import java.sql.Connection; // за връзката с базата
import java.sql.DriverManager; // за драйверите към //различни
    бази
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.util.*;
public class DBSamples {
  private Connection db; // връзка с базата от данни
  static String url = "jdbc:odbc:testDSN"; // статичен //адрес на
    източника
```

(продължава)

Взаимодействие на езика с базата от данни

```
// конструктор
  public DBSamples() {
    initDB();
    try {
      createDB();
    } catch (SQLException e) {
      System.out.println("Create database error");
    }
  }
```

Взаимодействие на езика с базата от данни

```
/**
  * Инициализация на базата
  */
 protected void initDB() {
   try {
     // зареждане на драйвера JDBC/ODBC вариант
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   // зареждане на драйвера ORACLE вариант
   // Class.forName ("oracle.jdbc.driver.OracleDriver");
   // получаване на връзката JDBC/ODBC вариант
          db = DriverManager.getConnection(url);
   // изисква експлицитно изпълнение на Commit
          db.setAutoCommit(false);
   } catch (Exception e) {
          System.out.println("Could not initialize the database.");
          e.printStackTrace();
   }
 }
```

(продължава)

Взаимодействие на езика с базата от данни

```
/**  създаване на база данни   */
    protected void createDB() throws SQLException {
    // обект от PreparedStatement, представящ командата
    PreparedStatement statement = null;
    try {
    statement = db.prepareStatement("CREATE TABLE StudentRegister (" +
                                    "StudentFN CHAR(6) NOT NULL PRIMARY KEY, " +
                                    "NAME CHAR(30) NOT NULL, " +
                                    "PersonSex CHAR(3) NOT NULL," +
                                    "PersonTitle CHAR(10) NOT NULL, " +
                                    "UpdDate DATETIME)");
            statement.execute();
            db.commit();
            statement.close();
    } catch (SQLException e) {
            // e.printStackTrace();
    }
 }
```

(продължава)

Взаимодействие на езика с базата от данни

```
 /**   * Добавяне на студент в базата от данни    */
protected void insert(Collection oColl, int index) {
    // обект от PreparedStatement, представящ командата
    PreparedStatement statement = null;
    try {
  Student oStudent = (Student) oColl.get(index);
  statement = db.prepareStatement("INSERT INTO StudentRegister
    VALUES(?,?,?,?,?)");
    setValues(statement, oStudent.getStrFacNumer(),
    oStudent.getName(), oStudent.getSex(), oStudent.getTitle(),
    new Timestamp(oStudent.getLastDate().getTime()));
    statement.executeUpdate();
            db.commit();
            statement.close();
    } catch (SQLException e) {   // e.printStackTrace();   }
  }
```

Взаимодействие на езика с базата от данни

```
/**
 * четене на студентите от базата и запис в колекция
 */
public void loadFromDB(Collection oColl) {
  PreparedStatement statement = null;
  try {
        // Изпълнение на заявката
        ResultSet rs = statement.executeQuery("select * from
  StudentRegister");
        // четене на данни и запис в контейнер
        Student oStudent = new Student();
```

Взаимодействие на езика с базата от данни

```
    while (rs.next()) {
            oStudent.setStrFacNumer(rs.getString("StudentFN"));
            oStudent.setName(rs.getString("NAME"));
            oStudent.setSex(rs.getString("PersonSex"));
            oStudent.setTitle(rs.getString("PersonTitle"));
            java.sql.Date sqlDate = rs.getDate("UpdDate");
            java.util.Date dbDate = new java.util.Date(sqlDate.getTime());
            oStudent.setDate(dbDate);
// добавяне на студента в колекцията …
     }
   } catch (Exception e) {
     oColl.clear(); // нулиране на данните
  }
 }
```

Взаимодействие на езика с базата от данни

```
public static void main(String[] args) {
    CollectionSamples objFile = new CollectionSamples("Input.txt");
    objFile.printColl(objFile.oColl);
    DBSamples db = new DBSamples();
    for (int i = 0; i < objFile.oColl.size(); i++) {
        try {
            db.insert((ArrayList) objFile.oColl, i);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    CollectionSamples objDB = new CollectionSamples();
    db.loadFromDB((ArrayList) objDB.oColl);
    objDB.printColl(objFile.oColl);
 }
}// край на класа
```

Взаимодействие на езика с базата от данни

Пример:

CollectionSamples.java пример за клас съдържащ в себе си колекция от данни за студентите. Те се прочитат от файл и се записват в база от данни

```
package dbTest;
import java.io.*;
import java.util.*;
public class CollectionSamples {
// Създаване на колекция от класа   ArrayList
    public Collection oColl;
    public CollectionSamples() {
      oColl = new ArrayList();
    }
```

(продължава)

Взаимодействие на езика с базата от данни

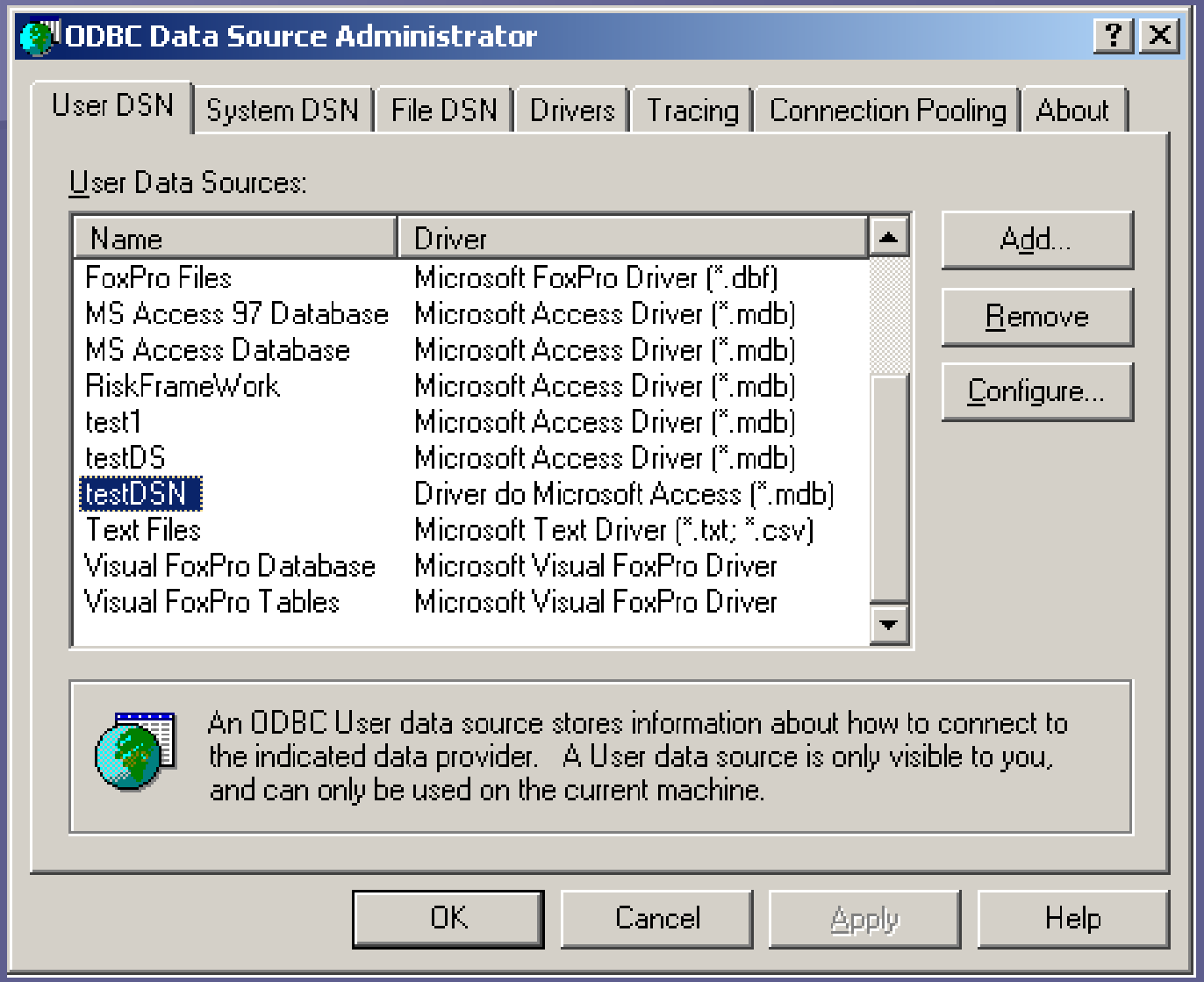
```
// Конструктор
public CollectionSamples(String fileName ) {
    try {
      Scanner scan = new Scanner (new File(fileName));
           while(scan.hasNextLine()) {
             oColl.add(new
             Student(scan.next(),scan.next(),scan.next(),scan.next()) );
           }
    } catch (FileNotFoundException e) {
           System.out.println("File Not Found ...");
    }
}
```

(продължава)

Взаимодействие на езика с базата от данни

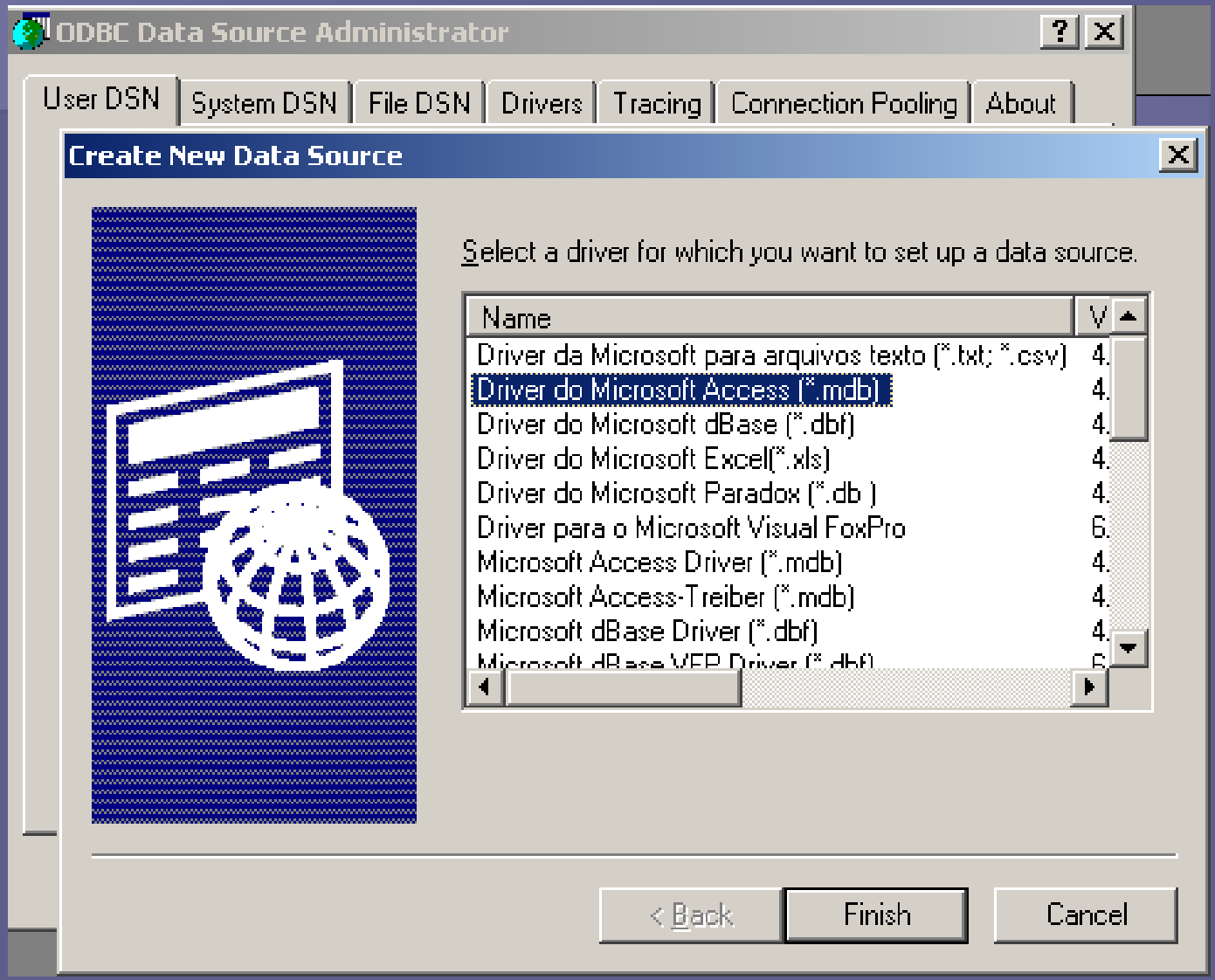
```
// Отпечатване на колекция
  public void printColl(Collection c) {
    for(Iterator It = c.iterator(); It.hasNext();)
      System.out.println(It.next().toString());
  }
  public static void main(String[] args) {
    CollectionSamples obj = new CollectionSamples("Input.txt");
    obj.printColl(obj.oColl);
   }
}
```

Настройка на връзката с базата от данни

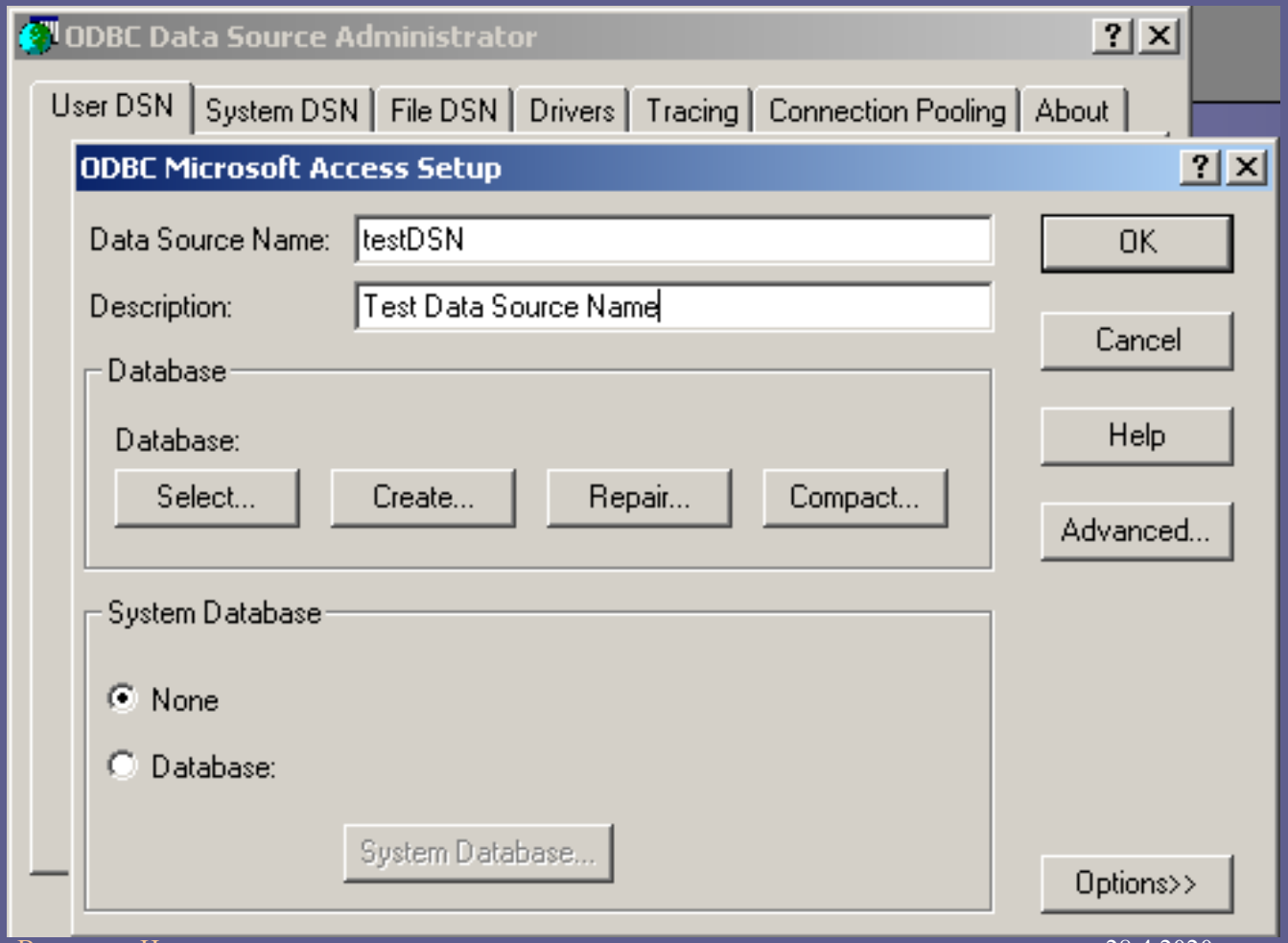


Настройка на връзката с базата от данни

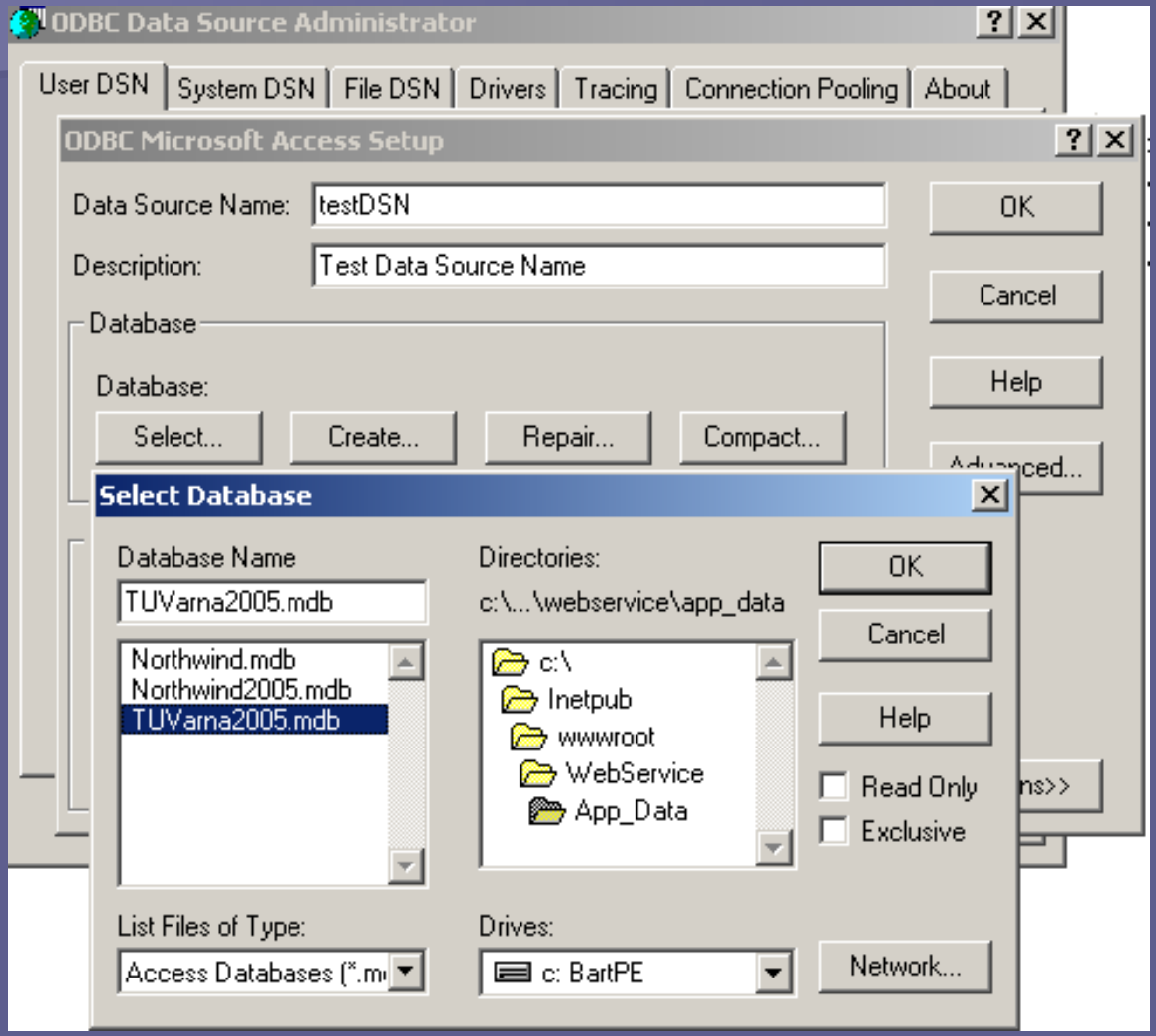
- Избор на драйвер:



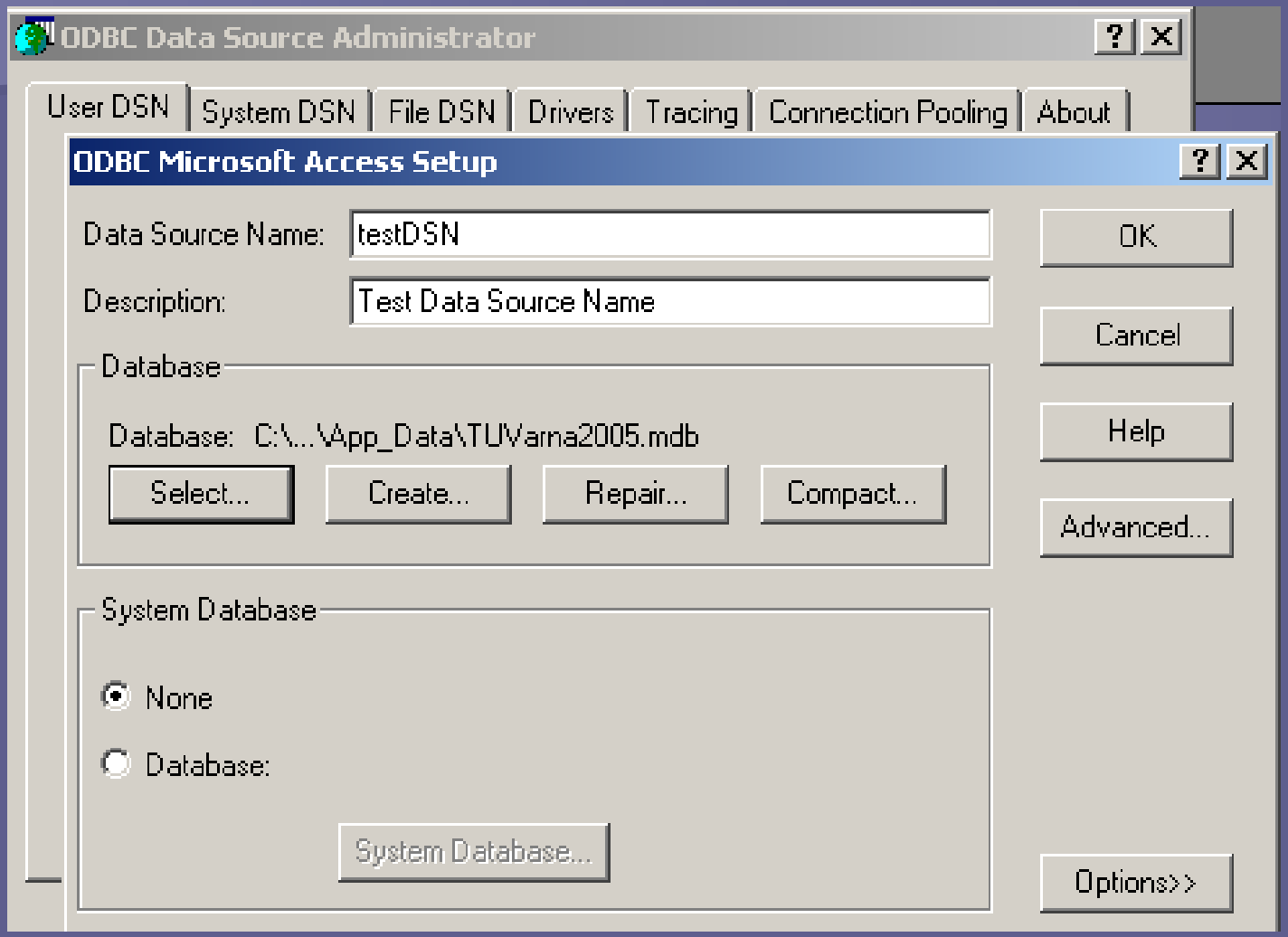
Настройка на връзката с базата от данни
Създаване на даннов източник Access



Настройка на връзката с базата от данни Създаване на даннов източник Access



Настройка на връзката с базата от данни
Създаване на даннов източник Access



Настройка на връзката с базата от данни Създаване на даннов източник Access

Резултат: името на данновия източник, който се използва в примерния код за достъп до базата от данни Access:

static String *url* = "jdbc:odbc:**testDSN**";

Настройка на връзката с базата от данни
Създаване на даннов източник Access
Java 7+

```
public void initDB() {
      try {
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
         db =
    DriverManager.getConnection("jdbc:ucanaccess://C:/…/People.mdb");
         db.setAutoCommit(false);
         statement = db.createStatement();
       } catch (Exception e) {
         System.out.println("Could not initialize the database.");
        e.printStackTrace();
      }
   }
```

Настройка на връзката с базата от данни
Създаване на даннов източник Access
Java 7+

```
public void closeDB() {
    try {
        if (statement != null) {
            db.commit();
            statement.close();
        }
        if (db != null) {
            db.close();
        }
    } catch (Exception e) {
        System.out.println("Could not close the current connection.");
        e.printStackTrace();
    }
}
```

Въпроси ?