Techniki O/RM

Hibernate

Agenda

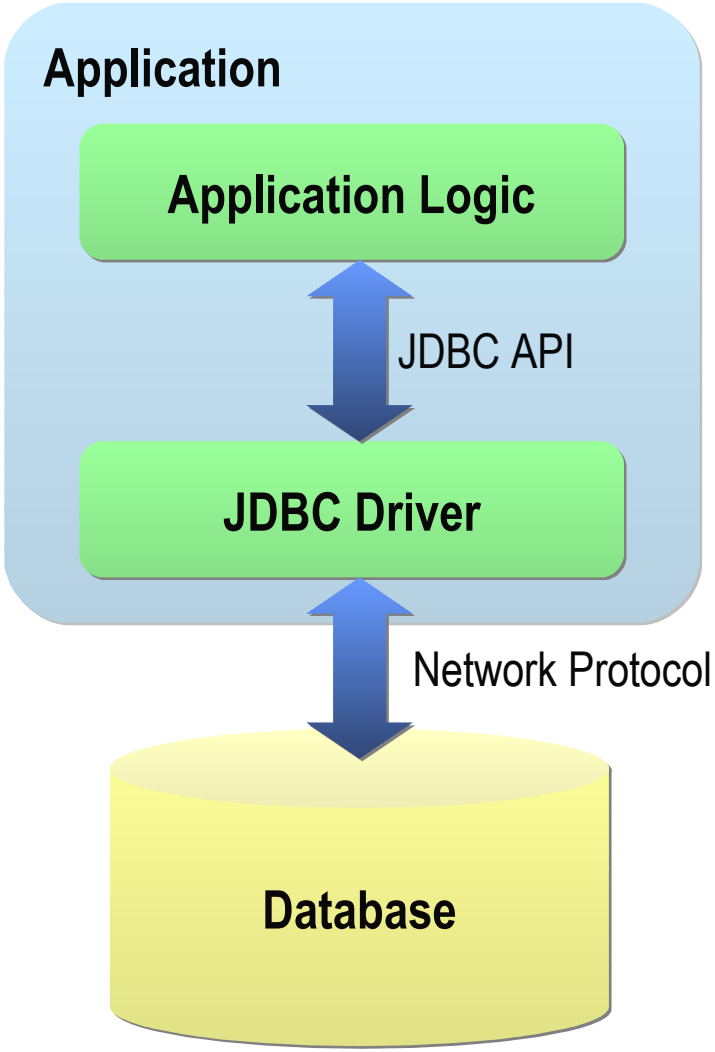
- Hibernate ogólnie
- Konfiguracja Hibernate
- Mapowanie obiektów
- Operacje odczyt / zapis
- HQL
- Sesja Hibernate

- Mapowanie asocjacji międzyobiektowych
- Mapowanie dziedziczenia
- Zagadnienia współbieżności
- Zaawansowane

Problemy JDBC

Problemy

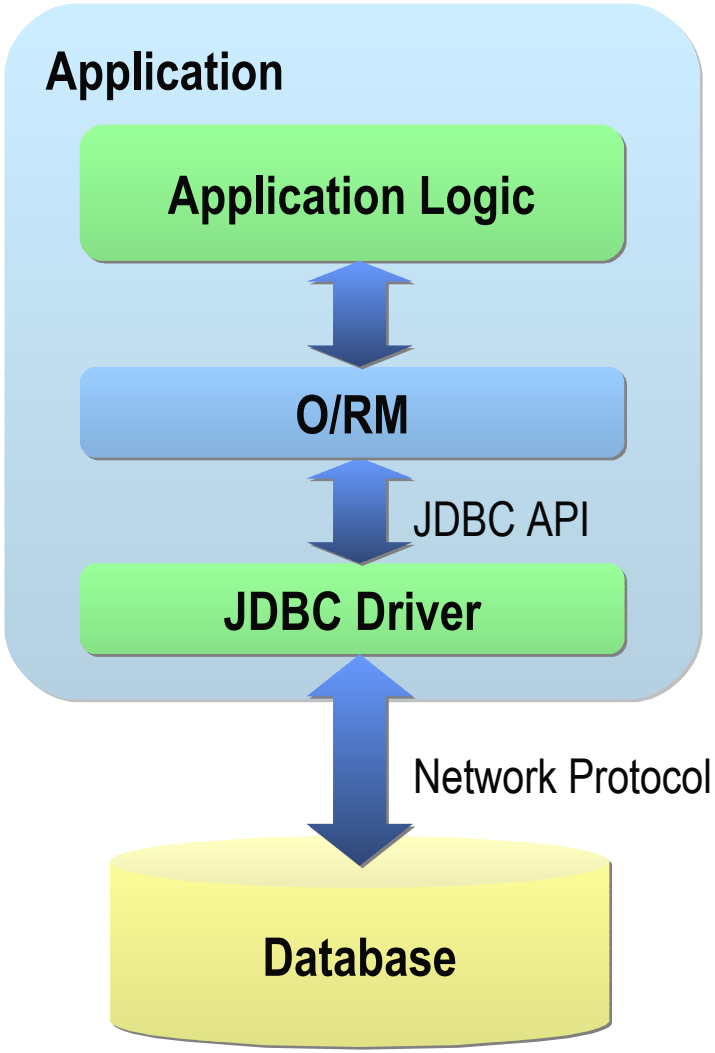
- Dużo kodu
- Łatwo popełnić błąd
- Uciążliwe zmiany
- Konwersje obiektów na postać relacyjną



O/RM – Mapowanie relacyjno-obiektowe

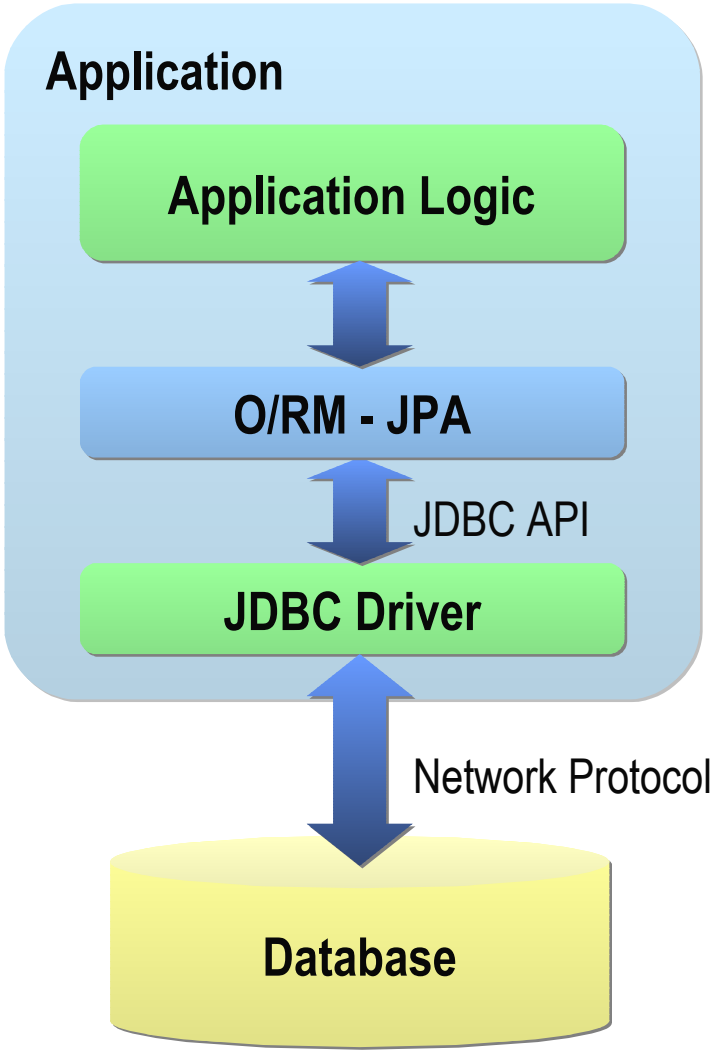
Korzyści

- Mało kodu
- Prostota
- Łatwe zmiany
- Mapowanie obiektów na tabele



O/RM - JPA

- Przeźroczysta persystencja
- Prostota konfiguracji i użytkowania
- Własny język zapytań JP-QL
- Niezależność od motoru bazy danych
- Część specyfikacji EJB 3.0
- Kilka implementacji OpenSource i komercyjnych
  - **Hibernate**
  - Eclipselink / TopLink
  - OpenJPA / Kodo
  - DataNucleus / JPOX

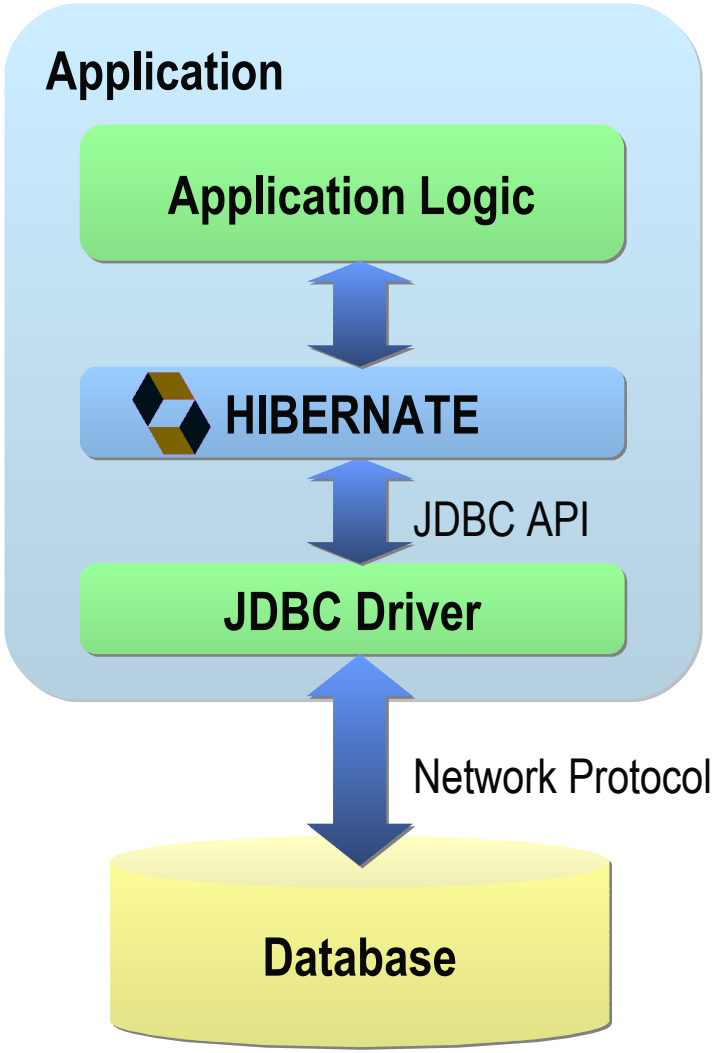


Hibernate

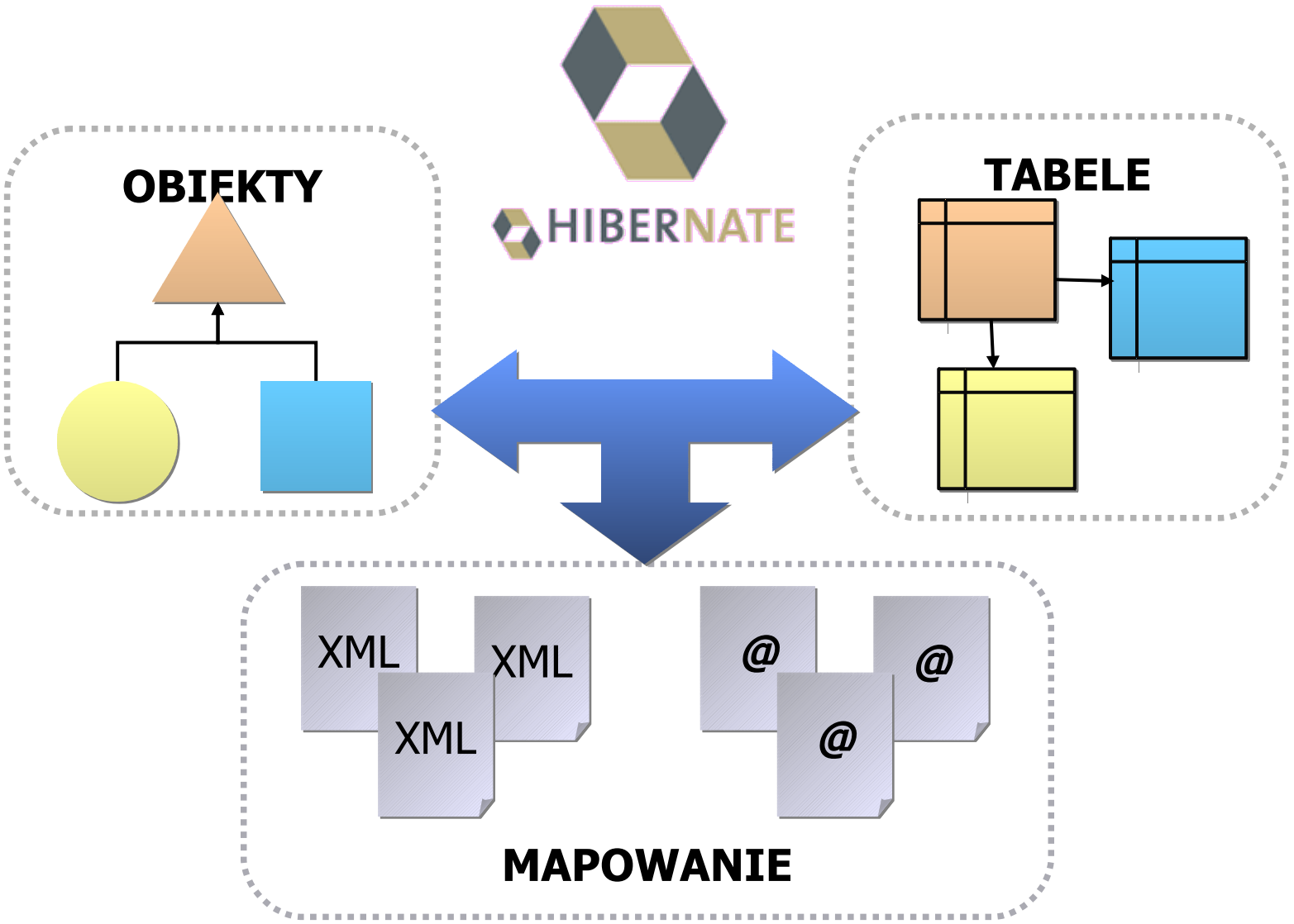
- Klasyk / Standard de-facto
- Podstawa specyfikacji JPA
- Historia:
  - Rozpoczęty przez Gavin'a King'a w 2001 roku
  - Przejęty przez JBoss'a
  - Przejęty przez RedHat'a

- Benchmarki pokazują że TopLink / EclipseLink jest szybszy

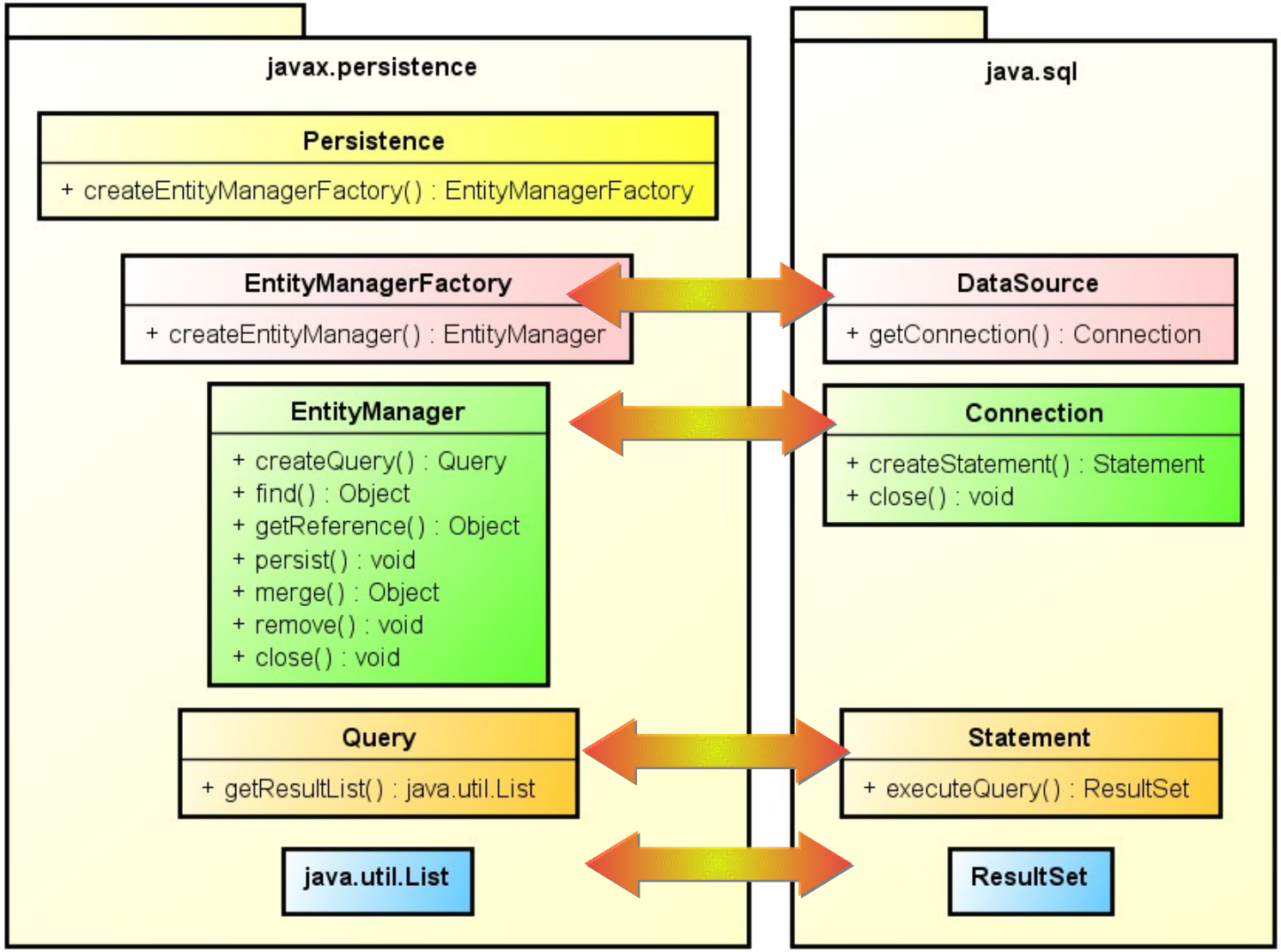
- Ma więcej funkcji niż przewiduje specyfikacja JPA



Hibernate



Podobieństwo API JPA <-> JDBC



Hibernate / JPA - Konfiguracja

Właściwość	Znaczenie
javax.persistence.jdbc.driver	klasa jdbc driver
javax.persistence.jdbc.url	jdbc URL
javax.persistence.jdbc.user	użytkownik bazy danych
javax.persistence.jdbc.password	hasło użytkownika
hibernate.dialect	rodzaj użytej bazy danych
hibernate.hbm2ddl.auto	automatyczna generacja DDL
hibernate.show_sql	pokazuje wykonywany SQL alternatywnie ustawieć DEBUG na logger'ze ***org.hibernate.SQL***

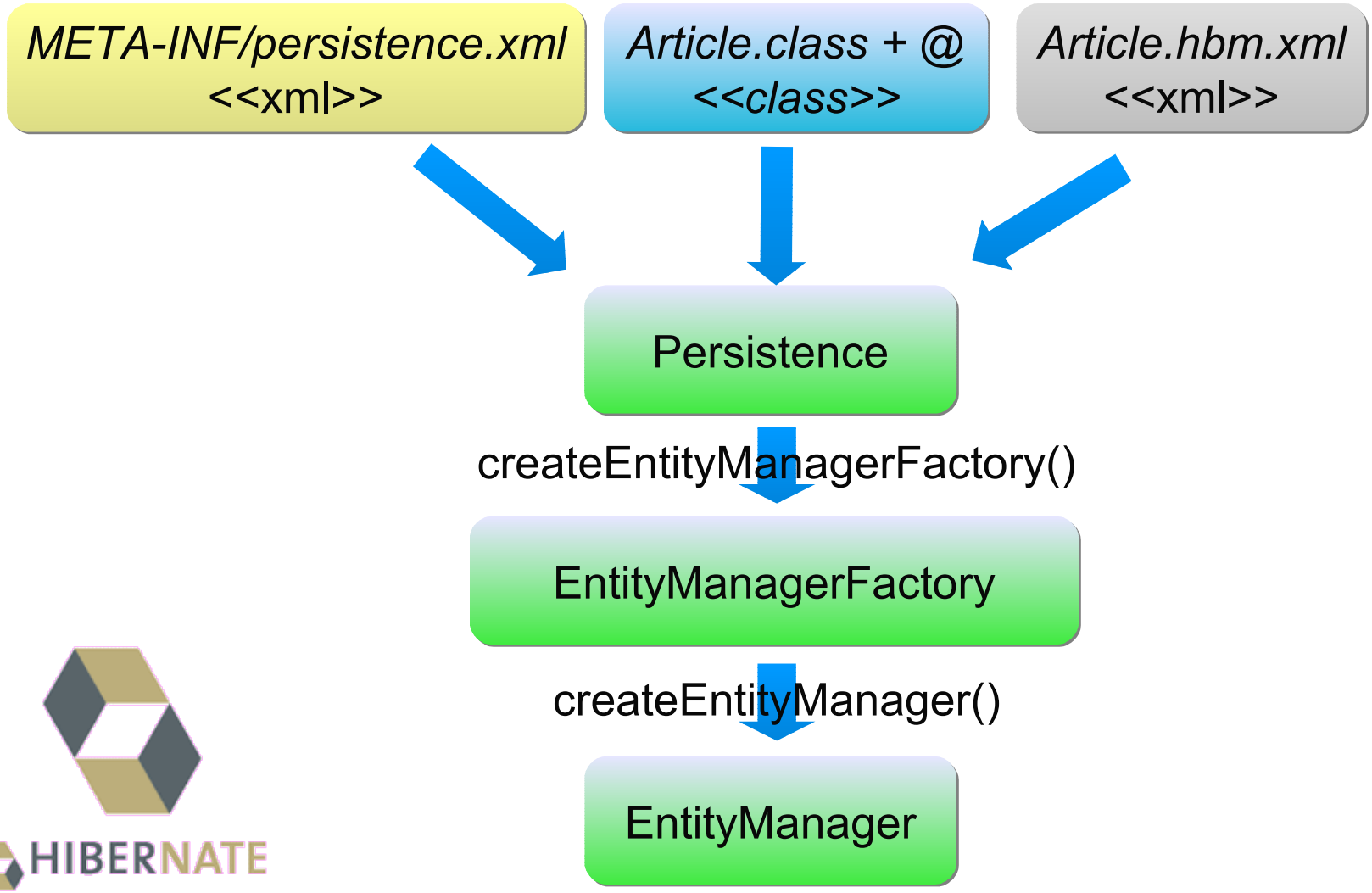
META-INF/persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="microcms">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />

      <property name="javax.persistence.jdbc.password" value="mysql" />
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost/microcms" />

      <property name="javax.persistence.jdbc.user"      value="root" />
      <property name="hibernate.hbm2ddl.auto"           value="update" />
      <property name="hibernate.show_sql"               value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

META-INF/persistence.xml

- **<class>** - dodanie jawnie klasy z adnotacjami
- **<jar-file>** - jawne wskazanie biblioteki z klasami
- **<jta-data-source>** - wskazanie na źródło danych w JNDI
- **<mapping-file>** - wskazanie lokalizacji pliku orm.xml z mapowaniami
- **<shared-cache-mode>** - włącz/wyłącz do cache'a
- **<validation-mode>** - steruje walidacjami

Konfiguracja Hibernate



Code

Konfiguracja i inicjalizacja

Mapowanie O/RM

- obiekt – entity
- obiekt - component / embeddable
- właściwość / pole
- referencja
- kolekcja (list/set/map)
- iteracja
- obiekt - immutable entity

- rekord z kluczem głównym
- zestaw pól w tabeli

- pole
- klucz obcy

- kursor
- widok

- procedura składowana

Mapowanie klas - zalecenia

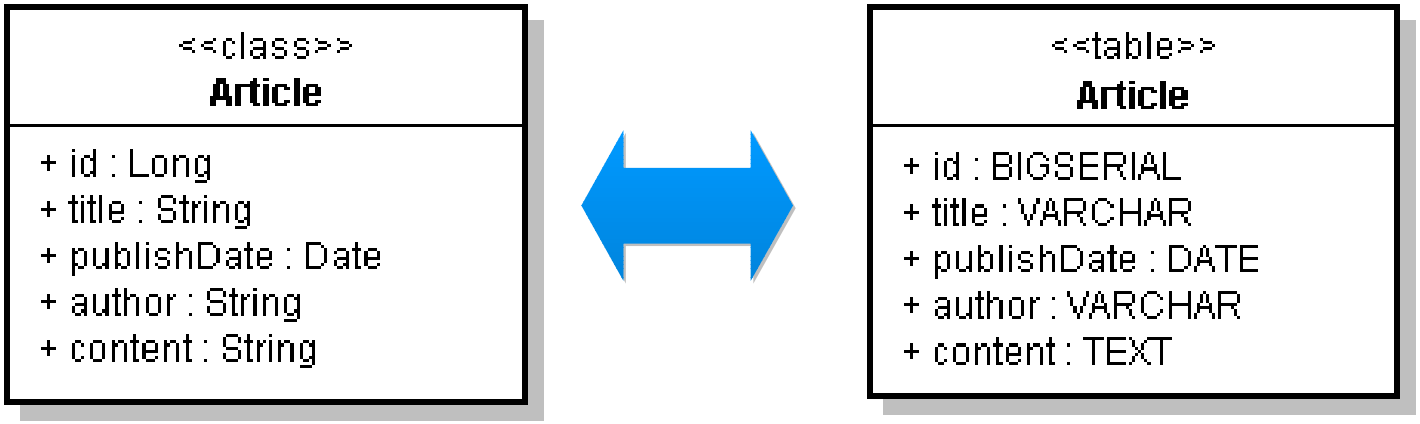
- Pusty domyślny konstruktor – może być ***default*** lub ***protected***

- Zaleca się **spójne używanie**
  - setter'ów i getter'ów – mogą być nawet ***private***
  - lub persystencję bezpośrednio na polach

- Zaleca się wyodrębnienie technicznego klucza głównego i stworzenie pola / property które będzie go zawierać

Baza wiadomości

- **id –** Identyfikator
- **title –** Tytuł
- **publishDate –** Data publikacji
- **author –** Autor
- **content –** Treść



Mapowanie typów Java <-> SQL

String, Character, char	• VARCHAR, CHAR, TEXT, CLOB
Boolean, boolean	• NUMBER, NUMERIC, CHAR, BOOL, BOOLEAN
Byte, byte	• NUMBER, NUMERIC, TINYINT
Short, short	• NUMBER, NUMERIC, SMALLINT
Integer, int,	• NUMBER, NUMERIC, INT, INTEGER
Long, long	• NUMBER, NUMERIC, BIGINT
Double, double, Float, float	• NUMERIC, FLOAT, DOUBLE
BigInteger, BigDecimal	• NUMBER, NUMERIC
Date, Calendar	• DATE, DATETIME, TIMESTAMP
Serializable, byte[]	• BLOB, BINARY, BYTEA

Article.java

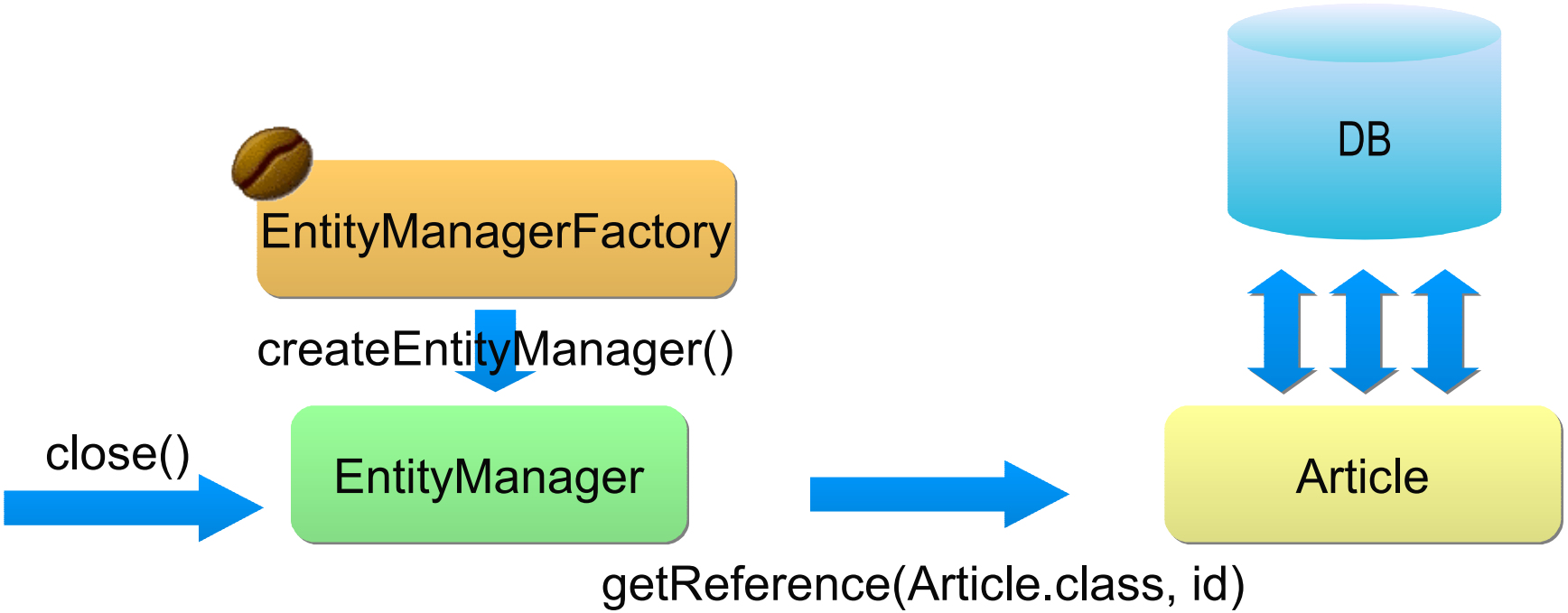
```
@Entity
public class Article {

    @Id private Long id;

    private String title;

    private String content;

    private String author;

    private Date publishDate;

    // metody ....

}
```

```
@Entity
public class Article {

    private Long id;

    private String title;

    private String content;

    private String author;

    private Date publishDate;

    @Id public Long getId() {
        return id;
    }

  // reszta setterów i getterów
}
```

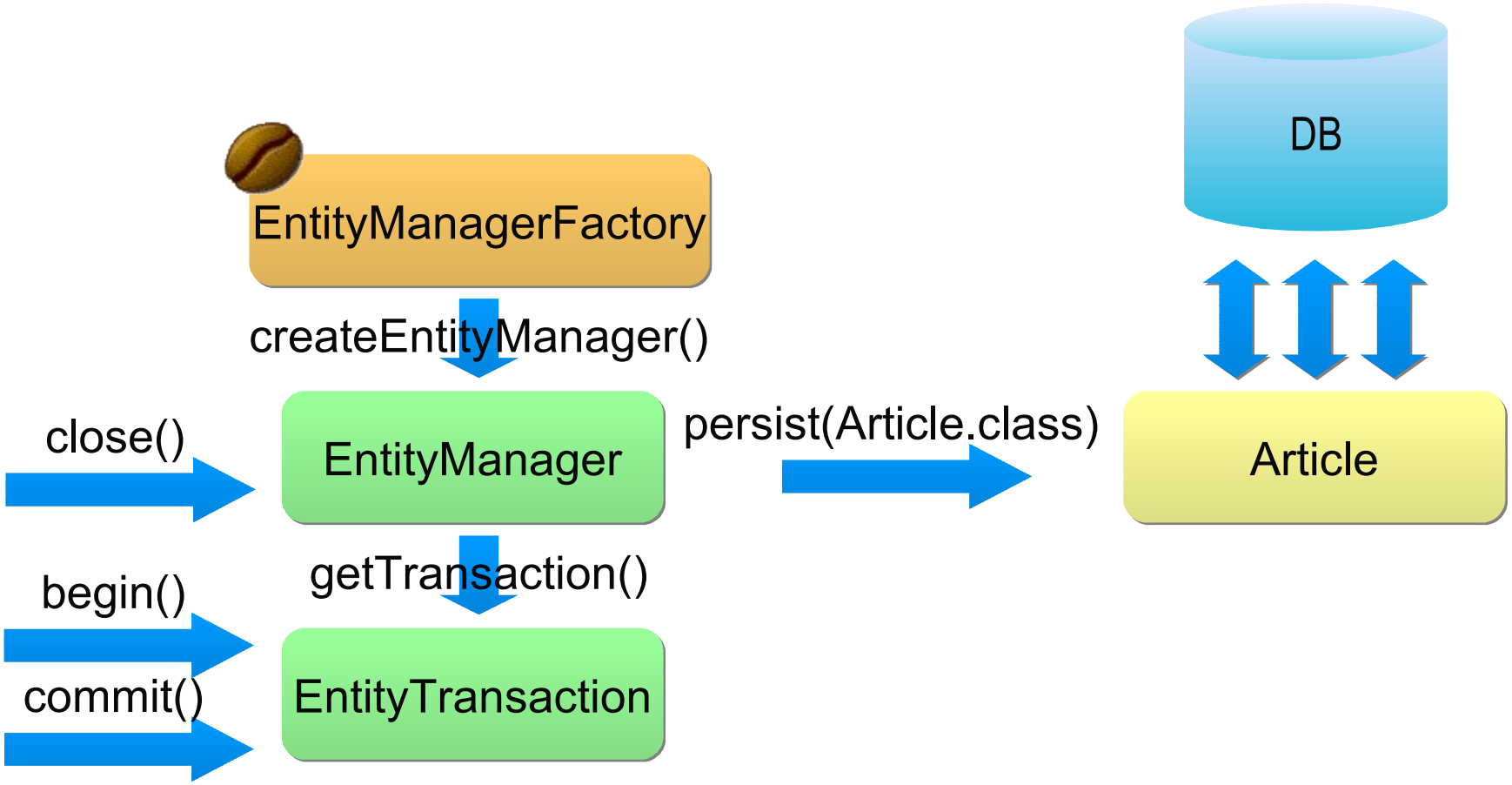
Code

Article.java

Test odczytu



Test zapisu


DB
EntityManagerFactory
createEntityManager()
close()
EntityManager
persist(Article.class)
Article
getTransaction()
begin()
commit()
EntityTransaction

Generacja identyfikatorów przez JPA / Hibernate

```
@Id
@GeneratedValue
private Long id;
```

```
@GeneratedValue(
    strategy=AUTO | TABLE | SEQUENCE | IDENTITY
    generator="MYGENERATOR"
)
```

Generacja identyfikatorów przez Hibernate

@org.hibernate.annotations.GenericGenerator

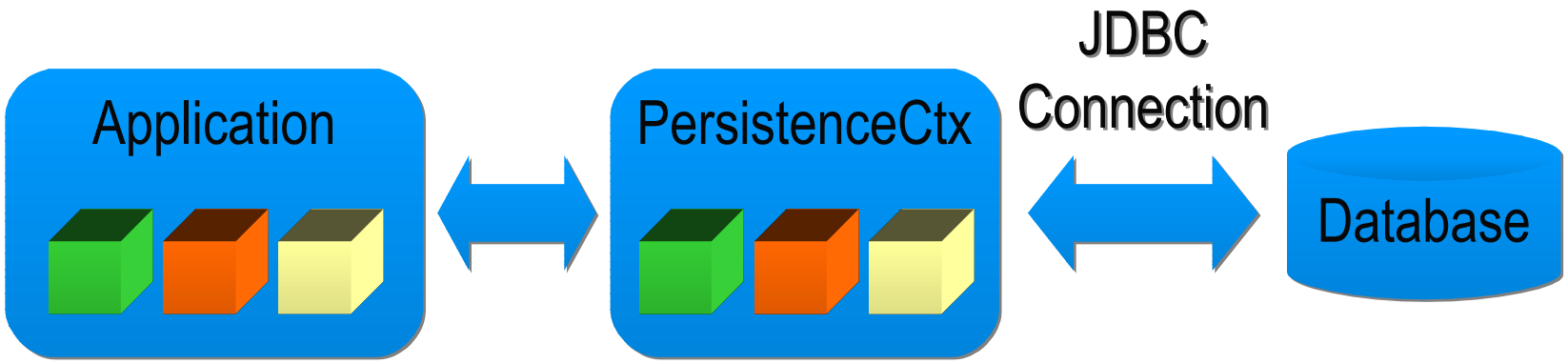
- **increment** – wewnętrzny licznik Hibernate (nie stosować)
- **identity** – dla pól typu AUTO_INCREMENT albo IDENTITY - ***IDENTITY***
- **sequence** – używa wskazanej sekwencji
- **hilo** – algorytm HI/LO oparty o tabelę w bazie danych - ***TABLE***
- **seqhilo** – algorytm HI/LO oparty o sekwencję - ***SEQUENCE***
- **uuid** – generacja unikalnego 8-bajtowego identyfikatora przez Hibernate (String)
- **guid** – generacja unikalnego identyfikatora przez bazę danych (GUID)
- **native** – automatyczny wybór najlepszy dla danej bazy danych - ***AUTO***
- **assigned** – aplikacja sama będzie przypisywać identyfikatory
- **select** – trigger bazodanowy, hibernate będzie wykonywał select do bazy danych
- **foreign** – używa klucza obcego do innego obiektu

Code

Testy generatora

PersistenceContext – Cache pierwszego poziomu

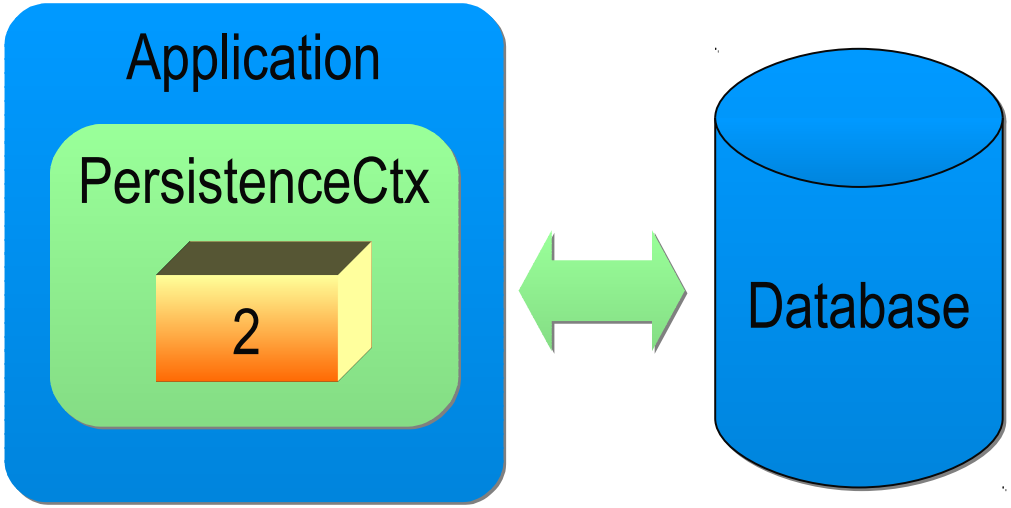
PersistenceContext zachowuje się jak cache.



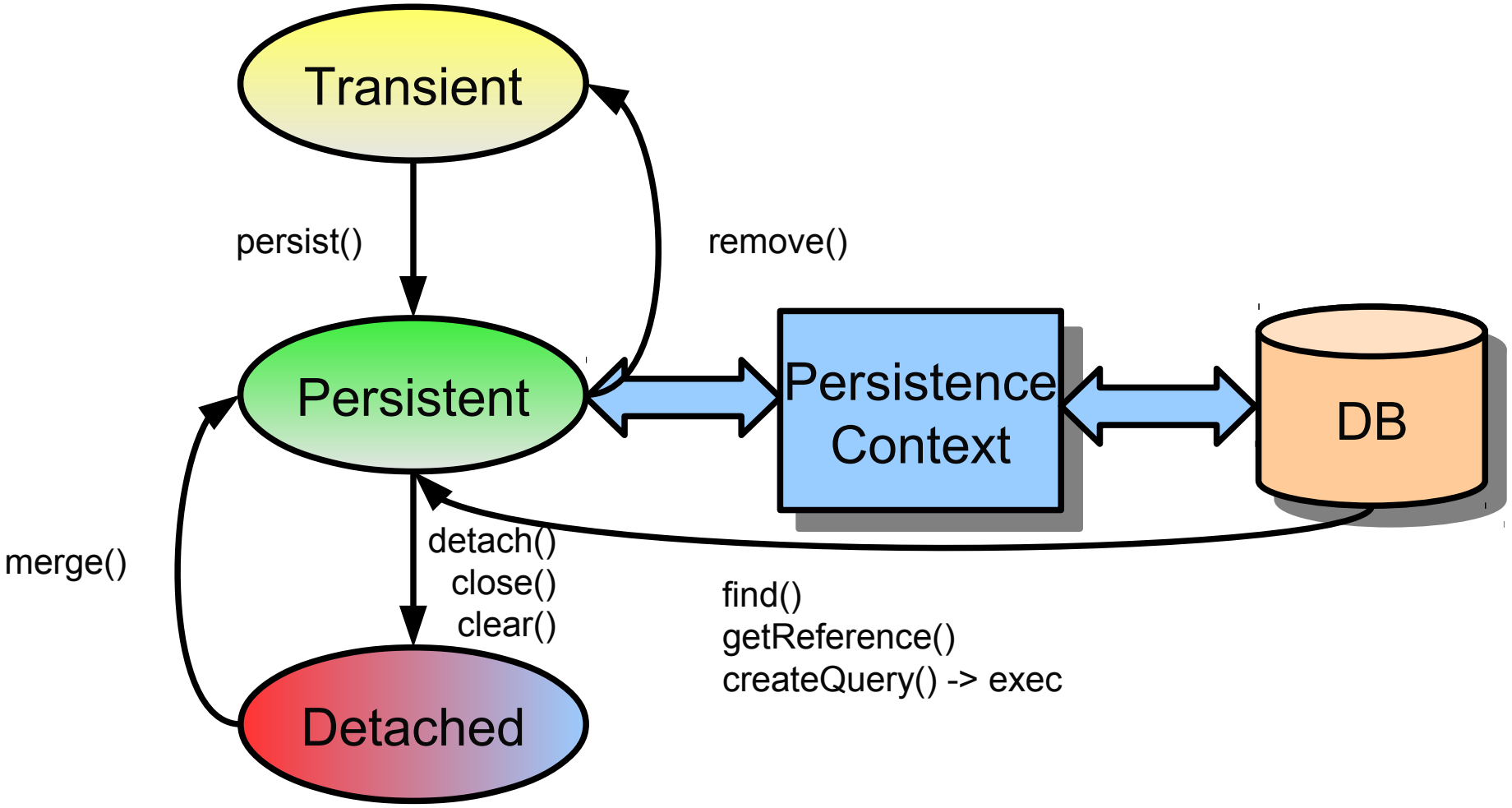
PersistenceContext – Opóźnione zapisy

Modyfikacje obiektów skutkują automatycznym zapisem stanu do bazy.
Hibernate realizuje funkcjonalność opóźnionych zapisów.

1. createEntityManager()
2. obj = find() => SELECT
3. obj.setValue(„2”)
4. commit()
    4a. flush() => UPDATE
    4b close()



Stany obiektów



Transient

- nowe obiekty (stworzone przez **new**())
- jeszcze nie skojarzone z żadnym PersistenceContextem
- nie skojarzone z rekordem w bazie danych
- nie posiadają identyfikatora

Persistent

- posiadaja identyfikator
- skojarzony z rekordem (lub będzie)
- załadowany z bazy przez EntityManager'a
- zapisany do bazy przez EntityManager'a
- skojarzony z PersistenceContext
- zmiany w tym obiekcie będą automatycznie zapisywane do skojarzonego rekordu

Detached

- posiadają identyfikator
- skojarzone z rekordem w bazie danych
- załadowany z bazy przez EntityManager'a
- zapisany do bazy przez EntityManager'a
- odłączony od PersistenceContext'u
- zmiany w tym obiekcie będą nie będą automatycznie zapisywane do skojarzonego rekordu
- można go ponownie podłączyć do PersistenceContext'u

Operacje na obiektach

- **persist**(transient -> persistent) : void
  ~ INSERT
- **merge**(detached -> detached) : persistent
  ~ UPDATE
- **remove**(transient | persistent -> transient) : void
  ~ DELETE

Operacje ładowania z bazy

- **find**(id) : persistent or null
  ~ SELECT
- **getReference**(id) : persistent or exception
  ~ SELECT
- **refresh**(persistent -> persistent) : void
  ~ SELECT
- **lock**(detached | persistent -> persistent) : void
  ~ SELECT [FOR UPDATE]

Dodatkowe operacje

- **detach**(persistent -> transient) : void
- **clear**() : void
- **close**() : void

Zapytania w Hibernate

- **HQL / JPQL**
  - podobne do SQL, łatwo się nauczyć
  - dobre gdy zapytania są z góry znane
- **Criteria API**
  - bogate i złożone API oparte o koncepcje DSL
  - dobre gdy trzeba dynamicznie budować zapytania
- **SQL**
  - kiedy chcemy ominąć tłumaczenie HQL -> SQL
  - kiedy chcemy ręcznie optymalizować zapytania
  - możemy nawet zastosować procedury składowane

HQL a SQL

HQL/JPQL i SQL są do siebie bardzo podobne. SQL operuje na tabelach, HQL/JPQL na obiektach.

[SELECT ....]
FROM ....
[WHERE ....]
[ORDER BY ...]
[GROUP BY ...]

HQL/JPQL – Przykład – zapytania obiektowe

- **SELECT** a **FROM** *Article* a;
- **SELECT** a **FROM** *Article* a **WHERE** a.publishDate<?;
- **SELECT** a **FROM** *Article* a **WHERE** a.publishDate<? **ORDER BY** a.title;

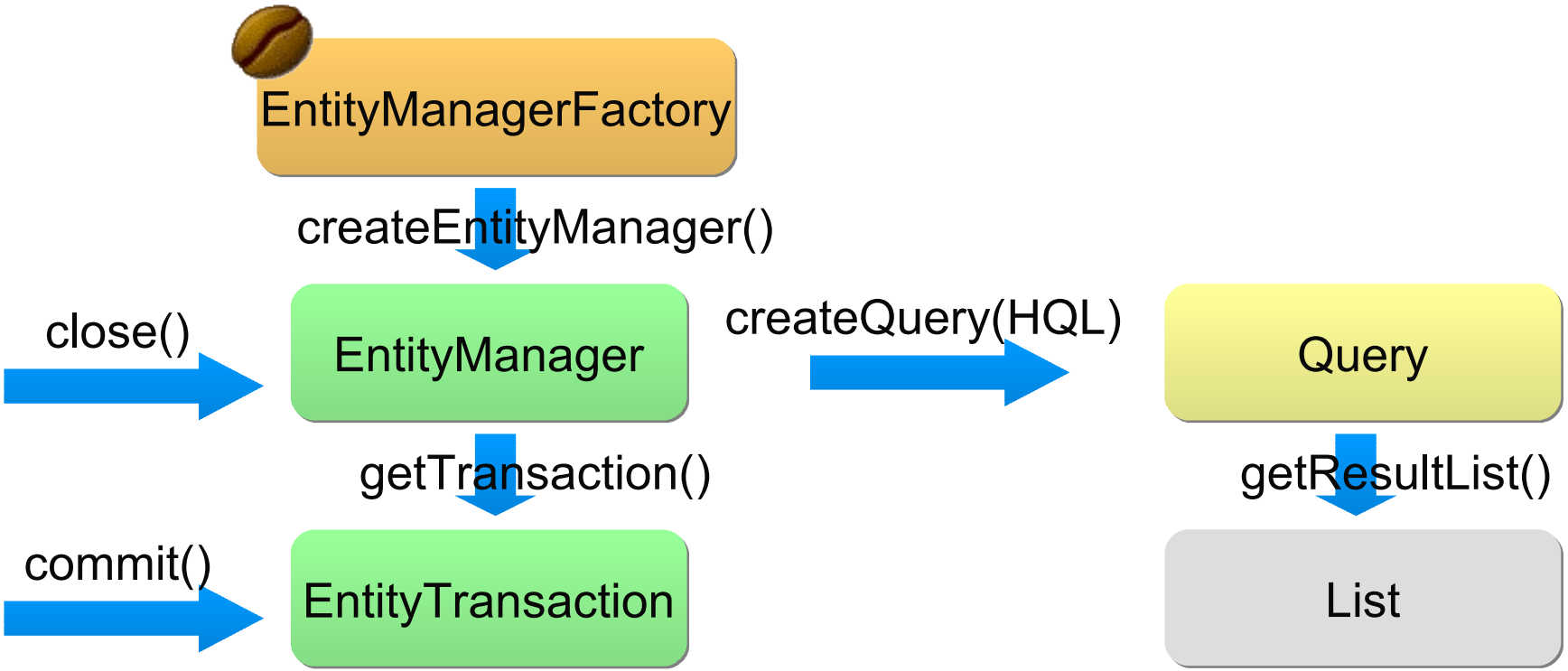
- **List<Article>** result =
  em.createQuery("....").getResultList();

HQL/JPQL – Przykład – zapytania skalarne

- **SELECT** a.title, a.publishDate **FROM** *Article* a **WHERE** a.author=? **ORDER BY** a.title;
- **SELECT** COUNT(a.title) **FROM** *Article* a **WHERE** a.author=?;
- **SELECT** COUNT(a.title), a.author **FROM** *Article* a **GROUP BY** a.author**;**

- **List<Object[]>** result =
  em.createQuery("....").getResultList();

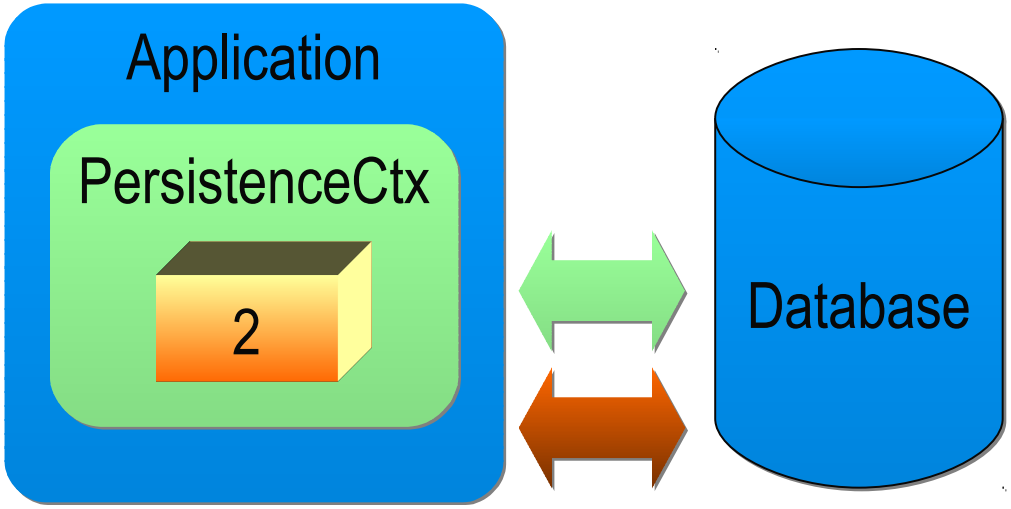
Hibernate – HQL/JPQL Query



PersistenceCtx – Opóźnione zapisy

Modyfikacje obiektów skutkują automatycznym zapisem stanu do bazy.
Hibernate realizuje funkcjonalność opóźnionych zapisów.

1. createEntityManager()
2. obj = find() => SELECT
3. obj.setValue(„2”)
4. query(„SELECT ...”)
    4a. flush() => UPDATE
    4b. SELECT



Criteria API

SELECT a **FROM** *Article* a
WHERE a.publishDate<? **ORDER BY** a.title;

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Article> q = cb.createQuery(Article.class);
Root<Article> a = q.from(Article.class);
q.where(cb.lessThan(a.<Date>get("publishDate"), date));
q.orderBy(cb.asc(a.get("title")));

List<Article> articles = em.createQuery(q).getResultList();
```

Adnotacje javax.persistence

- Podstawowe
  - *@Entity*
  - *@Id @GeneratedValue @Version*
  - *@Basic @Temporal @Enumerated @Lob*
  - *@Table @Column*
  - *@Transient @Formula*
  - *@Embedded @Embeddable*
- Asocjacje
  - *@OneToOne @ManyToOne @OneToMany*
  - *@ManyToMany*
  - *@JoinColumn @JoinTable*
  - *@MapKey @OrderColumn @OrderBy*

Adnotacje - więcej przykładów

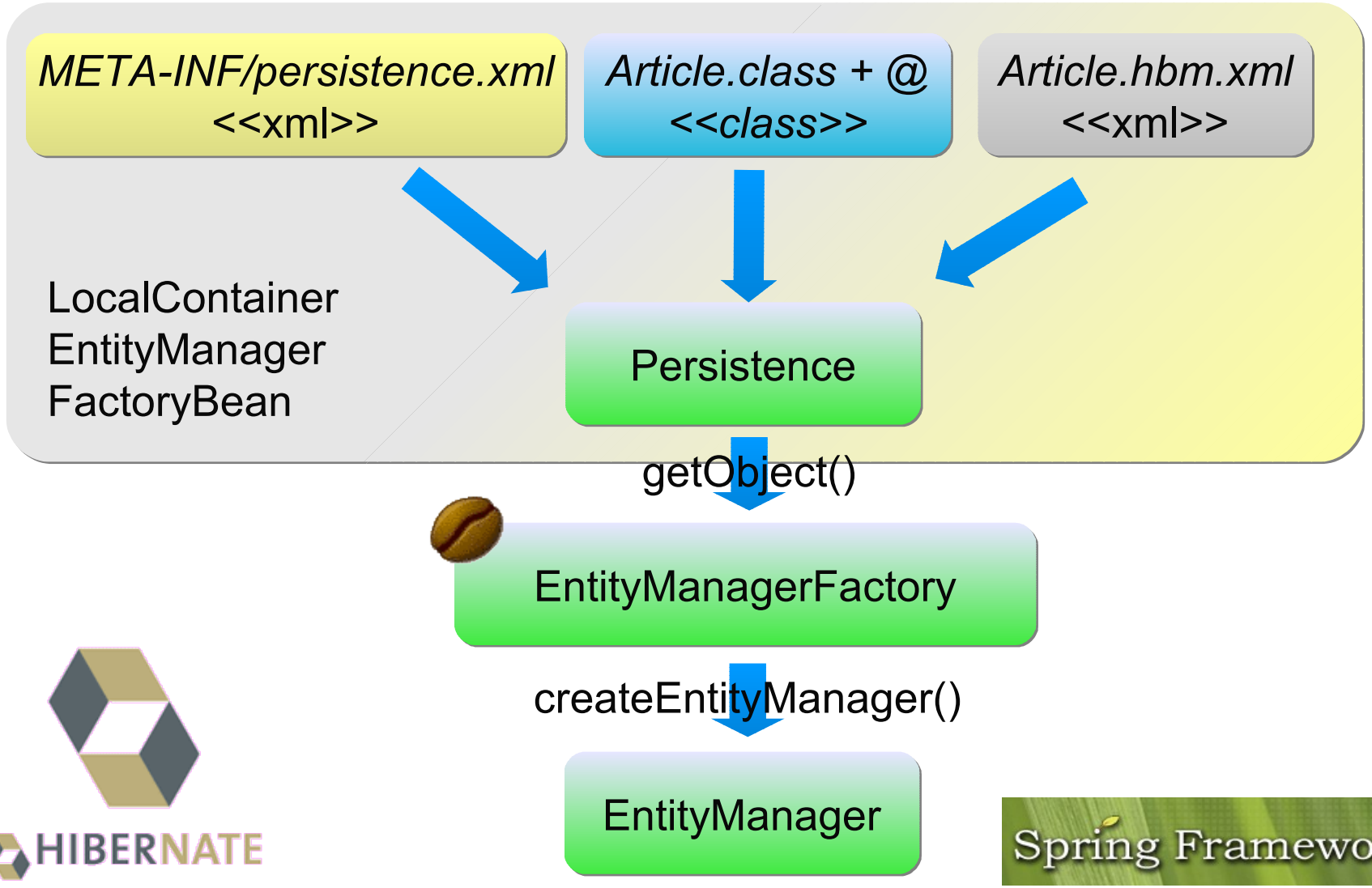
- ***@Entity(name = "ARTYKUL")***
  własna nazwa encji w HQL/JPQL
- ***@Basic(optional = false)***
  pole NOT NULL
- ***@Temporal(value=DATE)***
  pole typu DATE (bez czasu)
- ***@Temporal(value=TIMESTAMP)***
  pole typu TIMESTAMP (do sekund lub milisekund)

Adnotacje - więcej przykładów

- ***@Table(name = "ARTYKUL")***
  własna nazwa tabeli w DDL
- ***@Column(***
  - ***name = "tytul" -*** własna nazwa kolumy
  - ***unique = true -*** indeks UNIQUE na kolumnie
  - ***nullable = false -*** kolumna NOT NULL
  - ***insertable = false -*** INSERT bez kolumny
  - ***updateable = false -*** UPDATE bez kolumny
  - ***columnDefinition = "varchar(256)"***
  - ***length = "256" -*** własna długość kolumny
  - ***precision = "19" -*** DECIMAL(N, ...)
  - ***scale = "2" -*** DECIMAL(..., M)

)

Konfiguracja Hibernate przez Springframework



Konfiugracja przez Springframework

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost/microcms" />
    <property name="username" value="root" />
    <property name="password" value="mysql" />
  </bean>

  <bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="persistenceUnitName" value="microcms" />
    <property name="jpaVendorAdapter">
      <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="true" />
        <property name="generateDdl" value="true" />
      </bean>
    </property>
  </bean>
</beans>
```

Zdarzenia persystencji

- ***@PrePersist*** - przed persist()
- ***@PostPersist*** - po INSERT

- ***@PreRemove*** - przed remove()
- ***@PostRemove*** - po remove()

- ***@PreUpdate*** - przed UPDATE
- ***@PostUpdate*** - po UPDATE

- ***@PostLoad*** - po find(), getReference(), refresh() - po przejściu do stanu ***Persisted*** z bazy

Zastosowanie zdarzeń

```
@Temporal(TemporalType.TIMESTAMP)
private Date lastModified;

@PreUpdate
@PrePersist
public void updateLastModified() {
    this.lastUpdated = new Date();
}
```

Podsumowanie

- O/RM Hibernate umożliwia łatwe mapowanie obiektów na bazę danych
- Hibernate wprowadza wiele optymalizacji, które trudno byłoby zrealizować samemu w oparciu o czyste JDBC
- Semantyka transakcyjna jest kluczowa w zrozumieniu pojęcia sesji i zachowania Hibernate'a