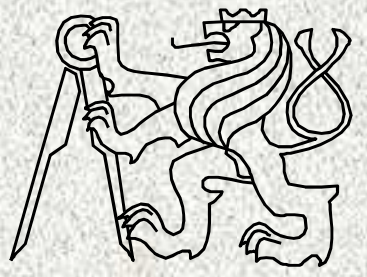
Databáze a JDBC API

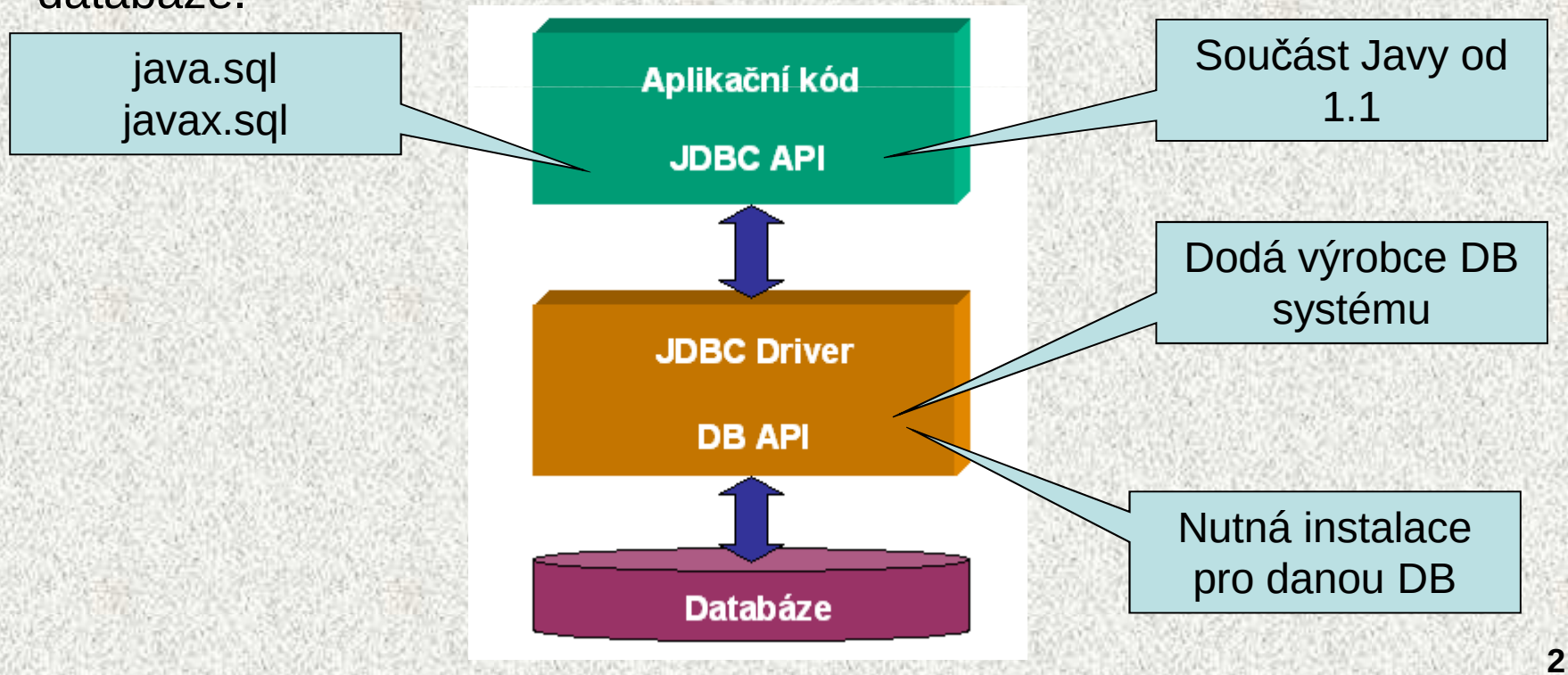


A0B36PR2-Programování 2

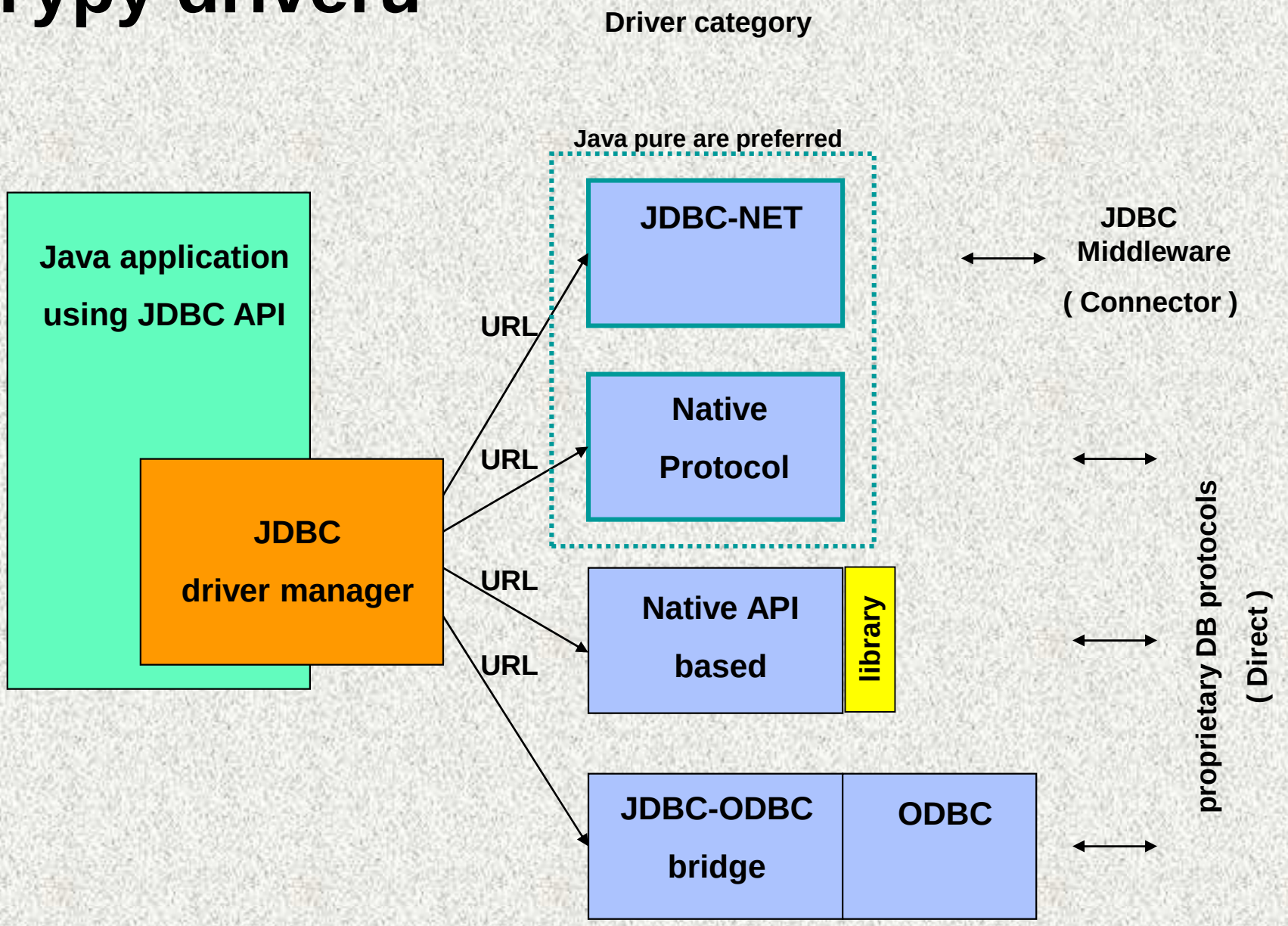
Fakulta elektrotechnická
České vysoké učení technické

JDBC ~ Java Data Base Connectivity

- **Java Database Connectivity** (**JDBC**) je API, které definuje jednotné rozhraní pro přístup především k relačním databázím.
- Pro přístup ke konkrétnímu databázovému serveru je potřeba JDBC ovladač, který poskytuje tvůrce databázového serveru.
- Základem konceptu JDBC je využití funkčnosti poskytované JDBC ovladačem, který je následně překládá do nativních volání dané databáze.



Typy driverů



Pro zájemce

Příklad pro Derby / MySQL / Oracle

Před spuštěním je třeba zajistit cestu k driverům, tj. classpath musí vést ke zkompilovaným třídám driverů, které bývají uloženy v adresářích souborů typu jar či zip - viz firemní stránky.

- derbyclient.jar v …\jdk1.6\db\lib\ či ...\AppServer\derby\lib\
- mysql-connector-java-3.1.8a.zip (http://dev.mysql.com/downloads)
- classes12.zip (www.oracle.com/technology/software/index.html)

```
Class.forName( "org.apache.derby.jdbc.ClientDriver" );
Class.forName( "org.gjt.mm.mysql.Driver" );
Class.forName( "oracle.jdbc.driver.OracleDriver" );

String derby  = "jdbc:derby://localhost:1527/samples" ,
       mysql = "jdbc:mysql://localhost:3306/Z" ,
       oracle = "jdbc:oracle:thin:@cs.felk.cvut.cz:1526:oracle" ;
Connection c1 = DriverManager.getConnection( derby, "app", "app" ),
             c2 = DriverManager.getConnection( mysql, "root", "" ),
             c3 = DriverManager.getConnection( oracle, "scott", "tiger" );
Statement stmt = c3.createStatement( );
```

Kategorie driverů

Drivery vyrábí a dodává producent DB

Preferované a Java-pure:

- 3. JDCB-NET – Komunikuje s mezivrstvou vhodným síťovým protokolem (Javový protokol), nejčastěji používaný
- 4. Native protocol – Překládá JDBC volání přímo do síťového protokolu DBMS, což umožňuje klientovi přímé volání DB, neinterpretuje, přímo překládá do formátu DBMS.

Pro dočasná řešení:

- 2. Native API based – Nástavba na klientské knihovně, překládá JDBC volání do volání té knihovny, nepřenositelnost kódu.
- 1. JDBC-ODBC Bridge – Open Database Connectivity standardizované softwarové API, Microsoft, musí být instalován na každém klientovi, nerobustní, nedoporučený, pomalý.

Pro zájemce

Proces JDBC

1. Zavedení ovladače JDBC
2. Připojení k systému DBMS
3. Vytvoření a spuštění dotazu SQL
4. Zpracování odpovědi ze systému DBMS
5. Odpojení od systému DBMS

1. Zavedení ovladače JDBC

`Class.forName(String className)`

- Sdělení, který ovladač budeme používat, který máme k dispozici

Např.

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
Class.forName("oracle.jdbc.driver.OracleDriver");
Class.forName(„sun.jdbc.odbc.JdbcOdbcDriver");
```

Kde např. **`oracle.jdbc.driver.OracleDriver,`**

je název ovladače databáze

Dříve musíme pomocí administrátora datových zdrojů připojit k databázi tento ovladač, viz cvičení v NetBeans

Ovladač **derby** v NetBeans

- Ovladač „derby“ je multiuser DB napsaná v Javě jednak embedded, jednak network, reflektuje JDBC 4.0, SQL-99, SQL-2003.
- DB server musí být instalován např. na: AppServer či JavaDB

Knkrétně v Netbeans:

Ve složce projektu Libraries → Add JAR/Folder:

glassfish-v2.1 → javadb → lib → připojit derbyclient.jar .

Ve složce Services ->

- Start DB serveru: Database → Java DB → Start Server

  Output: Apache Derby … on port 1527 …
- Připojení DB:

  Databases → jdbc:derby://…/... [...on ...] → RIGHT BUTTON

  Connect app app , public public …
- Vytvoření další DB:

  JavaDB → Create Database ... vyplnit

  Connect user password

Podrobně: **http://db.apache.org/derby/** , **http://wiki.netbeans.org/wiki/**

Viz cvičení

2. Připojení k systému DBMS

```
static DriverManager.Connection getConnection
                    (String url, String user, String password)
```

- Připojení se do systém DBMS, například

```
Connection c = DriverManager.getConnection
      ("jdbc:derby://localhost:1527/Z", "jel", "jel");
```

  - `"jdbc:derby://localhost:1527/Z",` - URL databáze
  - `"jel"` - login
  - `"jel"` – heslo

- Jiný příklad

```
Connection conn =
   DriverManager.getConnection("jdbc:oracle:thin:@oraserv.felk
   .cvut.cz:1521:devaite", "T10", "oracle");
```

- Metoda vrací instanci třídy **`java.sql.Connection`**, které řídí komunikaci mezi ovladačem a programem
- Adresu lze získat od administrátora DB

3. Vytvoření a spuštění dotazu SQL

Statement createStatement()
ResultSet executeQuery()

- Vytvoření dotazu – vytvoření instance třídy **Statement**

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

- **c –** instance třídy **Connection**
- vytvoří objekt pro zasílání dotazů do databáze

2. Odeslání SQL dotazu **executeQuery()**, argumentem je SQL dotaz

```
ResultSet r = s.executeQuery ("SELECT * FROM PERSON");
ResultSet r = s.executeQuery ("SELECT * FROM PERSON
                               WHERE JMENO='Jana' ");
```

3. Metoda vrací výsledek jako instanci třídy **ResultSet**

4. Spojení s databázovým systémem **s.close();**

4. Zpracování odpovědi ze systému DBMS

`ResultSet` definuje metody pro zpracování výsledku

- Informace o existenci řádky získáme **`next()`**
- Obsah řádky získáme metodou

```
get<typ>( <název sloupce>)
get<typ>( <číslo sloupce>)
```

```
String jmeno;
String prijmeni;
while (s.next()) {
jmeno = s.getString("NAME");
prijmeni = s.getString("CITY");
System.out.println(jmeno + "   " + prijmeni));
}
```

5. Odpojení od systému DBMS

- Uzavření dotazů z databáze

```
Statement s;
s.close();
```

- Uzavření spojení s databází

```
Connection c;
c.close();
```

Výjimky ve zpracování databází

```
try{
Class.forName(„sun.jdbc.odbc.JdbcOdbcDriver");
Connection c = DriverManager.getConnection
      ("jdbc:derby://localhost:1527/sample","app","app");

} catch (ClassNotFoundException e){
System.err.println(“nepodarilo se zavest jdbc” +
                                        e.getMessage());
System.exit(1);

} catch (SQLException e){
System.err.println(“nepodarilo se pripojit k databazi” +
                                          e.getMessage());
System.exit(2);

} finally {
c.close();
}
```

Metadata, rozhraní **`ResultSetMetaData`**

Informace o databázi:

- Počet sloupců
- Název sloupce
- Datový typ, velikost
- …

`ResultSetMetaData getMetaData();`	**-** přístup k metadatům
`int getColumnCount()`	- počet sloupců
`String getColumnName(i)`	- název sloupce
`String getColumnTypeName(i)`	- typ sloupce

Rozhraní ResultSet, změna obsahu databáze

Standardní SQL příkaz UPDATE pro nalezení podmnožiny pro změnu

`int executeUpdate(String sql)` - počet změněných řádků

Příklad

```
String dotaz = "UPDATE CUSTOMER SET ZIP = '4444' WHERE
  NAME = 'JumboCom'";
s=c.createStatement();
int pozadavek =s.executeUpdate(dotaz);
r=s.executeQuery("SELECT  * FROM CUSTOMER");
```

Otevření spojení s databází

`Statement createStatement( ... )`

- Příkazový objekt se spustí okamžitě potom, co je spuštěn
- Před spuštěním v DBMS je objekt zkompilován

`PreparedStatement prepareStatement( ... )`

- Používá se pro spuštění již předem zkompilovaného dotazu (zkompilovaný dotaz je vytvořen před odesláním do DBMS)
- Používá se při násobném spouštění dotazů, kompilace jen jednou
- Dotaz lze modifikovat výběrovým kritériem – náhradním znakem

`CallableStatement prepareCall( ... )`

- Používá se ke spuštění uložených procedur uvnitř DBMS
- SQL dotaz obsahuje volání speciálních procedur
- Provedení příkazů je rychlejší než u **`Statement`**
- Krok kompilace je přeskočen

Porovnání typů Statement

Vytvoření dotazu – vytvoření instance třídy **Statement**

	Statement	**Prepared Statement**	**Callable Statement**
Where is code created	client	client	server
Where stored	client	server	server
Writing code	Java SQL operation	Java language SQL operation	DB procedural language
Configurability	high	high first time then low	low
Portability	high	high provided DB supports prepared	low
Data transfer efficiency	low	low first time then high	high

PreparedStatement

Dotaz modifikovaný výběrovým kritériem – náhradním znakem:

1. Formulace dotazu, kde specifikované hodnoty jsou nahrazeny ‘?’
2. Deklarace objektu **PreparedStatement**
3. Metodou **void setString(int parameterIndex, String x)** se přiřadí „pořadí“ otazníku skutečné hodnotě parametru
4. Spustí se dotaz

```
String ss= "select * from CUSTOMER where
                        CUSTOMER_ID = ? or ZIP= ?";
      PreparedStatement ps = c.prepareStatement(ss);
      ps.setString(1, "25");
      ps.setString(2, "94401");
      r=ps.executeQuery();
```

Další možnosti JBCD API

- **`CallableStatement`** , spuštění uložených procedur uvnitř DBMS
- Zpracování jednotlivých záznamů, virt. kurzor
- Zpracování skupiny záznamů
- Vytvoření databáze
- Vkládání a rušení řádků