PA165: Persistence Layer

Petr Adámek

Content: Persistence I.

- Introduction to data persistence
  - Where to store data
  - How to work with data (Persistence technologies in Java EE)
  - Architecture of data persistence layer
- Introduction to ORM
  - What is ORM
  - Basic Principles
- JPA
  - Introduction
  - Entities in JPA
  - JPA Components
  - Entity Lifecycle

Content: Persistence II

- JPA
  - Advanced Mapping
  - Querying
    - JPQL
    - Criteria API
- Alternatives
  - EJB 2.x
  - JDO
  - JDBC
  - Embedded SQL
  - Spring JDBC
  - iBatis
- Best practices

INTRODUCTION INTO DATA PERSISTENCE LAYER

Where to store data

Data can be stored on different places

- Relational Database (RDBMS)
- Object Database
- XML Database
- DMS (Document Management System)
- CMS (Content Management System)
- Post-relational database (Caché)
  - Temporary
  - Hiearchicke
  - Prostorove
- No-SQL Database
- Another Information System (CRM, ERP, etc.)

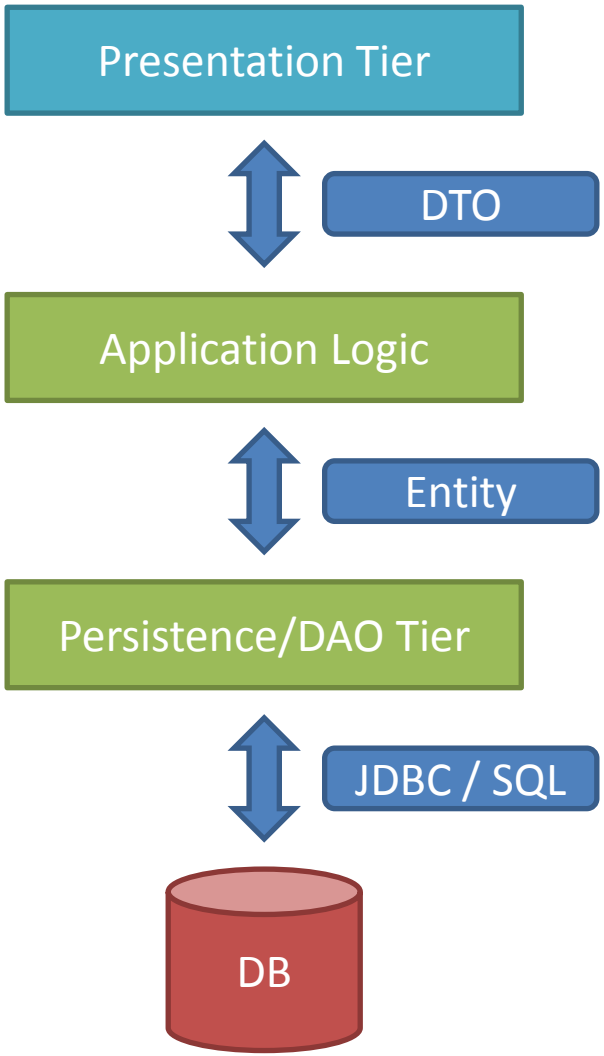
Relational Databases

- The most frequent data storage for enterprise applications
  - Relational data model is simple but very powerful
  - Suitable and sufficient for most of common applications
  - Good theoretical model (Relational Algebra, Relational Calculus)
  - Simplicity => High Performance (eg. due simple optimizations)
  - Proven and well established technology (40 years of development, tools, standards, reliability, high penetration, lots of experts, etc.)
  - Data are separated from application and can be easily shared between different applications
  - Independent on concrete platform or programming language

Persistence Technologies

- With Object Model (ORM or other model conversion)
  - Legacy EJB 2.x
  - Hibernate
  - JPA
  - JDO
- With Relational Model
  - JDBC
  - Commons DB Utils
  - Spring JDBC
  - iBatis
  - Embedded SQL

Architecture of persistence layer



Architecture of persistence layer

- DAO (*Data Access Object*) design pattern
  - Separation persistence and data access from business logic
  - Allows to modify persistence implementation or replace persistence technology at all without affecting application logic.

Transaction management

- Transaction are not controlled on DAO level
- Why?
  - Single transaction can contain more operations provided by different DAO components
  - Transaction management should be independent on persistent technology
- Transaction management will be discussed later in lecture focused on Service Layer

INTRODUCTION TO ORM

What is ORM

Object-relational mapping (ORM)

- Technique for automatic conversion between object model and relational data model
- You work with objects, but they are stored in a traditional relational database.

INSERT INTO people (id, name) VALUES (1, "Pepa");

```
Person p = new Person(1, "Pepa");
em.persist(p);
em.getTransaction().commit();
```

UPDATE people SET name = "Honza" WHERE id = 2;

```
Person p = em.find(Person.class,2);
p.setName("Honza");
em.getTransaction().commit();
```

Why ORM

- As we already discussed, the most frequent storage is RDBMS
- But we usually want to work with Object Model
- Why Object Model?
  - It is natural for object oriented programming language
  - Working with Object Data Model is straightforward, friendly and easy
  - See example

Why ORM: JDBC example

```
public String getPersonName(long personId) throws SQLException {
  PreparedStatement st = null;
  try {
    st = connection.prepareStatement(
      "SELECT name FROM people WHERE id = ?");
    st.setLong(1, personId);
    ResultSet rs = st.executeQuery();
    if (rs.next()) {
      String result = rs.getString("name");
      assert !rs.next();
      return result;
    } else {
      return null;
    }
  } finally {
    if (st != null) { st.close(); }
  }
}
```

Why ORM: JPA Example

```
public String getPersonName(long personId) {
    Person p = em.find(Person.class,personId);
    return p.getName();
}
```

ORM: What We Get and Lost

- ORM Benefits
  - Possibility to work with natural object model
  - Portability between different RDBMS with different SQL Dialect
  - Type checking at compile phase
  - No runtime error in SQL statements
  - Simplifies testing
  - Simpler and clearer code
  - More effective development (auto complete, access to JavaDoc, etc.)
- ORM drawbacks
  - Less performance in some cases (ORM has some overhead).
  - No access to advantages of relational model and features of RDBMS (e.g. storage procedures)

Basic terms

Pojmy, které byste měli znát

- **JDBC, SQL, Transakce,**

Pojmy, které si definujeme

- **Entita** – doménový objekt, který reprezentuje data ukládaná do databáze (např. osoba, faktura, předmět).
- **DTO** (*Data Transfer Object*) – objekt, který slouží pro zapouzdření dat a jejich transport mezi komponentami.
- **POJO** (*Plain Old Java Object*) – jednoduchá třída, na kterou nejsou kladeny žádné zvláštní požadavky. Nemusí implementovat žádné rozhraní, nemusí rozšiřovat žádnou jinou třídu, ani není závislá na žádné jiné třídě, balíku nebo frameworku. Jediné podmínky, které na ni mohou být kladeny, je přítomnost bezparametrického konstruktoru, a dodržení konvencí pro pojmenování get/set metod.

Standards and Approaches

Entity EJB (EJB 2.1/JSR 153; J2EE 1.4)

- Vyžaduje aplikační server s EJB kontejnerem.
- Entita je heavyweight komponenta, jejíž instance se nachází v EJB kontejneru a přístup k ní probíhá prostřednictvím vzdáleného volání metod.
- Problém s latencemi (řeší se pomocí DTO, příp. DAO).
- CMP a BMP
- Od verze EJB 3.0 (JSR 220) je preferováno JPA.

JDO (JDO 3.0/JSR 243)

- Obecný standard pro perzistenci v Javě.
- Není omezeno na relační databáze, objekty mohou být ukládány do úložiště libovolného typu

JPA (JPA 2.0/JSR 317; Java EE 6)

- Java EE standard pro ORM inspirovaný Hibernate.
- Entita je lightweight POJO, který může být libovolně předáván mezi komponentami, lokálně i vzdáleně.

EJB 2.x Entity

```
public abstract class PersonBean implements EntityBean {
    private EntityContext context;

    public abstract Long getId();
    public abstract void setId(Long id);
    public abstract String getName();
    public abstract void setName(String name);

    public Long ejbCreate (Long id, String name) throws CreateException {
       setId(id); setName(name); return id;
    }
    public void ejbPostCreate (Long id, String name) throws CreateException {}

    public void setEntityContext(EntityContext ctx)  { context = ctx; }
    public void unsetEntityContext() { context = null; }

    public void ejbRemove() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbPassivate() {}
    public void ejbActivate() {}

    public PersonDTO getPersonDTO() {
        return new PersonDTO(getId(),getName());
    }
}
```

POJO Entity

```
public class Person {

    private Long id;
    private String name;

    public Long getId()                { return id; }
    public void setId(Long id)         { this.id = id; }
    public String getName()            { return name; }
    public void setName(String name)   { this.name = name; }

    public boolean equals(Object o) {
        if (this == o) { return true; }
        if (getId() == null) { return false; }
        if (o instanceof Person) {
            return getId().equals(((Person) o).getId());
        } else {
            return false;
        }
    }

    public int hashCode() { return id==null?0:id.hashCode(); }
}
```

Mapping Definition

Prostřednictvím anotací

- Definice objektového modelu i jeho mapování je na jednom místě.
- Větší přehlednost, jednodušší vývoj, snazší údržba.

Prostřednictvím externího souboru (obvykle XML)

- nezávislost kódu entit na konkrétní technologii zajišťující ORM;
- možnost měnit mapování bez nutnosti modifikovat kód.

Pomocí speciálních JavaDoc komentářů

- Z doby, kdy Java nepodporovala anotace.
- Viz XDoclet.

Generování schématu databáze na základě definice mapování

- Máme vytvořené entity a definici mapování a chceme si ušetřit práci s vytvářením schématu databáze.
- Je možné automaticky vytvářet tabulky při prvním spuštění aplikace.
- Výhodné zejména při vývoji, kdy dochází ke změnám datového modelu.
- Vhodné, pokud je datový model zcela pod kontrolou naší aplikace.
- Problém, pokud se mění datový model a již máme v databázi existující data.

Generování entit a definice mapování na základě schématu databáze

- Máme vytvořené schéma databáze a chceme si ušetřit práci s vytvářením entit a definicí mapování (např. vyvíjíme aplikaci pro přístup k již existujícím datům).
- Obvykle je nutné vygenerované soubory ručně opravit.

JAVA PERSISTENCE API

JPA Introduction

Java Persistence API

- POJO Entity, inspirované ORM nástrojem Hibernate
- Jedná se o rozhraní, které implementují různé ORM nástroje od různých dodavatelů.
- Obsahuje základní funkce, jednotlivé implementace mohou prostřednictvím svého proprietárního rozhraní poskytovat řadu dalších služeb a možností.

Verze a specifikace

- **JPA 1.0** – součást Java EE 5; vzniklo jako součást EJB 3.0 (JSR 220), lze jej ale použít zcela nezávisle.
- **JPA 2.0** – součást Java EE 6; JSR 317.

ORM nástroje implementující JPA

- Hibernate
- TopLink, TopLink Essentials
- Eclipse Link (JPA 2.0)
- Open JPA

Entity in JPA

Entity

- Reprezentují jednotlivé doménové objekty.
- Jsou klasické POJO, tj. jednduché a obyčejné objekty.
- Entita má atributy, které reprezentují vlastnosti doménového objektu.
- Atributy jsou přístupné pomocí get/set metod.
- Entita musí mít bezparamentrický konstruktor a pokud má být používána k přenosu dat prostřednictvím RMI, musí být serializovatelná.

Definice mapování

- Způsob uložení entity do relační databáze je definován pomocí anotací, nebo pomocí XML souboru.
- Důsledně se uplatňuje princip *convention-over-configuration.*

JPA Entity

```
@Entity
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;

    public Long getId()               { return id; }
    public void setId(Long id)        { this.id = id; }
    public String getName()           { return name; }
    public void setName(String name) { this.name = name; }

    public boolean equals(Object o) {
        if (this == o) { return true; }
        if (getId() == null) { return false; }
        if (o instanceof Person) {
            return getId().equals(((Person) o).getId());
        } else {
            return false;
        }
    }

    public int hashCode() { return id==null?0:id.hashCode(); }
}
```

Example

- Working With Entity

Configuration

Konfigurace

- Uložena v souboru persistence.xml
- Může obsahovat více tzv. Persistence Unit.

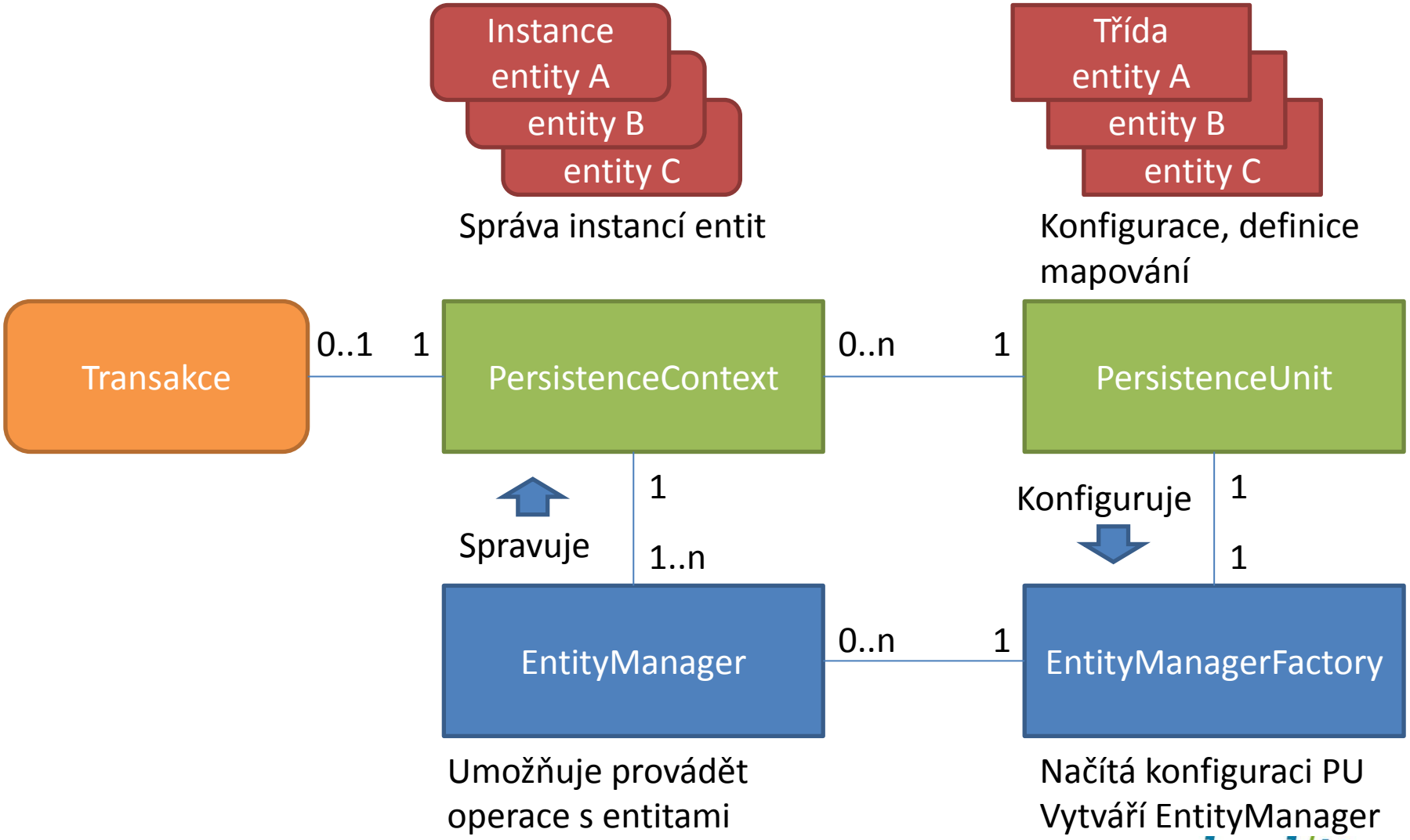
Persistence Unit

- Seznam tříd, které daná PU spravuje
- Konfigurace připojení k databázi
  - JNDI název DataSource
  - JDBC url, jméno, heslo
- Způsob řízení transakcí
- Způsob generování tabulek při spuštění aplikace

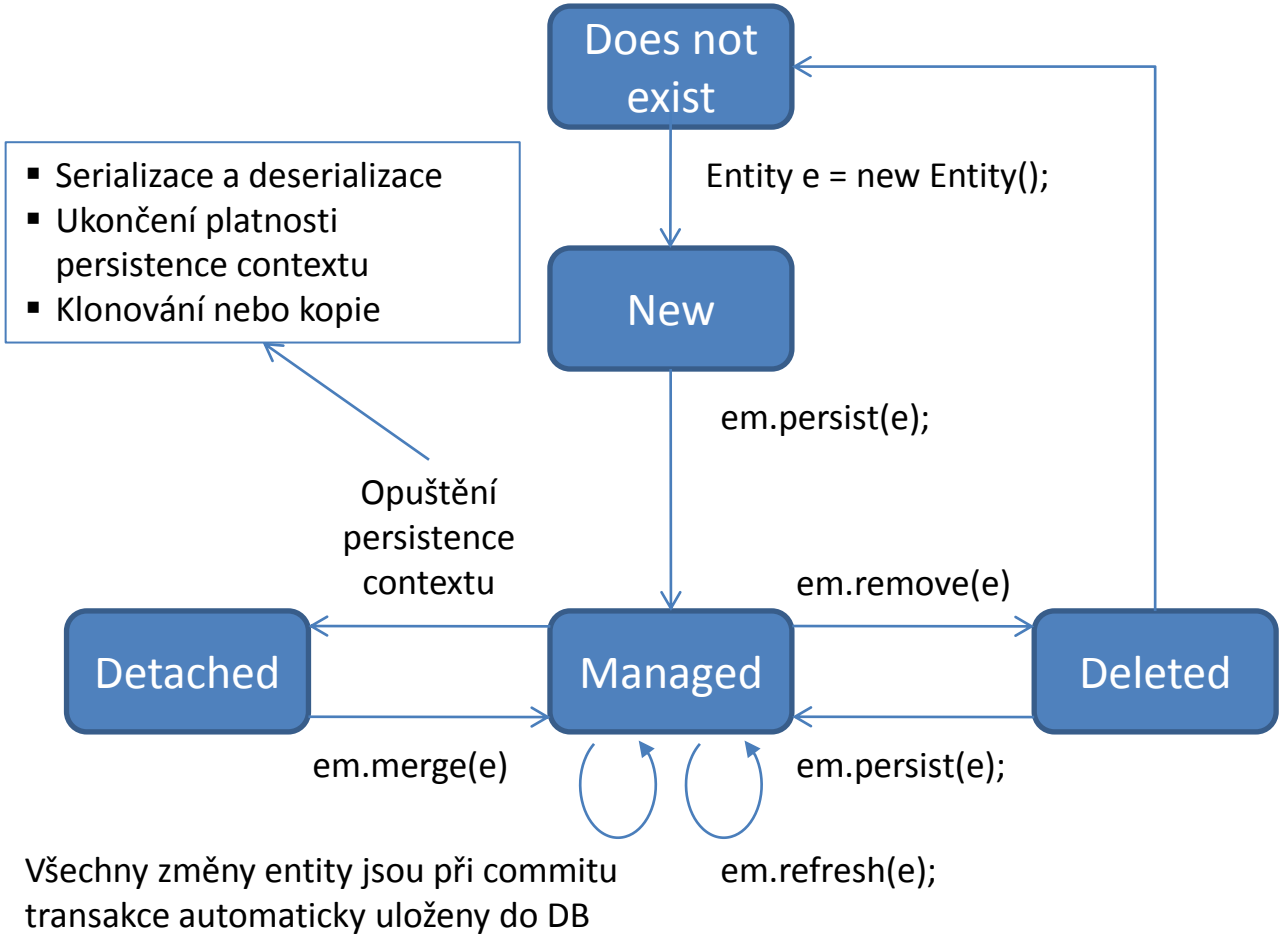
Konfigurační parametry

- U JPA 1.0 nejsou názvy konfiguračních parametrů standardizovány, u JPA 2.0 již ano.
- Parametry mohou být nastaveny také při vytváření instancí EntityManagerFactory a EntityManager

JPA Architecture



Entity Lifecycle



- Serializace a deserializace
- Ukončení platnosti persistence contextu
- Klonování nebo kopie

Quiz

- Entity Lifecycle

RELATIONSHIPS AND ADVANCED MAPPING

Example

- Relationships
  - OneToOne
  - OneToMany
  - ManyToOne
  - ManyToMany
  - Uniderctional
  - Bidirectional
  - Cascade Operations

QUERYING

Querying

- JPQL
  - Query language similar to SQL
  - Supports scalar values, tuples, entities or constructed objects
- Criteria API
  - From JPA 2.0
  - Allows to build query programatically

Example

- JPQL
  - Executing simple query
  - Named queries
  - Returning scalar values
  - Returning tuples
  - Constructing objects
- Criteria API
  - Simple example

OTHER TECHNOLOGIES

EJB 2.x

- Incompatible with DAO Design Pattern
  - Actually, DAO Pattern was designed as replacement for EJB 2.x Entities
- Requires Java EE Application server with EJB Container
- Entity is heavyweight component, instances are located in EJB Container and accessed remotely
- Problem with latencies (reason for introducing DAO and DTO design patterns)
- CMP versus BMP
- JPA is preferred from EJB 3.0

EJB 2.x Entity

```
public abstract class PersonBean implements EntityBean {
    private EntityContext context;

    public abstract Long getId();
    public abstract void setId(Long id);
    public abstract String getName();
    public abstract void setName(String name);

    public Long ejbCreate (Long id, String name) throws CreateException {
       setId(id); setName(name); return id;
    }
    public void ejbPostCreate (Long id, String name) throws CreateException {}

    public void setEntityContext(EntityContext ctx)  { context = ctx; }
    public void unsetEntityContext() { context = null; }

    public void ejbRemove() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbPassivate() {}
    public void ejbActivate() {}

    public PersonDTO getPersonDTO() {
        return new PersonDTO(getId(),getName());
    }
}
```

Embedded SQL

- SQL written directly in the code
- Code is processed with special preprocessor before compiling
- Preprocessor process SQL expressions, checks their validity, performs type checking a translates them into expression of used programming language.
- Preprocessor requires database connection

```
public String getPersonName(long personId) {
    String name;
    #sql {
        SELECT name INTO :name
        FROM people WHERE id = :personId
    };
    return name;
}
```

Spring JDBC

- Spring library implementing *Template Method* design pattern
- Cleaner code, faster development, easier maintenance
- Unlike ORM or Embedded SQL does not solve the problem with errors in SQL expressions, that become apparent until runtime

```
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

public String getPersonName(long personId) {
  return jdbcTemplate.queryForObject(
    "SELECT name FROM people WHERE id = ?",
    String.class,personId);
}
```

Apache Commons DBUtils

- Another library implementing *Template Method* design pattern.

```
QueryRunner queryRunner =
  new QueryRunner(dataSource);
ResultSetHandler<String> stringHandler =
  new ScalarHandler<String>();

public String getPersonName(long personId) {
  return queryRunner.query(
    "SELECT name FROM people WHERE id = ?",
    stringHandler,personId);
}
```

iBatis/MyBatis

- Originally Apache project, retired, currently developed as MyBatis
- SQL Queries are separated from code
  - In XML file
  - In annotations (in new versions)
- More powerful that simple libraries like Spring JDBC, more lightweight than ORM

Questions

?