

# Technologie Java Enterprise

## Persistence API #1

Łukasz Rybka · Gdańsk 2015

# JPA - wprowadzenie

**JPA - wprowadzenie**

## **Mapowanie obiektowo-relacyjne**

**JPA - wprowadzenie**

**Mapowanie obiektowo-relacyjne**  
**JDBC i Connection Pool**

## JPA - wprowadzenie

**Mapowanie obiektowo-relacyjne**

**JDBC i Connection Pool**

**Definiowanie encji**

## JPA - wprowadzenie

Mapowanie obiektowo-relacyjne

JDBC i Connection Pool

Definiowanie encji

Entity Manager

## JPA - wprowadzenie

Mapowanie obiektowo-relacyjne

JDBC i Connection Pool

Definiowanie encji

Entity Manager

Transakcje

# JPA - relacje i walidacja



# JPA - relacje i walidacja

## One-to-one

## JPA - relacje i walidacja

One-to-one

One-to-many

## JPA - relacje i walidacja

One-to-one

One-to-many

Many-to-one

## JPA - relacje i walidacja

**One-to-one**

**One-to-many**

**Many-to-one**

**Many-to-many**

## JPA - relacje i walidacja

**One-to-one**

**One-to-many**

**Many-to-one**

**Many-to-many**

**Bean validation w JPA**

# JPA - wyszukiwanie

**JPA - wyszukiwanie**

# Java Persistence Query Language

**JPA - wyszukiwanie**

# Java Persistence Query Language

## Criteria API



# Dlaczego warto uczyć się JPA?

## Java Portal Developer

### Jeśli:

- *lubisz się rozwijać, czytać, i pogłębiać swoją wiedzę,*
- *nie lubisz bezczynności, ograniczeń i korporacyjnej inercji,*
- *nie cierpisz mechanicznej pracy, wykonywania tych samych czynności i popełniania tych samych błędów dwa razy,*
- *nie przyjmujesz niczego na wiarę, tylko sprawdzasz i drążysz aż do zrozumienia,*
- *chcesz być wśród najlepszych w tym co robisz i być dumnym ze swoich osiągnięć,*
- *nie boisz się wyzwań,*
- *(opcjonalne) lubisz grać w piłkarzyki lub ping-ponga*

### przy czym

- *programujesz w Java Enterprise co najmniej pół roku (również hobbystycznie) ,*
- *lubisz poznawać zakamarki różnych technologii, a co najmniej trzy spośród JPA, EJB, JMS, JBoss, JDBC, Hibernate, PostgreSQL, JSP, Struts w*

### co więcej, interesuje Cię

- *nie tylko czy coś działa - ale czy działa wydajnie i przewidywalnie*
- *jak w szczegółach funkcjonują poważne serwisy internetowe,*
- *w których kierunkach rozwija się dzisiejszy Internet*

# Dlaczego warto uczyć się JPA?

## Czego oczekujemy

- znajomości Java, SQL, Spring Framework 4, JPA;
- doświadczenia w modelowaniu danych z JPA i optymalizacji operacji na danych w bazie SQL;
- umiejętności tworzenia interfejsów Web API bazujących na JSON, REST;
- umiejętności podstawowej konfiguracji i zarządzania serwerem w środowisku Linux

# JPA - Entity requirements

## JPA - Entity requirements

**@javax.persistence.Entity**

## JPA - Entity requirements

**@javax.persistence.Entity**

**Public or protected, no-argument constructor  
(bean!)**

## JPA - Entity requirements

**@javax.persistence.Entity**

**Public or protected, no-argument constructor  
(bean!)**

**Not final (class and methods)**

## JPA - Entity requirements

**Persistence instance variables must be declared private, protected or package-private and can be accessed directly only by the entity class methods. Clients must access the entity state through accessor or business methods.**

# JPA - Persistence fields



## JPA - Persistence fields

**Java primitive types**

## JPA - Persistence fields

Java primitive types  
`java.lang.String`

## JPA - Persistence fields

Java primitive types

`java.lang.String`

Other serializable types (including wrappers of primitive types, `BigInteger/BigDecimal`, `Date/Calendar/Date/Time`)

## JPA - Persistence fields

Java primitive types

java.lang.String

Other serializable types (including wrappers of primitive types, BigInteger/BigDecimal, Date/Calendar/Date/Time)

User serializable types

# JPA - Persistence fields

# Enumerated types

## JPA - Persistence fields

Enumerated types

Other entities and/or collections of entities

## JPA - Persistence fields

**Enumerated types**

**Other entities and/or collections of entities**

**Embeddable classes**



# JPA - Primary Key

# Typy prymitywne

## JPA - Primary Key

Typy prymitywne  
`java.lang.String`

## JPA - Primary Key

**Typy prymitywne**

**java.lang.String**

**Typy serializowalne**

## JPA - Primary Key

Typy prymitywne

java.lang.String

Typy serializowalne

Typy wyliczeniowe

## JPA - Primary Key

Typy prymitywne

`java.lang.String`

Typy serializowalne

Typy wyliczeniowe

Inne encje lub ich kolekcje

## JPA - Primary Key

Typy prymitywne

`java.lang.String`

Typy serializowalne

Typy wyliczeniowe

Inne encje lub ich kolekcje

Klasy embed

# JPA - Primary Key



## JPA - Primary Key

[@javax.persistence.id](mailto:@javax.persistence.id)

## JPA - Primary Key

@javax.persistence.Id

@javax.persistence.GeneratedValue:

SEQUENCE

AUTO

IDENTITY

TABLE

**HSQldb**

**HSQLDB**

# HyperSQL Database

**HSQLDB**

# HyperSQL Database Baza in-memory

**HyperSQL Database**

**Baza in-memory**

**Wystarczy jeden plik JAR (!)**

**HyperSQL Database**

**Baza in-memory**

**Wystarczy jeden plik JAR (!)**

**Tryb serwera oraz prostego klienta  
(wykonanego w Swing'u)**

# HSQldb - uruchomienie

```
java -cp hsqldb.jar org.hsqldb.server.Server --database.0.mem:my-db --  
dbname.0 workkdb
```

```
java -cp hsqldb.jar org.hsqldb.util.DatabaseManagerSwing --url  
jdbc:hsqldb:hsqldb://localhost/workkdb
```



# HSQldb - uruchomienie (Maven)

```
<dependency>  
  <groupId>org.hsqldb</groupId>  
  <artifactId>hsqldb</artifactId>  
  <version>2.3.2</version>  
</dependency>
```

# HSQldb - uruchomienie (Maven)

```
mvn exec:java -Pserver
```


```
mvn exec:java -Pclient
```

# JDBC Connection Pool

## New JDBC Connection Pool (Step 1 of 2)

Identify the general settings for the connection pool.

### General Settings

**Pool Name:** 

**Resource Type:**

Must be specified if the datasource class implements more than 1 of the interface.

**Database Driver Vendor:**

Select or enter a database driver vendor

**Introspect:** ☐ **Enabled**

If enabled, data source or driver implementation class names will enable introspection.

# JDBC Connection Pool

## New JDBC Connection Pool (Step 2 of 2)

Identify the general settings for the connection pool. Datasource Classname or Driver Classname must be specified for the connection pool.

### General Settings

**Pool Name:** HSQLPool

**Resource Type:** javax.sql.DataSource

**Database Driver Vendor:**

**Datasource Classname:**



org.hsqldb.jdbc.JDBCDataSource

Select or enter vendor-specific classname that implements the DataSource and/or XADataSource APIs

**Driver Classname:**



Select or enter vendor-specific classname that implements the java.sql.Driver interface.

**Ping:**

☐ **Enabled**

When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes

**Description:**

# JDBC Connection Pool

Additional Properties (4)		
<div><div><div></div><div></div></div><div>Add Property</div><div>Delete Properties</div></div>		
Select	Name	Value
<input type="checkbox"/>	Password	
<input type="checkbox"/>	User	SA
<input type="checkbox"/>	DatabaseName	jdbc:hsqldb:hsqldb://localhost/workdb
<input type="checkbox"/>	connectionAttributes	

# JDBC Resource

**JNDI Name:** \*

**Pool Name:**

Use the [JDBC Connection Pools](#) page to create new pools

**Description:**

**Status:** ☒ **Enabled**

**persistence.xml**

**persistence.xml**

**Umieszczamy go w katalogu  
src/main/resources/META-INF**



**persistence.xml**

**Umieszczamy go w katalogu**

**src/main/resources/META-INF**

**Możliwe tworzenie wielu jednostek**

**persystencji - komunikacja z wieloma bazami**

**danych w jednej aplikacji**

# persistence.xml

```
<persistence>
  <persistence-unit name="MessagesManagement">
    <jta-data-source>jdbc/hsqldb</jta-data-source>
    <class>pl.edu.ug.messageboard.domain.Message</class>
  </persistence-unit>
</persistence>
```

# Entity Manager

## Entity Manager

**Przy jego użyciu wykonujemy wszystkie operacji komunikacji z bazą:**

**tworzenie**

**usuwanie**

**edycja**

**wyszukiwanie**

## Entity Manager

Przy jego użyciu wykonujemy wszystkie operacji komunikacji z bazą:

tworzenie

usuwanie

edycja

wyszukiwanie

**javax.persistence.EntityManager**

# Entity Manager

**Dwa rodzaje:**

**container-managed**

**(@javax.persistence.PersistenceContext)**

**application-managed**

**(@javax.persistence.PersistenceUnit)**

# persistence.xml - properties

```
<persistence>
  <persistence-unit name="MessagesManagement">
    ...

    <properties>
      <property name="javax.persistence.schema-
generation.database.action"
        value="drop-and-create" /> 1
      <property name="javax.persistence.sql-load-script-source"
        value="META-INF/sql/load.sql" /> 2
    </properties>
  </persistence-unit>
</persistence>
```

- 1 Sposób generowania schematu bazy danych przy uruchomieniu (deployment)
- 2 Załadowanie danych z pliku SQL przed uruchomieniem aplikacji

# Data i kalendarz



## Data i kalendarz

**Wymagane dla klas:**

**java.util.Date**

**java.util.Calendar**

## Data i kalendarz

### Wymagane dla klas:

`java.util.Date`

`java.util.Calendar`

### Automatyczne dla klas:

`java.sql.Date`

`java.sql.Calendar`

`java.sql.Timestamp`

# Temporal field

**Temporal field**

**Mówi o tym w jaki sposób ma być traktowane  
dane pole w bazie danych**

## Temporal field

Mówi o tym w jaki sposób ma być traktowane dane pole w bazie danych

**@javax.persistence.Temporal:**

**javax.persistence.DATE**

**javax.persistence.TIME**

**javax.persistence.TIMESTAMP**

# Lifecycle events

**Lifecycle events**

**@PrePersist**

## Lifecycle events

@PrePersist  
@PostPersist



## Lifecycle events

@PrePersist  
@PostPersist  
@PostLoad

## Lifecycle events

@PrePersist

@PostPersist

@PostLoad

@PreUpdate (EntityManager)

## Lifecycle events

@PrePersist

@PostPersist

@PostLoad

@PreUpdate (EntityManager)

@PostUpdate

## Lifecycle events

@PrePersist

@PostPersist

@PostLoad

@PreUpdate (EntityManager)

@PostUpdate

@PreRemove (EntityManager)

## Lifecycle events

@PrePersist

@PostPersist

@PostLoad

@PreUpdate (EntityManager)

@PostUpdate

@PreRemove (EntityManager)

@PostRemove

# Lifecycle events

```
@Entity
public class Message implements Serializable {
    // ....

    @Temporal(TemporalType.DATE)
    private Date createdAt;

    @Temporal(TemporalType.DATE)
    private Date modifiedAt;

    // ....

    @PrePersist
    void createdAt() {
        this.createdAt = new Date();
        this.modifiedAt = this.createdAt;
    }

    @PreUpdate
    void modifiedAt() {
        this.modifiedAt = new Date();
    }
}
```

ANY  
QUESTIONS  
?

**Pytania?**