# Java – technologie zaawansowane

## JDBC

Instytut Fizyki i Informatyki Stosowanej
Uniwersytetu Łódzkiego

# JDBC

- Java Database Connectivity
  - API dla relacyjnych baz danych
    - Niezależność od implementacji
      - MySQL, PostgreSQL, ORACLE, Db2…
        - » Nawet płaskie (plikowe) bazy
      - Jeden kod wiele baz
        - » Różnica na poziomie drivera
        - » Driver Manager
    - Oparte na ODBC

# Implementacje

- Powszechnie używane
  - JDBC 3.0
  - JDBC 4.0
    - Java 6
      - Trochę fundamentalnych zmian
        - » DriverManager
        - » SQL <-> XML
        - » Annotacje SQL Query
  - Będziemy używać JDBC 3.0

# Czego potrzebujemy

- Bibliotek
  - java.sql, javax.sql
- Driver dla konkretnej bazy
    - (Typ 3,4) Czysta Java
    - (Typ 1,2) JNI (silnik przeważnie w C++)

# Połączenie z DB

- DriverManager
  - Załadowanie klasy Driver do obsługi konkretnej bazy danych
    - Singleton

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

  - Połączenie z bazą

```
Connection dCon =DriverManager.getConnection(
"jdbc:mysql:127.0.0.1/pkdb” ,Name,passwd);
```

# Proste zapytania

- Klasa Statement
  - Tworzenie i obsługa zapytań

```
Statement statement = dCon.createStatement();
```

  - Następnie mając Zapytanie SQL

```
sqlQuery = "SELECT * from student WHERE id>12 AND imie='Adam'";
```

    - statement.execute(sqlQuery)
      - » Dla SELECT
        - » statement.executeQuerry(sqlQuery)
      - » Dla UPDATE/INSERT/DELETEs
        - » statement.executeUpdate(sqlQuery)

# Zapytania Inaczej

- PreparedStatement
  - Zabezpieczenie przed SQL Injection
  - Przygotowanie zapytania

```
sqlQuery = "SELECT * from student WHERE id>? AND imie=?";
PreparedStatement pst= dCon.prepareStatement(querry);
```

    - ? – parametry zapytania
  - Wartości parametrów
    - setString(id,Value), setInt(id,Value),setDouble(id,Value) ..
    - Weryfikacja typu,
    - Wycinanie znaków: ', "

# PreparedStatements cd

- Wykonanie zapytania
  - prepareStatement.executeQuery()
  - prepareStatement.executeUpdate()

# CallableStatement

- Procedury „trzymane” na serwerze Bazy
  - CallableStatement

```
CallableStatement cst= dCon.prepareCall("{call procedura(?,?,?)}");
```

  - Reszta jak dla PreparedStatement

# Wyniki zapytań

- ResultSet
  - Otrzymywany z obu typów zapytań
    - statement.getResultSet();
    - prepareStatement.getResultSet();
    - callableStatement.getResultSet();
  - Iterator
    - rs.next()
    - rs.getString("imie"),getInt("id"),…

# Szybki Update/Insert/delete

- ResultSet
  - Update

```
rs.absolute(5);
rs.updateString("NAME", "AINSWORTH");
rs.updateRow();
```

  - Insert/Delete

```
rs.insertRow();
rs.deleteRow();
```

# Transakcje

- Wykonywanie kilku zapytań jednocześnie
  - Konsystencja danych
    - » Struktura np. PRIVATE/FOREIGN KEY
  - Wyłączamy AutoCommit

```
dCon. setAutoCommit(false);
```

  - Planujemy zapytania
  - Dokonujemy wszystkich zaplanowanych zapytań

```
dCon. commit();
```

# Transakcje cd

- savepoint, rollback
- Kiedy warto użyć rollback
  - Jeżeli są błędy w zapytaniu
    - SQLException

# Typy Danych

- Znane dobrze z Javy
  - String, Int, Float,Double,LongInt…
  - Date
- Specyficzne dla baz danych
  - BLOB
    - (Binary Large Objects)
  - CLOB, NCLOB
    - (Character Large Objects)

# Zamykanie pracy

- Zaleca się wykonanie close() na każdym obiekcie JDBC

```
Statement st = dCon.createStatement();
try {
    ResultSet rs = stmt.executeQuery( "SELECT * FROM TableX" );
    try { ... }
    }
    finally {rs.close();}
}
finally {st.close();}
```

  - Nie zrzucajmy wszystkiego na GarbageCollector

# RDB <->Java

- Relacyjne Bazy Danych
  - Dane oparte na tabelach i relacjach
- Java
  - Dane to obiekty
    - Często złożone
- Obiekt Jawy odpowiada wielu tabelom
- Obiektowe Bazy Danych + JAVA
  - Najlepsze rozwiązanie ?

# O/R Mapping

- Object-Relational Mapping
  - Tłumaczenie pomiędzy
    - Obiektem Javy
    - Relacyjną bazą danych
- JPA/HIBERNATE
  - Biblioteka/famework

# MVC

- Zasada
  - Nie mieszamy
    - Wyświetlana
    - Kontroli
    - Danych
  - Co zrobić z Bazą danych
    - DAO (Data Access Object)
      - Wzorzec
      - Pośrednicy do danych

# DAO

- Dla każdego obiektu
  - Enkapsulacja
  - Np. klasa Osoba
    - Tworzymy klasę OsobaDAO
      - » Osoba findById(Long id)
      - » void saveOrUpdate(Osoba o)
      - » void delete (Osoba o)
      - » List<Osoba> findAll()
      - » List<Osoba> findByImie(String imie)
      - » …

# OsobaDAO

```
public Osoba findById(Long id)
{    Transaction trans = null;
     Session session= MySessionFactory.getInstance().getCurrentSession();
     //…. Pobieramy z bazy
     return os;
     }
     finally {
     session.close();
     }
}
```

# MVC

- PLUS
  - Nasza aplikacja
    - Bez zmian (poza DAO)
      - Działa z danymi  z pliku
      - Struktura bazy mniej ważna
        - » Działa z bazą testową
        - » Bazą produkcyjną (jeśli ma inną strukturę)