Java Persistence API

Jeszenszky Péter
Debreceni Egyetem, Informatikai Kar
jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2020. január 21.

Fogalmak

- Perzisztencia (*persistence*)
- Adatelérési objektum (*data access object* – DAO)
- Szakterületi modell (*domain model*)
- Vérszegény szakterületi modell (*anemic domain model*)
- „Jó öreg Java objektum” (*plain old Java object* – POJO)
- JavaBean
- Objektum-relációs leképezés (*object-relational mapping* – ORM)

Perzisztencia

- Jelentése tartós fennmaradás.
- Az informatikában az olyan adatra használjuk, mely túléli a létrehozó folyamatot.

Perzisztencia megvalósítása

- A Java számos megoldást kínál perzisztencia megvalósításához:
  - Állománykezelés
  - *Java Architecture for XML Binding* (JAXB)
  - JDBC
  - Objektum sorosítás: lásd a `java.io.Serializable` interfészt https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/Serializable.html
  - …
- A továbbiakban adatok relációs adatbázisokban való tárolásával foglalkozunk.

Data access object (DAO)

- Egy adatforráshoz olyan módon való hozzáférést biztosító objektum, melynek interfésze mögött teljesen elrejtésre kerülnek az adatforrás implementációs részletei.
  - Egy interfészt definiál egy adott entitáshoz kapcsolódó perzisztencia műveletek végrehajtásához.
- Lásd:
  - Deepak Alur, Dan Malks, John Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd edition. Prentice Hall, 2003. http://corej2eepatterns.com/
    - *Data Access Object* http://corej2eepatterns.com/DataAccessObject.htm

Szakterületi modell

- Egy adott szakterület objektum modellje, mely viselkedést és adatokat is magában foglal. https://martinfowler.com/eaaCatalog/domainModel.html
  - Lásd:
    - Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

Vérszegény szakterületi modell

- Viselkedést nem tartalmazó szakterületi modell.
  - Martin Fowler szerint antiminta.
- Lásd:
  - Martin Fowler: *AnemicDomainModel*. https://www.martinfowler.com/bliki/AnemicDomainModel.html

POJO

- Rebecca Parsons, Josh MacKenzie és Martin Fowler által kitalált fogalom.
- Egy közönséges Java objektum, melyre nem vonatkoznak megszorítások.
  - Például nem implementál valamilyen előírt interfészeket.
- Lásd:
  - Martin Fowler: *POJO*. https://www.martinfowler.com/bliki/POJO.html

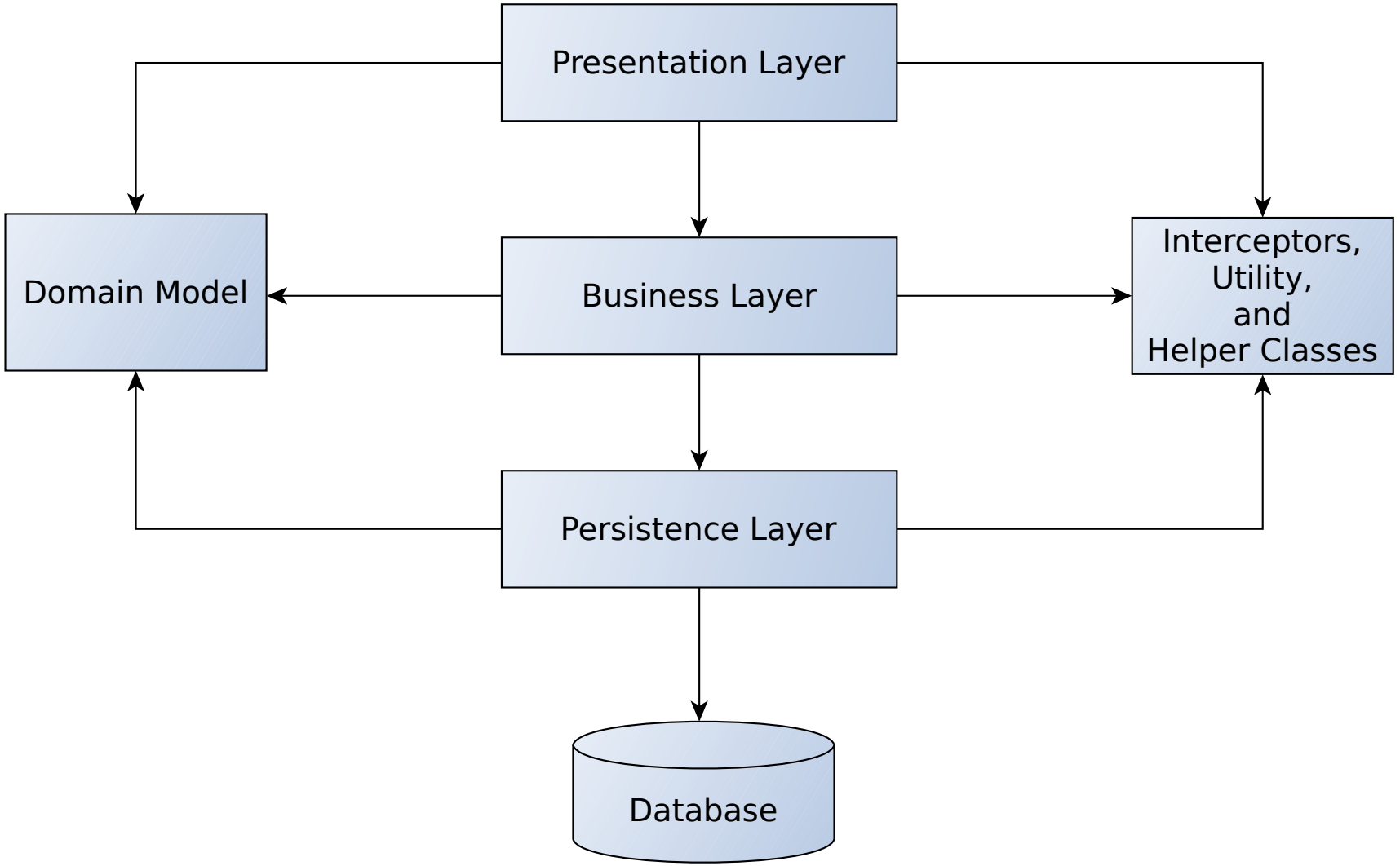
JavaBean

- A `java.io.Serializable` interfészt implementáló osztály, melynek van paraméter nélküli konstruktora, valamint lekérdező és beállító metódusokat szolgáltat a tulajdonságaihoz való hozzáféréshez.
  - Lásd:
    - *JavaBeans Specification 1.01 (Final Release)* (August 8, 1997) https://www.oracle.com/technetwork/articles/javaee/spec-136004.html
    - *The Java Tutorials – Trail: JavaBeans* https://docs.oracle.com/javase/tutorial/javabeans/
    - Stephen Colebourne. *The JavaBeans specification*. November 28, 2014. https://blog.joda.org/2014/11/the-javabeans-specification.html

Objektum-relációs leképezés

- Objektumorientált programozási nyelvek objektumai és relációs adatbázis táblák közötti konverziót jelent.
  - Szakterületi modellek objektumainak relációs adatbázis táblákban való tárolása.

Rétegelt alkalmazás architekúra

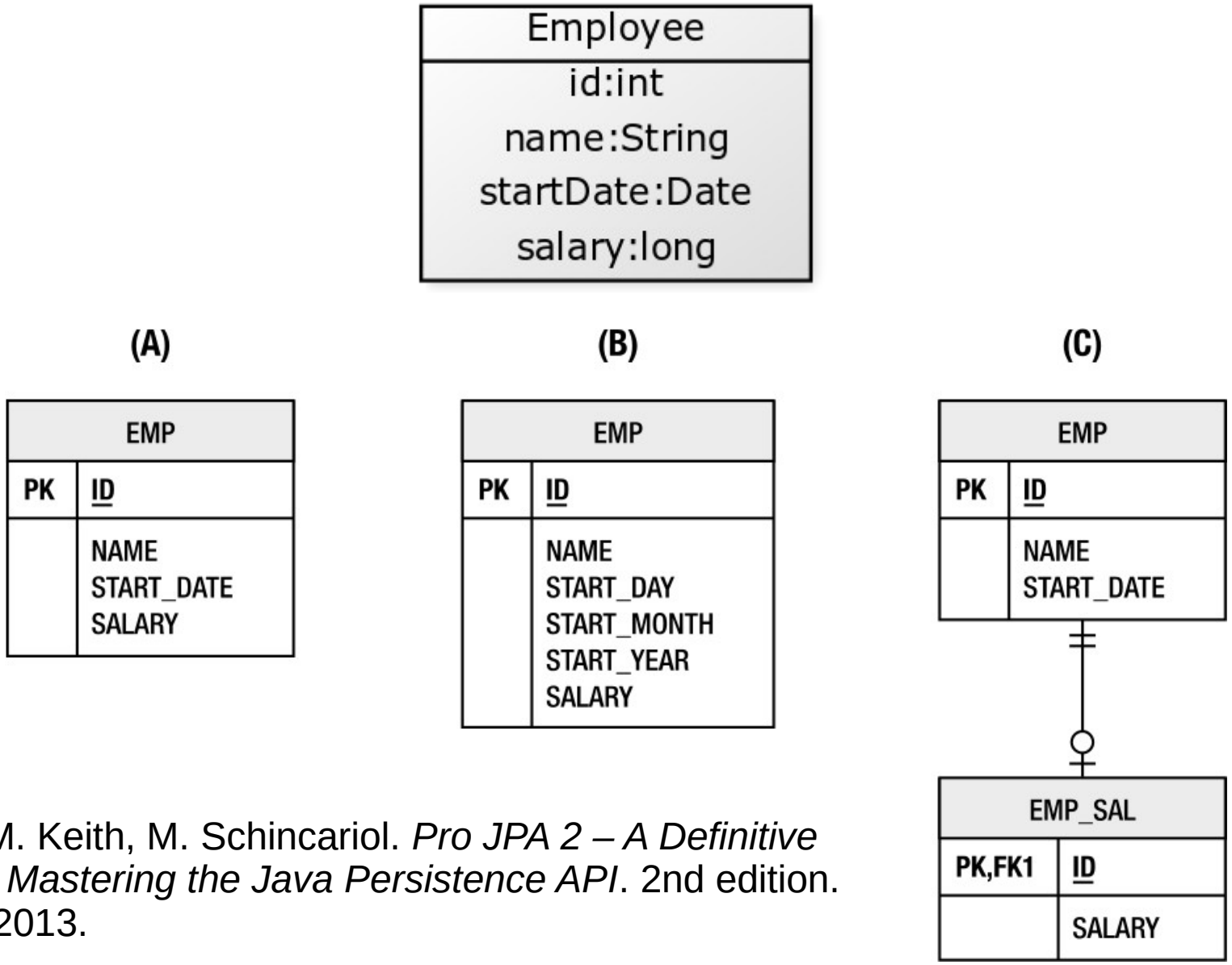


Forrás: Christian Bauer, Gavin King, Gary Gregory. *Java Persistence with Hibernate*. 2nd edition. Manning, 2015.

Paradigma ütközés

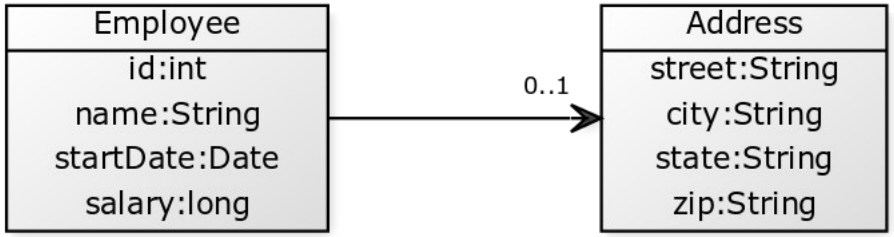
- *Object-relational impedance mismatch*:
  - A két világ között eltérésekből fakadó nehézségeket jelenti.
  - Eltérő fogalmak használata, melyek nem mindegyikének van megfelelője a másik világban.
    - Példa:
      - Asszociációk kezelése: *N*:*M* kapcsolatok ábrázolásához kapcsolótábla szükséges, mely közvetlenül nem jelenik meg az objektum modellben.

Objektum-relációs leképezés megvalósítása (1)

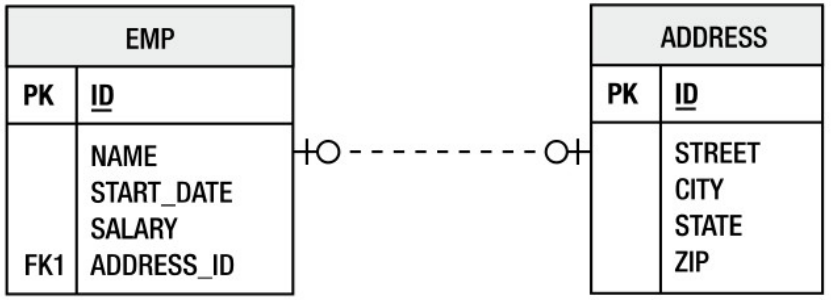


Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

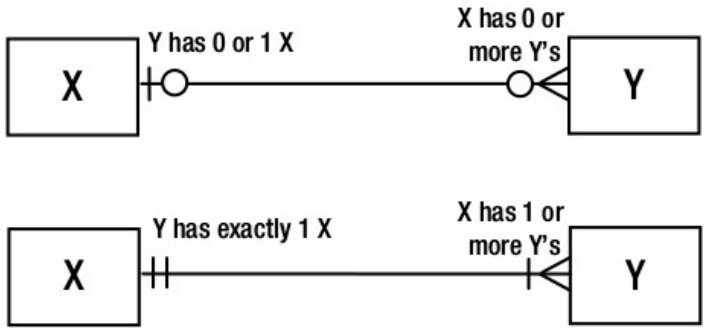
Objektum-relációs leképezés megvalósítása (2)



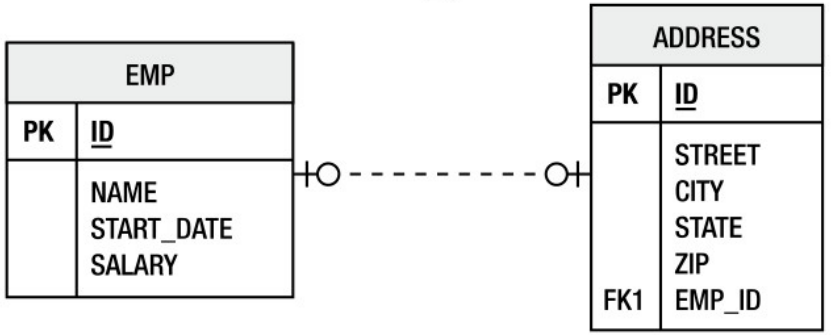
(A)



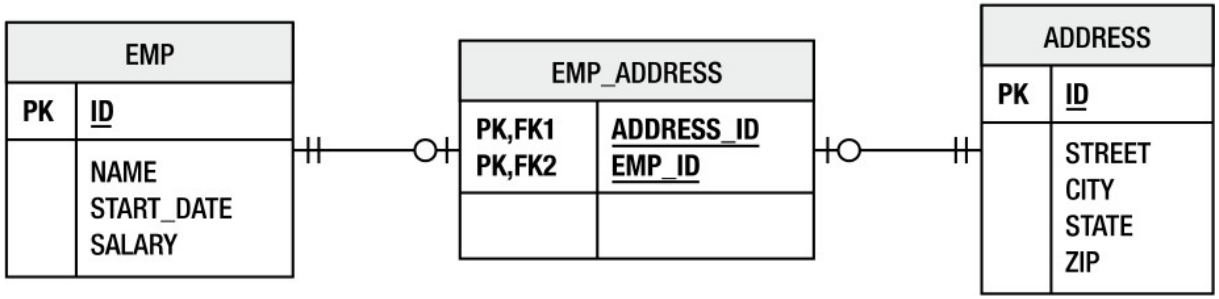
Legend:



(B)

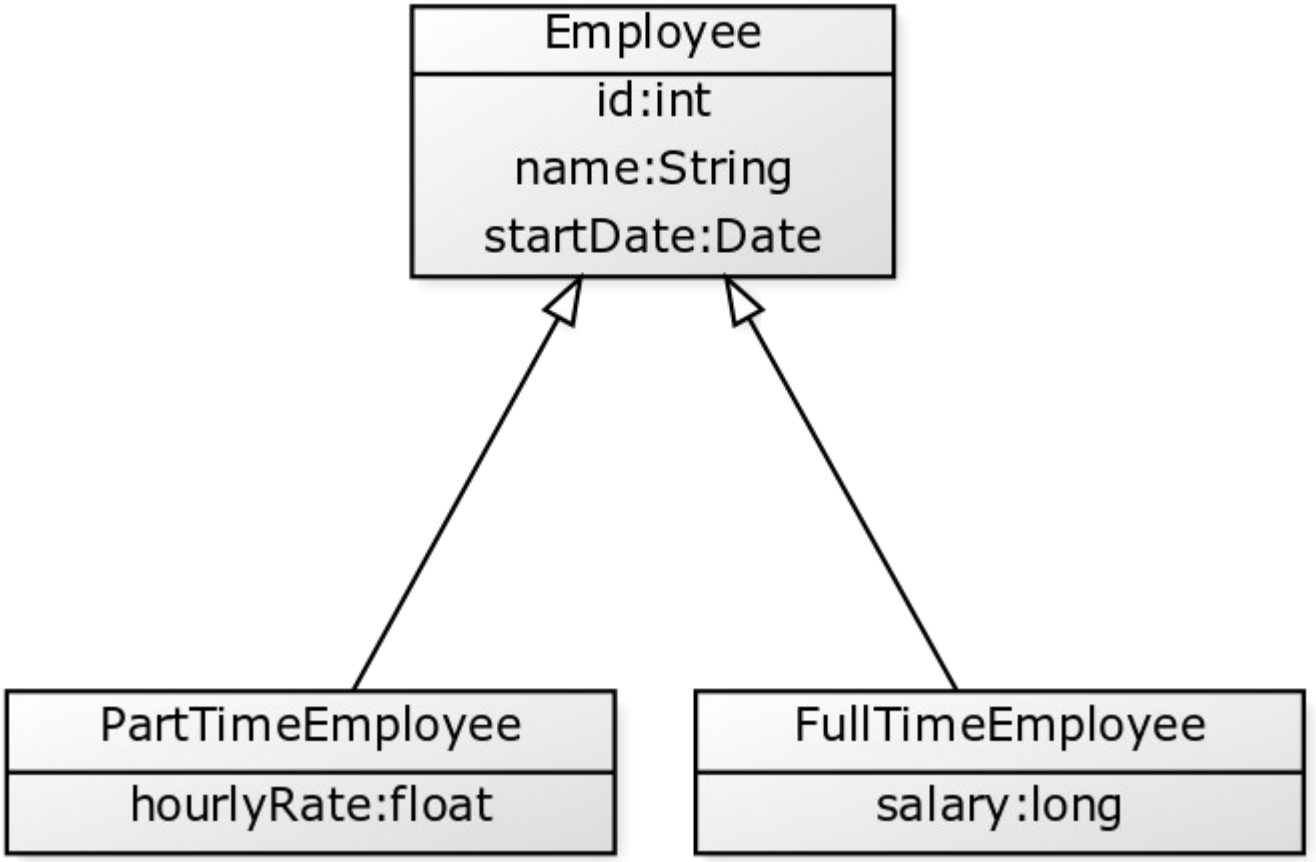


(C)



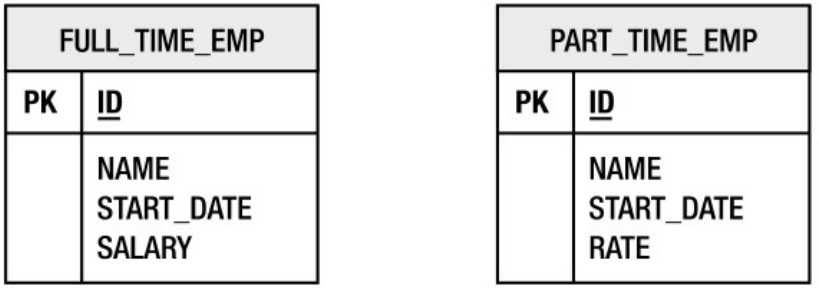
Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

Objektum-relációs leképezés megvalósítása (3)

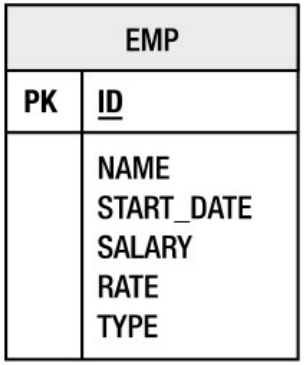


Objektum-relációs leképezés megvalósítása (4)

(A)

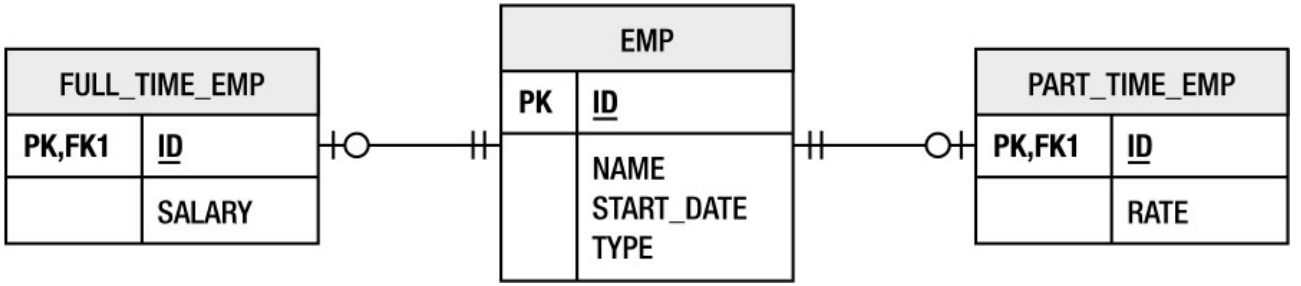


(B)



Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

(C)



Korábbi megoldások Java objektum perzisztencia megvalósítására (1)

- **JDBC**
- **Gyártófüggő megoldások**:
  - Szabad és nyílt forrású szoftverek:
    - *Hibernate ORM* (licenc: LGPLv2.1) https://hibernate.org/orm/
  - Nem szabad szoftverek:
    - *Oracle TopLink* https://www.oracle.com/middleware/technologies/top-link.html

Korábbi megoldások Java objektum perzisztencia megvalósítására (2)

- **Adat leképezők** (***data mappers***):
  - Részleges megoldást jelentenek félúton a JDBC és a teljes ORM megoldások között, ahol az alkalmazásfejlesztő szolgáltatja az SQL utasításokat.
    - Lásd:
      - Martin Fowler: *Data Mapper*. https://martinfowler.com/eaaCatalog/dataMapper.html
      - Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
  - Szabad és nyílt forrású szoftverek:
    - *Jdbi* (licenc: *Apache License 2.0*) http://jdbi.org/
    - *MyBatis* (licenc: *Apache License 2.0*) http://www.mybatis.org/mybatis-3/

Korábbi megoldások Java objektum perzisztencia megvalósítására (3)

- ***Java Data Objects*** (***JDO***):
  - Specifikáció:
    - *JSR 243: Java Data Objects 3.0 (Maintenance Release 3)* (April 9, 2010) https://www.jcp.org/en/jsr/detail?id=243
  - Szabad és nyílt forrású szoftverek:
    - *Apache JDO* (licenc: *Apache License 2.0*) https://db.apache.org/jdo/
    - *DataNucleus Access Platform* (licenc: *Apache License 2.0*) http://www.datanucleus.org/

Alternatív ORM megoldások

- Szabad és nyílt forrású szoftverek:
  - *Apache Cayenne* (licenc: *Apache License 2.0*)
    https://cayenne.apache.org/
  - *Speedment* (licenc: *Apache License 2.0*)
    https://github.com/speedment/speedment

Java Persistence API (1)

- POJO-alapú ORM megoldás Java objektum perzisztencia megvalósítására.
  - Eredetileg az *Enterprise JavaBeans* (EJB) 3.0 specifikációban jelent meg 2006-ban a Java EE 5 részeként.
  - Az aktuális verzió a 2.2 számú, mely 2017-ben jelent meg.

Java Persistence API (2)

- Az alábbi csomagok tartalmazzák (Java EE 8, Jakart EE 8):
  - `javax.persistence`
    https://javaee.github.io/javaee-spec/javadocs/javax/persistence/package-summary.html
    https://jakarta.ee/specifications/platform/8/apidocs/javax/persistence/package-summary.html
  - `javax.persistence.criteria`
    https://javaee.github.io/javaee-spec/javadocs/javax/persistence/criteria/package-summary.html
    https://jakarta.ee/specifications/platform/8/apidocs/javax/persistence/criteria/package-summary.html
  - `javax.persistence.metamodel`
    https://javaee.github.io/javaee-spec/javadocs/javax/persistence/metamodel/package-summary.html
    https://jakarta.ee/specifications/platform/8/apidocs/javax/persistence/metamodel/package-summary.html
  - `javax.persistence.spi`
    https://javaee.github.io/javaee-spec/javadocs/javax/persistence/spi/package-summary.html

Specifikáció

- *JSR 338: Java Persistence 2.2 (Maintenance Release)* (July 17, 2017) https://jcp.org/en/jsr/detail?id=338
  - Főbb újdonságok:
    - Számos annotáció típus – például `AttributeOverride`, `JoinColumn`, `NamedQuery` – ismételhetőnek jelölése.
    - A `java.time` csomag alábbi típusainak támogatása:
      - `java.time.LocalDate`
      - `java.time.LocalTime`
      - `java.time.LocalDateTime`
      - `java.time.OffsetTime`
      - `java.time.OffsetDateTime`
    - A `Stream getResultStream()` alapértelmezett metódus hozzáadása a `javax.persistence.Query` interfészhez, a `Stream<`*X*`> getResultStream()` metódus hozzáadása a `javax.persistence.TypedQuery` interfészhez.
- *Jakarta Persistence 2.2* https://jakarta.ee/specifications/persistence/2.2/

Jellemzők

- **POJO-alapú**
- **Nem tolakodás** (***non-intrusiveness***)
  - A perzisztencia API elkülönül a perzisztens osztályoktól.
    - Az alkalmazás üzleti logikája hívja meg, paraméterként adja át az API-nak a perzisztens objektumokat.
- **Objektumorientált lekérdező nyelv**
  - Java Persistence Query Language (JPQL)
- **Mobil entitások**
  - A perzisztens objektumok egy Java virtuális gépből egy másikba vihetőek át.
- **Egyszerű konfigurálás**
  - Alapértelmezések használata, az ezektől való eltérésnél szükséges csak konfigurálás (*configuration by exception*).
  - A leképezéshez metaadatok annotációkkal adhatóak meg vagy külsőleg XML dokumentumokban.
- **Nem szükséges hozzá alkalmazásszerver**
  - Java SE alkalmazásokhoz is használható.

JPA 2.2 megvalósítások

- Szabad és nyílt forrású szoftverek:
  - *Apache OpenJPA* (licenc: *Apache License 2.0*) http://openjpa.apache.org/
  - *DataNucleus Access Platform* (licenc: *Apache License 2.0*) http://www.datanucleus.org/
  - *EclipseLink (licenc: Eclipse Public License v1.0/Eclipse Distribution License v1.0)* http://www.eclipse.org/eclipselink/
  - *Hibernate ORM* (licenc: LGPLv2.1) http://hibernate.org/
    - Ezt használja alapértelmezésben perzisztencia szolgáltatóként a *WildFly* alkalmazásszerver.
      - Lásd: *WildFly Developer Guide – JPA Reference Guide* http://docs.wildfly.org/18/Developer_Guide.html#JPA_Reference_Guide

JPA támogatás NoSQL adatbázisokhoz

- Szabad és nyílt forrású szoftverek:
  - *EclipseLink* (licenc: *Eclipse Public License v1.0/Eclipse Distribution License v1.0*) https://www.eclipse.org/eclipselink/
    - Lásd: *EclipseLink NoSQL* http://www.eclipse.org/eclipselink/documentation/2.7/concepts/nosql.htm
  - *Hibernate OGM* (licenc: LGPLv2.1) http://hibernate.org/ogm/
  - *Kundera* (licenc: *Apache License 2.0*) https://github.com/Impetus/Kundera
- Nem szabad szoftverek:
  - *ObjectDB* https://www.objectdb.com/

IDE támogatás

- **Eclipse**:
  - *Dali Java Persistence Tools* (licenc: *Eclipse Public License v1.0*) https://eclipse.org/webtools/dali/
    - Az *Eclipse IDE for Java EE Developers* része.
- **IntelliJ IDEA**:
  - Csak az *Ultimate* kiadásban áll rendelkezésre!
    - Lásd: *Enabling JPA Support* https://www.jetbrains.com/help/idea/enabling-jpa-support.html
- **NetBeans**:
  - *Jeddict* (licenc: *Apache License 2.0*) https://jeddict.github.io/ https://github.com/jeddict/jeddict http://plugins.netbeans.org/plugin/53057/jpa-modeler

Hibernate komponensek

- *Hibernate OGM* http://hibernate.org/ogm/
  - JPA támogatás NoSQL adatbázisokhoz (például *MongoDB*, *Neo4j*)
- *Hibernate ORM* http://hibernate.org/orm/
  - JPA implementáció
- *Hibernate Search* http://hibernate.org/search/
  - Teljes szövegű keresés megvalósítása
- *Hibernate Validator* http://hibernate.org/validator/
  - Annotáció-alapú megoldás objektumokra vonatkozó megszorítások ellenőrzéséhez
- *Hibernate Tools* http://hibernate.org/tools/
  - Fejlesztőeszközök (Eclipse bővítmények)
- …

Támogatott adatbázis-kezelő rendszerek

- Lásd az `org.hibernate.dialect` csomagot
  http://docs.jboss.org/hibernate/orm/5.4/javadocs/org/hibernate/dialect/package-summary.html
  - Microsoft SQL Server, Oracle, MySQL, PostgreSQL, Apache Derby, H2, HSQLDB, …

Rendelkezésre állás

- Elérhető a központi Maven tárolóból
  - Termékek:
    - `org.hibernate:hibernate-core`
      https://javalibs.com/artifact/org.hibernate/hibernate-core
    - `org.hibernate:hibernate-tools`
      https://javalibs.com/artifact/org.hibernate/hibernate-tools
    - `org.hibernate.validator:hibernate-validator`
      https://javalibs.com/artifact/org.hibernate.validator/hibernate-validator
    - ...

Maven támogatás

- *hibernate-tools-maven-plugin* (licenc: LGPLv2.1)
  https://github.com/hibernate/hibernate-tools/tree/master/maven
  - Elérhető a központi Maven tárolóból:
    `org.hibernate:hibernate-tools-maven-plugin`
    https://javalibs.com/artifact/org.hibernate/hibernate-tools-maven-plugin
  - Bővítmény célok:
    - `hbm2ddl`:
    - `hbm2java`:

NHibernate

- NHibernate (programozási nyelv: C#, licenc: LGPLv2.1) https://nhibernate.info/
  - A Hibernate portja a .NET platformra.

Entitás

- Egy pehelykönnyű perzisztens szakterületi objektum.

Entitás osztályok (1)

- Egy entitás osztályt a `javax.persistence.Entity` annotációval kell annotálni vagy entitásként kell jelölni az XML leíróban.
  - Rendelkeznie kell egy paraméter nélküli konstruktorral, mely `public` vagy `protected` kell, hogy legyen.
  - Felső szintű osztály kell, hogy legyen, `enum` vagy interfész nem jelölhető entitás osztályként.
  - Nem lehet `final`. A metódusaihoz és perzisztens példányváltozóihoz sem adható meg `final` módosító.

Entitás osztályok (2)

- Absztrakt és konkrét osztályok is lehetnek entitások.
- Az entitások kiterjeszthetnek entitás osztályokat és olyan osztályokat is, melyek nem entitások. Olyan osztályok is kiterjeszthetnek entitás osztályokat, melyek nem entitás osztályok.

Példányváltozók és tulajdonságok (1)

- Egy entitás perzisztens állapotát példányváltozók ábrázolják, melyek JavaBean tulajdonságoknak felelhetnek meg.
- A példányváltozókat kizárólag metódusokon keresztül érhetik el az entitás kliensei.
- Az entitás példányváltozóit a perzisztencia szolgáltató futtató környezet közvetlenül („*field access*”) vagy lekérdező/beállító metódusokon keresztül („*property access*”) éri el.
  - A példányváltozók láthatósága `private`, `protected` vagy csomagszintű kell, hogy legyen az elérés módjától függően.
  - Ha a példányváltozók elérése lekérdező/beállító metódusokon keresztül történik, ezek láthatósága `public` vagy `protected` kell, hogy legyen.

Példányváltozók és tulajdonságok (2)

- Kollekció értékű perzisztens példányváltozók és tulajdonságok típusaként a `java.util.Collection`, `java.util.List`, `java.util.Map` és `java.util.Set` interfészeket kell használni.
  - Inicializáláshoz használható az interfészeket implementáló kollekció osztályok bármelyike.

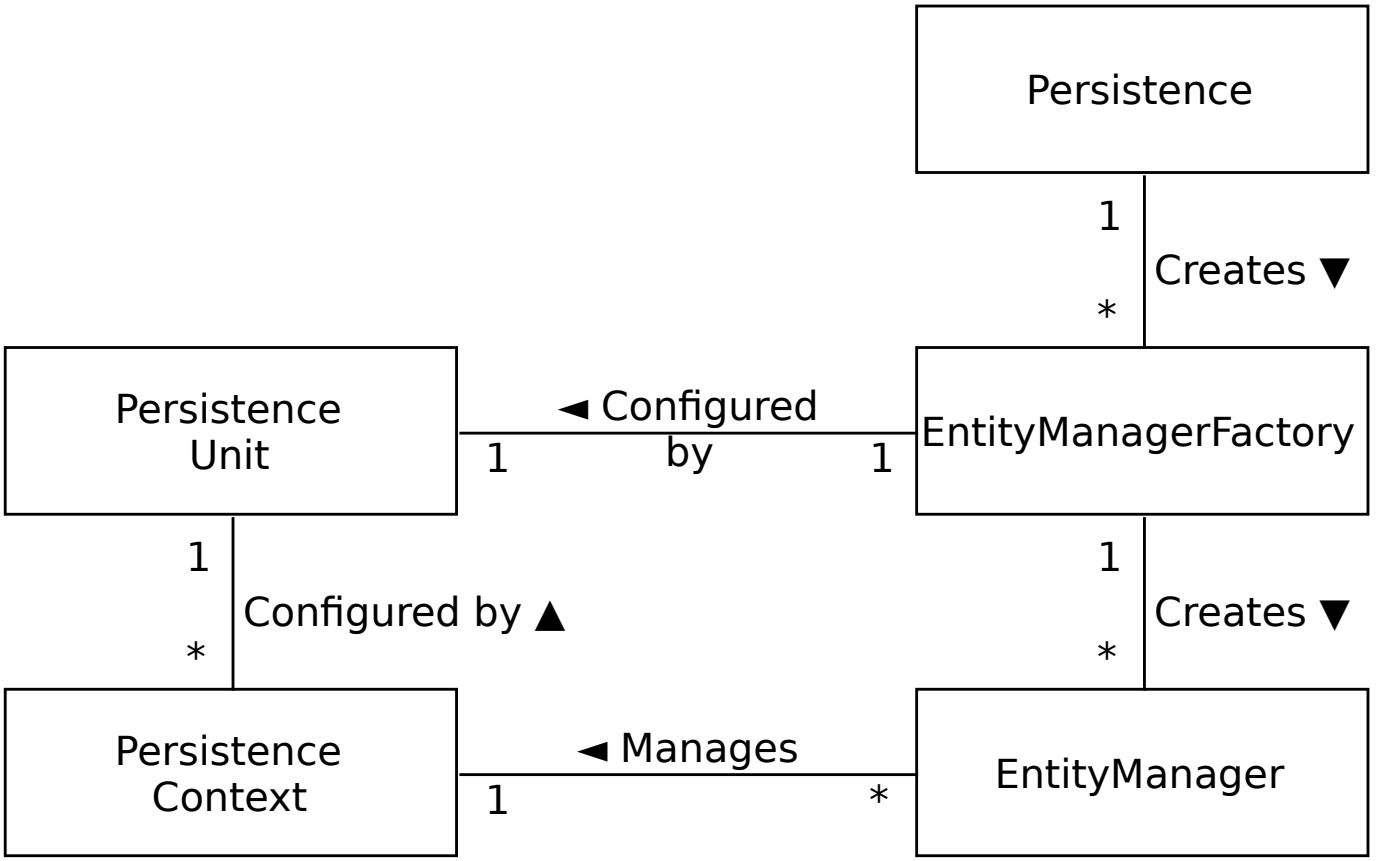
Példányváltozók és tulajdonságok (3)

- A példányváltozók és tulajdonságok típusaként az alábbiak használhatók:
  - Primitív típusok
  - `java.lang.String`
  - További, a `java.io.Serializable` interfészt implementáló típusok (primitív típusok csomagolóosztályai, `java.math.BigInteger`, `java.math.BigDecimal`, `java.util.Date`, `java.util.Calendar`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`, `byte[]`, `Byte[]`, `char[]`, `Character[]`, `java.time.LocalDate`, `java.time.LocalTime`, `java.time.LocalDateTime`, `java.time.OffsetTime`, `java.time.OffsetDateTime`)
  - A `java.io.Serializable` interfészt implementáló felhasználói típusok
  - Enumok
  - Entitás osztályok
  - Entitás osztályok kollekciói
  - Beágyazható osztályok
  - Beágyazható objektumokból vagy elemi típusú elemekből álló kollekciók

Elsődleges kulcsok és entitás identitás

- Minden entitásnak kell, hogy legyen egy elsődleges kulcsa.
- Az elsődleges kulcs az entitás osztály egy vagy több példányváltozójának vagy tulajdonságának felel meg.
  - Elsődleges kulcs lehet egyszerű vagy összetett.
- Az elsődleges kulcsot alkotó példányváltozók vagy tulajdonságok típusaként használható:
  - Tetszőleges primitív típus vagy annak csomagoló típusa, `java.lang.String`, `java.util.Date`, `java.sql.Date`, `java.math.BigDecimal`, `java.math.BigInteger`.

JPA fogalmak és kapcsolatuk



Forrás: Mike Keith, Merrick Schincariol. *Pro JPA 2*. 2nd edition. Apress, 2013.

EntityManager (1)

- A `javax.persistence.EntityManager` interfész biztosít API-t a perzisztencia műveletekhez. https://javaee.github.io/javaee-spec/javadocs/javax/persistence/EntityManager.html
  - Lásd például a `find()`, `persist()` és `remove()` metódusokat.

EntityManager (2)

- Miután az `EntityManager` egy referenciát kap egy entitás objektumra, akkor azt mondjuk, hogy az objektumot az `EntityManager` **kezeli** (***manages***).
  - Referenciát kaphat egy entitás objektumra metódushívás paramétereként vagy adatbázisból való beolvasás révén.
- Forrás: Mike Keith, Merrick Schincariol. *Pro JPA 2*. 2nd edition. Apress, 2013.

Perzisztencia kontextus

- **Perzisztencia kontextus**nak (***persistenc context***) nevezzük egy `EntityManager` által kezelt entitás objektumokat.
  - Egy perzisztencia kontextuson belül minden egyes entitás példánynak egyedi perzisztens identitása kell, hogy legyen.
- Egy `EntityManager` által használt entitás osztályokat egy perzisztencia egység határozza meg.

Perzisztencia egység

- Egy **perzisztencia egység** (***persistence unit***) osztályok egy olyan halmazát határozza meg, melyeket egy alkalmazás együtt kezel, és melyeket ugyanarra az adatbázisra kell leképezni.

EntityManagerFactory

- A `javax.persistence.EntityManagerFactory` interfész szolgál egy adott perzisztencia egységhez `EntityManager` objektumok létrehozására.
  https://javaee.github.io/javaee-spec/javadocs/javax/persistence/EntityManagerFactory.html
  - Java SE alkalmazások a `javax.persistence.Persistence` osztály `createEntityManagerFactory()` metódusát kell, hogy használják egy példány létrehozásához.
    https://javaee.github.io/javaee-spec/javadocs/javax/persistence/Persistence.html

Perzisztencia egység (1)

- Egy perzisztencia egység egy logikai csoport, mely az alábbiakat tartalmazza:
  - Egy `EntityManagerFactory` és `EntityManager`-ei a konfigurációs beállításaikkal együtt.
  - A prezisztencia egységben az `EntityManagerFactory` `EntityManager`-ei által kezelt osztályok.
  - Az osztályok az adatbázisra való leképezését meghatározó metaadatok (annotációk és/vagy XML metaadatok formájában).
- Definiálása egy `persistence.xml` állományban történik.

Perzisztencia egység (2)

- Minden perzisztencia egységnek van egy neve.
  - A perzisztencia egységek neve egyedi kell, hogy legyen a hatáskörükben.

persistence.xml (1)

- Egy vagy több perzisztencia egységet definiáló XML dokumentum.
  - XML séma: http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd
- Az állományt egy JAR állomány vagy könyvtár `META-INF` könyvtárában kell elhelyezni.
  - A tartalmazó JAR állományt vagy könyvtárat a perzisztencia egység gyökerének nevezik.

persistence.xml (2)

- Dokumentum szerkezet:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
    version="2.2">
  <!-- Egy vagy több persistence-unit elem: -->
  <persistence-unit name="név">
    <!-- ... -->
  </persistence-unit>
  <!-- ... -->
</persistence>
```

persistence.xml (3)

- Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.2">
  <persistence-unit name="test" transaction-type="RESOURCE_LOCAL">
    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
      <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:test"/>
      <property name="javax.persistence.schema-generation.database.action"
        value="drop-and-create"/>
      <property name="javax.persistence.schema-generation.scripts.action"
        value="drop-and-create"/>
      <property name="javax.persistence.schema-generation.scripts.create-target"
        value="./create.sql"/>
      <property name="javax.persistence.schema-generation.scripts.drop-target"
        value="./drop.sql"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.use_sql_comments" value="true"/>
      <property name="javax.persistence.sql-load-script-source"
        value="META-INF/load.sql"/>
    </properties>
  </persistence-unit>
</persistence>
```

XML ORM leíró (1)

- Az objektum-relációs leképezéshez metaadatokat tartalmazó XML dokumentum, mely révén kiegészíthetők vagy felülírhatók az annotációkkal adott metaadatok.
  - A `persistence-unit-metadata/xml-mapping-metadata-complete` elem jelenléte esetén az annotációkat teljesen figyelmen kívül kell hagyni.
    - Egyébként a dokumentum felülírja illetve kiegészíti az annotációk révén szolgáltatott metaadatokat.
- XML séma: http://xmlns.jcp.org/xml/ns/persistence/orm_2_2.xsd

XML ORM leíró (2)

- Dokumentum szerkezet:

```
<entity-mappings
    xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
      http://xmlns.jcp.org/xml/ns/persistence/orm_2_2.xsd"
    version="2.2">
  <!-- ... -->
</entity-mappings>
```

JPA „Helló, világ!” program (1)

- `pom.xml`:

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.10.Final</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

JPA „Helló, világ!” program (2)

- `Message.java`:

```
package hello.model;

@javax.persistence.Entity
public class Message {

  @javax.persistence.Id
  @javax.persistence.GeneratedValue
  private Long id;
  private String text;

  public Message() {
  }

  public String getText() {
    return text;
  }

  public void setText(String text) {
    this.text = text;
  }
}
```

JPA „Helló, világ!” program (3)

- `HelloWorld.java`:

```
package hello;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import hello.model.Message;

public class HelloWorld {

  public static void main(String[] args) {
    EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("hello");
    EntityManager em = emf.createEntityManager();
    Message message = new Message();
    message.setText("Hello, World!");
    em.getTransaction().begin();
    em.persist(message);
    em.getTransaction().commit();
    em.close();
    emf.close();
  }
}
```

JPA „Helló, világ!” program (4)

- `META-INF/persistence.xml`:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        version="2.2">
  <persistence-unit name="hello">
    <class>hello.model.Message</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="org.h2.Driver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:h2:mem:test"/>
      <property name="javax.persistence.schema-
generation.database.action" value="drop-and-create"/>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.H2Dialect"/>
    </properties>
  </persistence-unit>
</persistence>
```

JPA „Helló, világ!” program (5)

- Módosítás az adatbázisban:

```
EntityManagerFactory emf =
  Persistence.createEntityManagerFactory("Hello");
EntityManager em = emf.createEntityManager();

em.getTransaction().begin();
Message message = em.createQuery("SELECT m FROM Message m")
  .getSingleResult();
assert message.getText().equals("Hello, World!");
message.setText("Hello, " + System.getProperty("user.name") + "!");
em.getTransaction().commit();

em.close();
emf.close();
```

Az előbbi példa annotációk nélkül (1)

- `Message.java`:

```
package hello.model;

public class Message {

  private Long id;
  private String text;

  public Message() {
  }

  public String getText() {
    return text;
  }

  public void setText(String text) {
    this.text = text;
  }

}
```

Az előbbi példa annotációk nélkül (2)

- `META-INF/persistence.xml`:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        version="2.2">
  <persistence-unit name="hello">
    <mapping-file>META-INF/orm.xml</mapping-file>
    <class>hello.model.Message</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="org.h2.Driver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:h2:mem:test"/>
      <property name="javax.persistence.schema-
generation.database.action" value="drop-and-create"/>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.H2Dialect"/>
    </properties>
  </persistence-unit>
</persistence>
```

Az előbbi példa annotációk nélkül (3)

- `META-INF/orm.xml`:

```
<entity-mappings
    xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
    version="2.2">
  <persistence-unit-metadata>
    <xml-mapping-metadata-complete/>
  </persistence-unit-metadata>
  <entity class="hello.model.Message" access="FIELD">
    <attributes>
      <id name="id">
        <generated-value strategy="AUTO"/>
      </id>
      <basic name="text"/>
    </attributes>
  </entity>
</entity-mappings>
```

Egy összetettebb példa (1)

- `Employee.java`:

```
@javax.persistence.Entity
public class Employee {

  @javax.persistence.Id
  private int id;
  private String name;
  private long salary;

  public Employee() {}

  public int getId() { return id; }
  public void setId(int id) { this.id = id; }

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }

  public long getSalary() { return salary; }
  public void setSalary(long salary) { this.salary = salary; }

}
```

Egy összetettebb példa (2)

- `EmployeeService.java`:

```
import java.util.List;
import javax.persistence.*;

public class EmployeeService {

  protected EntityManager em;

  public EmployeeService(EntityManager em) {
    this.em = em;
  }

  // alkalmazott létrehozása és mentése az adatbázisba
  public Employee createEmployee(int id, String name, long salary) {
    Employee emp = new Employee();
    emp.setId(id);
    emp.setName(name);
    emp.setSalary(salary);
    em.persist(emp);
    return emp;
  }
```

Egy összetettebb példa (3)

- `EmployeeService.java` (folytatás):

```
  // adott azonosítójú alkalmazott törlése az adatbázisból
  public void removeEmployee(int id) {
    Employee emp = findEmployee(id);
    if (emp != null) {
      em.remove(emp);
    }
  }

  // adott azonosítójú alkalmazott fizetésének növelése
  public Employee raiseEmployeeSalary(int id, long raise) {
    Employee emp = em.find(Employee.class, id);
    if (emp != null) {
      emp.setSalary(emp.getSalary() + raise);
    }
    return emp;
  }
```

Egy összetettebb példa (4)

- `EmployeeService.java` (folytatás):

```
  // adott azonosítójú alkalmazott betöltése az adatbázisból
  public Employee findEmployee(int id) {
    return em.find(Employee.class, id);
  }

  // az összes alkalmazott betöltése az adatbázisból
  public List<Employee> findAllEmployees() {
    TypedQuery<Employee> query = em.createQuery(
      "SELECT e FROM Employee e", Employee.class);
    return query.getResultList();
  }

}
```

Egy összetettebb példa (5)

- `EmployeeTest.java`:

```
import javax.persistence.*;
import java.util.List;

public class EmployeeTest {

  public static void main(String[] args) {
    EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("EmployeeService");
    EntityManager em = emf.createEntityManager();
    EmployeeService service = new EmployeeService(em);

    // alkalmazott létrehozása és mentése
    em.getTransaction().begin();
    Employee emp = service.createEmployee(158, "John Doe", 45000);
    em.getTransaction().commit();
    System.out.println("Persisted " + emp);
```

Egy összetettebb példa (6)

- `EmployeeTest.java` (folytatás):

```
    // adott alkalmazott betöltése
    emp = service.findEmployee(158);
    System.out.println("Found " + emp);

    // az összes alkalmazott betöltése
    List<Employee> emps = service.findAllEmployees();
    for (Employee e : emps) {
      System.out.println("Found employee: " + e);
    }
```

Egy összetettebb példa (7)

- `EmployeeTest.java` (folytatás):

```
    // adott alkalmazott módosítása
    em.getTransaction().begin();
    emp = service.raiseEmployeeSalary(158, 1000);
    em.getTransaction().commit();
    System.out.println("Updated " + emp);

    // adott alkalmazott törlése
    em.getTransaction().begin();
    service.removeEmployee(158);
    em.getTransaction().commit();
    System.out.println("Removed Employee 158");

    em.close();
    emf.close();
  }

}
```

Java 8 dátum és idő típusok használata (1)

- A Hibernate ORM 5.2.0 számú kiadásában jelent meg a Java 8 dátum és idő típusainak alapértelmezett támogatása, korábban ehhez egy `hibernate-java8` nevű programkönyvtár volt szükséges futásidőben.
  - Lásd: *Hibernate ORM 5.2 release*. Jun 1, 2016. http://in.relation.to/2016/06/01/hibernate-orm-520-final-release/
  - A JPA 2.2 számú verziója teszi lehetővé a `java.time` csomag dátum és idő típusainak használatát.
    - A Hibernate támogatja a `java.time.Duration`, `java.time.Instant`, `java.time.ZonedDateTime` típusokat is, a JPA 2.2 azonban nem!

Java 8 dátum és idő típusok használata (2)

- `Employee.java`:

```
@javax.persistence.Entity
public class Employee {

  @javax.persistence.Id
  private int id;
  private String name;
  private long salary;
  private java.time.LocalDate dob; // nincs további teendő

  public Employee() {}

  public int getId() { return id; }
  public void setId(int id) { this.id = id; }

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }

  public long getSalary() { return salary; }
  public void setSalary(long salary) { this.salary = salary; }

  public java.time.LocalDate getDob() { return dob; }
  public void setDob(java.time.LocalDate dob) { this.dob = dob; }
}
```

Alapértelmezett táblanév felülírása

- `Employee.java`:

```
@javax.persistence.Entity
@javax.persistence.Table(name="EMP", schema="HR")
public class Employee {

  // ...

}
```

Alapértelmezett oszlopnév felülírása

- `Employee.java`:

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  @Column(name="EMP_ID")
  private int id;

  private String name;

  @Column(name="SAL")
  private long salary;

  @Column(name="COMM")
  private String comments;
  // ...
}
```

Lusta betöltés

- `Employee.java`:

```
import javax.persistence.*;

@Entity
public class Employee {

   // ...
   @Basic(fetch=FetchType.LAZY)
   @Column(name="COMM")
   private String comments;
   // ...
}
```

Enum használata (1)

- Enum konstans sorszámának tárolása (FULL_TIME_EMPLOYEE: 1, PART_TIME_EMPLOYEE: 2, CONTRACT_EMPLOYEE: 3):
  - Új konstans hozzáadásával elcsúszhat a számozás!

```
public enum EmployeeType {
  FULL_TIME_EMPLOYEE,
  PART_TIME_EMPLOYEE,
  CONTRACT_EMPLOYEE
}

@javax.persistence.Entity
public class Employee {

  @javax.persistence.Id
  private int id;

  private EmployeeType type; // there is nothing to be done
}
```

Enum használata (2)

- Enum konstans nevének tárolása:

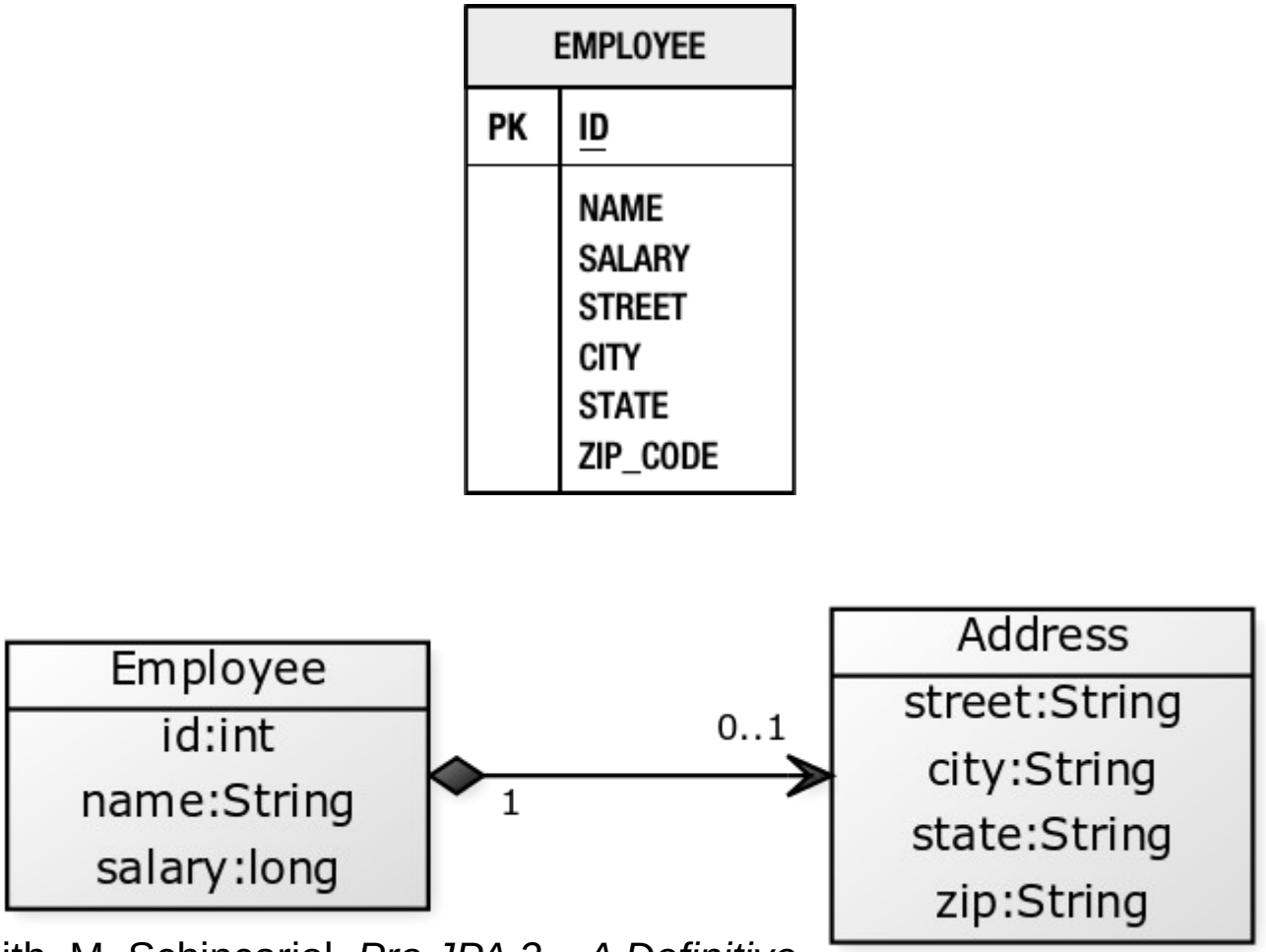
```
import javax.persistence.*;

public enum EmployeeType {
  FULL_TIME_EMPLOYEE,
  PART_TIME_EMPLOYEE,
  CONTRACT_EMPLOYEE
}

@Entity
public class Employee {

  @Id
  private int id;

  @Enumerated(EnumType.STRING)
  private EmployeeType type;
  // ...
}
```

Beágyazott objektumok (1)

- Olyan objektumok, melyek nem rendelkeznek saját identitással, hanem egy másik entitás állapotának részeként léteznek csak.

Beágyazott objektumok (2)

- Példa:



Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

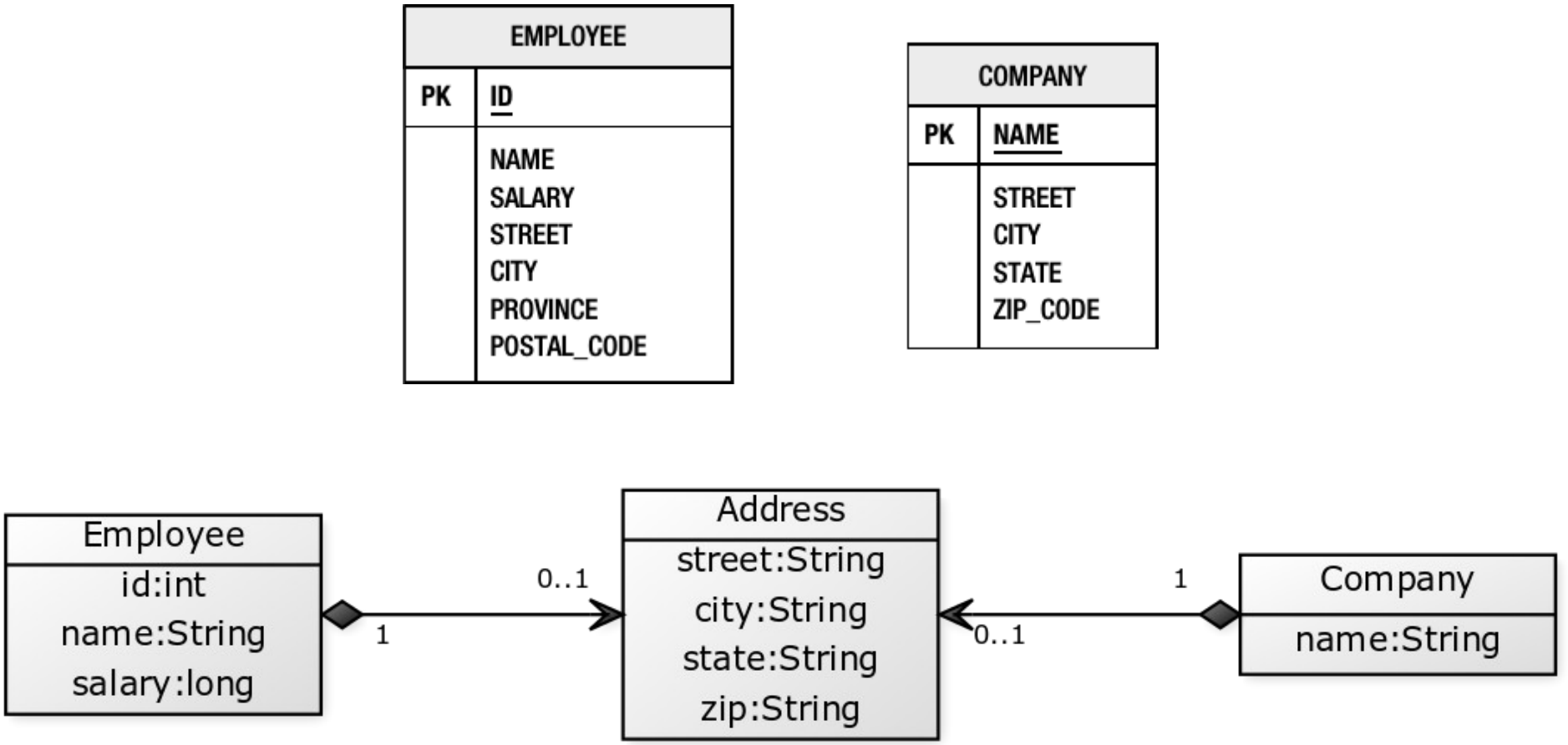
Beágyazott objektumok (3)

- Példa (folytatás):

```
import javax.persistence.*;

@Embeddable
@Access(AccessType.FIELD)
public class Address {

  private String street;
  private String city;
  private String state;
  @Column(name="ZIP_CODE") private String zip;
  // ...
}

@Entity
public class Employee {

  @Id private int id;
  private String name;
  private long salary;
  @Embedded private Address address;
  // ...
}
```

Beágyazott objektumok (4)

- Példa:



Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

Beágyazott objektumok (5)

- Példa (folytatás):

```
import javax.persistence.*;

@Entity public class Employee {

  @Id private int id;
  private String name;
  private long salary;
  @Embedded
  @AttributeOverrides({
    @AttributeOverride(name="state", column=@Column(name="PROVINCE")),
    @AttributeOverride(name="zip", column=@Column(name="POSTAL_CODE"))
  }) private Address address;
  // ...
}

@Entity public class Company {

  @Id private String name;
  @Embedded private Address address;
  // ...
}
```

Beágyazott objektumok (6)

- Példa (folytatás): A JPA 2.2 számú verziójától az `@AttributeOverride` annotáció ismételhető (Hibernate 5.3.0).

```
import javax.persistence.*;
@Entity public class Employee {

  @Id private int id;
  private String name;
  private long salary;
  @Embedded
  @AttributeOverride(name="state", column=@Column(name="PROVINCE"))
  @AttributeOverride(name="zip", column=@Column(name="POSTAL_CODE"))
  private Address address;
  // ...
}

@Entity public class Company {

  @Id private Str// ...ing name;
  @Embedded private Address address;
  // ...
}
```

Megszorítások (1)

- Használhatók a `javax.validation.constraints` csomag megszorításai.
  - *Bean Validation* https://beanvalidation.org/
    - A Java EE része (lásd a `javax.validation` csomagot és alcsomagjait).
      - *JSR 380: Bean Validation 2.0 (Final Release)* (August 3, 2017) https://jcp.org/en/jsr/detail?id=380
      - *Jakarta Bean Validation 2.0* https://jakarta.ee/specifications/bean-validation/2.0/
    - Referencia implementáció: *Hibernate Validator* http://hibernate.org/validator/

Megszorítások (2)

- Szükséges hozzá az *Unified Expression Language* (EL) implementációja, mely lehetővé teszi dinamikus kifejezések használatát a megszorítások megsértéséhez tartozó üzenetekben.
  - *JSR 341: Expression Language 3.0 (Final Release)* (April 29, 2013) https://jcp.org/en/jsr/detail?id=341
  - *Jakarta Expression Language 3.0* https://jakarta.ee/specifications/expression-language/3.0/
- Java EE konténerekben alapértelmezésben rendelkezésre áll az EL.

Megszorítások (3)

- `pom.xml`:

```
<dependencies>
  ...
  <dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.1.Final</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>jakarta.el</artifactId>
    <version>3.0.3</version>
    <scope>runtime</scope>
  </dependency>
  ...
</dependencies>
```

Megszorítások (4)

- `Employee.java`:

```
import javax.validation.constraints.*;

@javax.persistence.Entity
public class Employee {

  @javax.persistence.Id
  private int id;

  @NotNull(message="Employee name must be specified")
  @Size(min=1, max=100)
  private String name;

  @NotNull
  @Size(max=40)
  @Email
  private String email;
  // ...
}
```

Megszorítások (5)

- `Employee.java`: üzenet paraméterek és üzenet kifejezések használata

```
import javax.validation.constraints.*;

@javax.persistence.Entity
public class Employee {

  @javax.persistence.Id
  private int id;

  @NotNull(message="Employee name must be specified")
  @Size(min=1, max=100, message="The name must be between {min} and
    {max} characters long")
  private String name;

  @NotNull
  @Size(max=40)
  @Email(message="The email address '${validatedValue}' is invalid")
  private String email;
  // ...
}
```

Megszorítások (6)

- Érvényesítés történhet:
  - Automatikusan adatbázis műveletek végrehajtása előtt.
    - Alapértelmezésben `INSERT` és `UPDATE` műveletek végrehajtása előtt történik automatikus érvényesítés.
  - Egy `javax.validation.Validator` objektum `validate()` metódusának meghívásával.
    - Paraméterként kell átadni az objektumot, melyen érvényesítést kell végezni.

Megszorítások (7)

- Automatikus érvényesítés engedélyezése a `persistence.xml` állományban:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        version="2.2">
  <persistence-unit name="...">
    <validation-mode>AUTO|CALLBACK|NONE</validation-mode>
    <!-- ... -->
  </persistence-unit>
</persistence>
```

Megszorítások (8)

- A `validation-mode` elem lehetséges értékei:
  - **`AUTO`** (alapértelmezés): Ha rendelkezésre áll *Bean Validation* szolgáltató a környezetben, akkor a perzisztencia szolgáltató automatikus érvényesítést kell, hogy végezzen az entitásokon. Ha nem áll rendelkezésre *Bean Validation* szolgáltató a környezetben, akkor nem történik automatikus érvényesítés.
  - **`CALLBACK`**: A perzisztencia szolgáltató automatikus érvényesítést kell, hogy végezzen. Hiba, ha nem áll rendelkezésre *Bean Validation* szolgáltató a környezetben.
  - **`NONE`**: automatikus érvényesítés kikapcsolása.

Megszorítások (9)

- Érvényesítés esetén a megszorítások megsértése `javax.validation.ConstraintViolationException` kivételt eredményez.
- A Hibernate a megszorításokat felhasználja adatbázis séma létrehozásához.

EntityManager műveletek (1)

- `void persist(Object entity)`:
  - A paraméterként adott objektum mentése az adatbázisba.
    - Ha az objektumot az `EntityManager` már kezeli, akkor nem történik adatbázis művelet.
    - Az adatbázisban már létező objektum `javax.persistence.EntityExistsException` kivételt eredményez.

EntityManager műveletek (2)

- `<T> T find(Class<T> entityClass, Object primaryKey)`:
  - Az adott osztályú és elsődleges kulcsú entitás keresése.
    - A metódus a megtalált entitás példányt adja vissza, vagy `null`-t, ha nem létezik az entitás.

EntityManager műveletek (3)

- `void remove(Object entity)`:
  - Entitás példány törlése az adatbázisból.
    - Csak az `EntityManager` által kezelt objektum törölhető, egyébként a metódus `IllegalArgumentException` kivételt dob.

EntityManager műveletek (4)

- `void detach(Object entity)`:
  - Az adott entitás törlése a perzisztencia kontextusból.
    - Az objektum a továbbiakban **leválasztott**.
    - Az entitáson történt olyan módosítások, melyek nem lettek az adatbázisba kiírva, nem kerülnek szinkronizálásra az adatbázissal.

EntityManager műveletek (5)

- `<T> T merge(T entity)`:
  - Az adott (leválasztott) entitás állapotának belefésülése az aktuális perzisztencia kontextusba, a leválasztás ellentéte.
    - A paraméterként adott entitás továbbra is leválasztott marad.
    - A visszatérési érték egy kezelt példány.
      - Ez egy új kezelt példány, vagy pedig egy, a perzisztencia kontextusban már létező kezelt példány.

EntityManager műveletek (6)

- `void clear()`:
  - A perzisztencia kontextus törlése, mely az `EntityManager` által kezelt valamennyi objektumot leválasztottá teszi.
  - Az entitásokon történt, de az adatbázisba nem kiírt módosítások nem kerülnek mentésre.
- `void flush()`:
  - A perzisztencia kontextus szinkronizálása az adatbázissal.

Perzisztencia műveletek kaszkádolása (1)

- Alapértelmezésben valamennyi perzisztencia művelet csak a paraméterként adott entitásra vonatkozik.
  - A művelet nem lesz végrehajtva az adott entitással kapcsolatban lévő más entitásokkal.
  - A logikai kapcsolatokat definiáló annotációkhoz (`@ManyToMany`, `@ManyToOne`, `@OneToMany`, `@OneToOne`) megadható `cascade` elemmel bírálható felül az alapértelmezett viselkedés.

Perzisztencia műveletek kaszkádolása (2)

- A `javax.persistence.CascadeType` enum példányai ábrázolják, hogy mely műveleteknél történik kaszkádolás:
  - `ALL`
  - `DETACH`
  - `MERGE`
  - `PERSIST`
  - `REFRESH`
  - `REMOVE`

Perzisztencia műveletek kaszkádolása (3)

- Példa:

```
import javax.persistence.*;

@Entity
public class Employee {

   // ...
   @ManyToOne(cascade=CascadeType.PERSIST)
   Address address;
   // ...
}
```

Perzisztencia műveletek kaszkádolása (4)

- Kizárólag a `persist()` műveletre globálisan beállítható a kaszkádolás az XML leíróban:

```
<entity-mappings
    xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
    version="2.2">
  <persistence-unit-metadata>
    <persistence-unit-defaults>
      <cascade-persist/>
    </persistence-unit-defaults>
  </persistence-unit-metadata>
</entity-mappings>
```

Leválasztott entitások (1)

- Egy leválasztott entitás egy perzisztencia kontextushoz nem tartozó entitás.
  - A leválasztott objektumon végzett módosítások nem kerülnek mentésre az adatbázisban.
- Egy entitás többféle módon válhat leválasztottá, például:
  - Egy perzisztencia kontextus lezárásakor valamennyi kezelt entitása leválasztott lesz.
  - Az `EntityManager` `clear()` metódusa leválasztja az általa kezelt valamennyi entitást.
  - Az `EntityManager` `detach()` metódusa leválasztja a paraméterként adott entitást.
  - …

Leválasztott entitások (2)

- Leválasztott entitás kezelése (nem a várt eredményt adja):
  - Tranzakció véglegesítéskor nem módosul az adatbázisban az entitás.

```
EntityManager em;

Employee emp; // referencia egy leválasztott entitásra

em.getTransaction().begin();
em.merge(emp);
emp.setLastAccessTime(java.time.Instant.now());
em.getTransaction().commit();
```

Leválasztott entitások (3)

- Leválasztott entitás kezelése:
  - Tranzakció véglegesítéskor módosul az adatbázisban az entitás.

```
EntityManager em;

Employee emp; // referencia egy leválasztott entitásra

em.getTransaction().begin();
Employee managedEmp = em.merge(emp);
managedEmp.setLastAccessTime(java.time.Instant.now());
em.getTransaction().commit();
```

Szinkronizálás az adatbázissal

- Amikor a perzisztencia szolgáltató SQL utasításokat hajt végre az adatbázisban egy JDBC kapcsolaton keresztül, azt mondjuk, hogy a perzisztencia kontextus **kiírásra kerül** (***flushed***).
  - Ez bármikor megtörténhet, amikor a perzisztencia szolgáltató szükségesnek ítéli.
  - Két esetben garantált a perzisztencia kontextus kiírása:
    - Tranzakció véglegesítéskor.
    - Az `EntityManager` `flush()` metódusának meghívásakor.

Elsődleges kulcs generálása (1)

- Az elsődleges kulcs automatikus generálására szolgál a `javax.persistence.GeneratedValue` annotáció típus.
  - Egy perzisztencia szolgáltató csak egyszerű elsődleges kulcsokhoz kell, hogy támogassa az annotáció használatát.
  - A generált kulcs csak az entitás az adatbázisba történő beszúrása után lesz garantáltan elérhető.

Elsődleges kulcs generálása (2)

- Négyféle generálási stratégia választható, melyeket a `javax.persistence.GenerationType` enum példányai ábrázolnak:
  - **`AUTO`**: egy alkalmas stratégia automatikus választása az adott adatbázishoz.
  - **`IDENTITY`**: a perzisztencia szolgáltató egy *identity* oszlopot használ az elsődleges kulcs generálásához.
  - **`SEQUENCE`**: a perzisztencia szolgáltató egy szekvenciát használ az elsődleges kulcs generálásához.
  - **`TABLE`**: a perzisztencia szolgáltató egy adatbázis táblát használ az elsődleges kulcs generálásához.
    - Ez a legrugalmasabb és leghordozhatóbb stratégia.

Elsődleges kulcs generálása (3)

- Példa:

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;
  // ...
}
```

Elsődleges kulcs generálása (4)

- Példa (folytatás):
  - Az `Employee` adatbázis táblát létrehozó DDL utasítás (H2):

```
create table Employee (
  id bigint generated by default as identity,
  // ...
  primary key (id)
)
```

Lekérdezések (1)

- Java Persistence Query Language (JPQL):
  - A JPA részeként specifikált platformfüggetlen objektumorientált lekérdezőnyelv, mely révén adatbázisban tárolt perzisztens entitásokra vonatkozó lekérdezések fogalmazhatók meg.
  - Lefordítható egy célnyelvre, mint például az SQL.
  - 3 fajta utasítás: SELECT, UPDATE, DELETE.

Lekérdezések (2)

- Statikus és dinamikus lekérdezések:
  - A statikus lekérdezések definiálása annotációkban vagy XML leírókban.
    - Nevük van, nevesített lekérdezéseknek is nevezik őket, a végrehajtásnál a nevükkel kell hivatkozni rájuk.
  - A dinamikus lekérdezések futásidőben kerülnek megadásra.

Lekérdezések (3)

- A `javax.persistence.Query` és `javax.persistence.TypedQuery<`*X*`>` interfészek ábrázolják a lekérdezéseket.
  - Az `EntityManager` `createNamedQuery()`, `createNativeQuery()` és `createQuery()` metódusaival hozhatók létre.

Lekérdezések (4)

- Példák SELECT lekérdezésekre:

```
SELECT e FROM Employee e

SELECT e.name FROM Employee e

SELECT e.name, e.salary FROM Employee e
ORDER BY e.name

SELECT e FROM Employee e
WHERE e.address.state IN ('NY','CA')
```

Lekérdezések (5)

- Példák SELECT lekérdezésekre (folytatás):

```
SELECT d FROM Department d
WHERE SIZE(d.employees) = 2

SELECT d, COUNT(e), MAX(e.salary), AVG(e.salary)
FROM Department d JOIN d.employees e
GROUP BY d
HAVING COUNT(e) >= 5
```

Lekérdezések (6)

- Példa dinamikus lekérdezés végrehajtására:

```
TypedQuery<Employee> query = em.createQuery("SELECT e FROM
  Employee e", Employee.class);
List<Employee> employees = query.getResultList();
```

Lekérdezések (7)

- Példa dinamikus lekérdezés használatára:

```
public class QueryService {

  protected EntityManager em;

  public QueryService(EntityManager em) {
    this.em = em;
  }

  public long queryEmpSalary(String deptName,
      String empName) {
    return em.createQuery("SELECT e.salary "
        + "FROM Employee e "
        + "WHERE e.department.name = :deptName"
        + "    AND e.name = :empName", Long.class)
      .setParameter("deptName", deptName)
      .setParameter("empName", empName)
      .getSingleResult();
  }

  // ...
}
```

Lekérdezések (8)

- Példa statikus lekérdezés használatára:

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;

@Entity
@NamedQuery(name="Employee.findByName",
  query="SELECT e FROM Employee e WHERE e.name = :name")
public class Employee {

  // ...
}
```

Lekérdezések (9)

- Példa statikus lekérdezés használatára (folytatás):

```
public class QueryService {

  protected EntityManager em;

  public QueryService(EntityManager em) {
    this.em = em;
  }

  // ...

  public Employee findEmployeeByName(String name) {
    return em.createNamedQuery("Employee.findByName", Employee.class)
      .setParameter("name", name)
      .getSingleResult();
  }

}
```

Lekérdezések (10)

- Criteria API:
  - Lekérdezések programkóddal történő létrehozására szolgáló API.
  - A `javax.persistence.criteria` csomag tartalmazza.

Lekérdezések (11)

- Példa a Criteria API használatára:
  - A `SELECT e FROM Employee e WHERE e.name = 'John Smith'` lekérdezés végrehajtása:

```
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

// ...

CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> cq = cb.createQuery(Employee.class);
Root<Employee> root = cq.from(Employee.class);
cq.select(root)
  .where(cb.equal(root.get("name"), "John Doe"));

TypedQuery<Employee> query = em.createQuery(cq);
Employee emp = query.getSingleResult();
```

Kapcsolatok (1)

- Két entitás közötti viszonyt jelent a kapcsolat.
  - Egy entitás számos különböző típusú kapcsolatban vehet részt.
- Kapcsolat jellemzői:
  - Irány: egyirányú, kétirányú
    - Minden kétirányú kapcsolatot két egyirányú kapcsolatként tekintünk.
  - Számosság: `0..1`, `1`, `*`

Kapcsolatok (2)

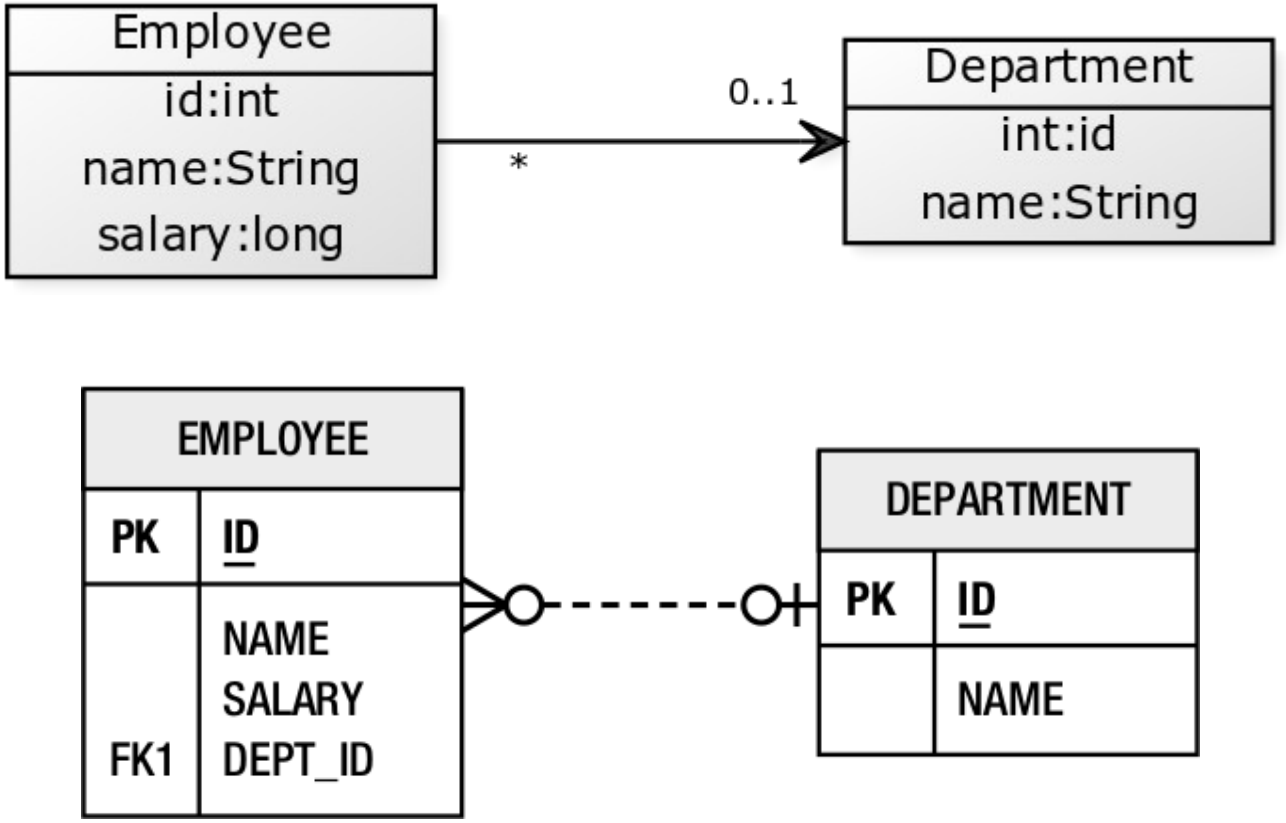
- Az alábbi fajta kapcsolatok használhatóak:
  - Több-egy (*many-to-one*)
  - Egy-egy (*one-to-one*)
  - Egy-több (*one-to-many*)
  - Több-több (*many-to-many*)

Kapcsolatok (3)

- Egyetlen értékű asszociáció (*single-valued association*):
  - Több-egy
  - Egy-egy
    - Speciális esetként tárgyaljuk a kétirányú változatot.
- Kollekció értékű asszociáció (*collection-valued association*):
  - Egy-több
    - Speciális esetként tárgyaljuk az egyirányú változatot.
  - Több-több

Több-egy kapcsolat (1)

- Példa:



Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

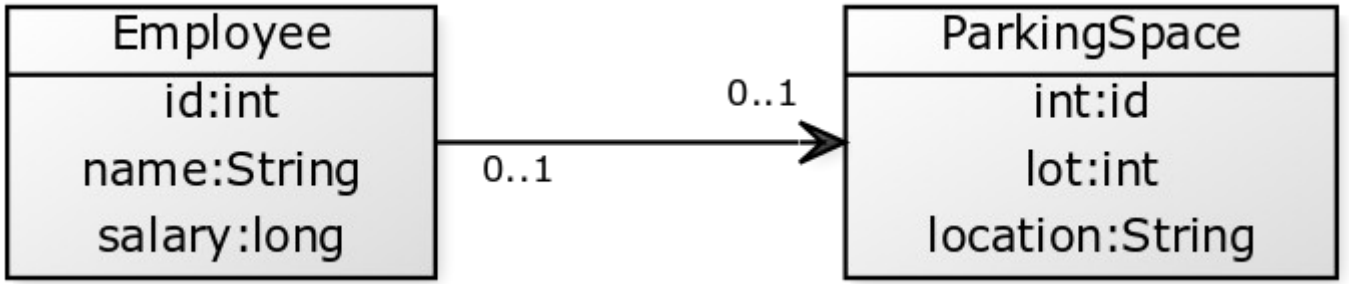
Több-egy kapcsolat (2)

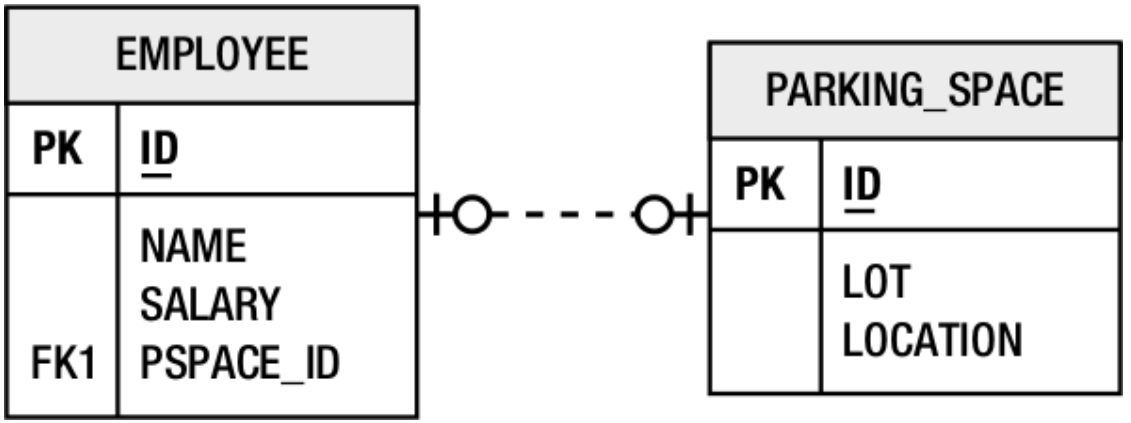
- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;

  @ManyToOne
  @JoinColumn(name="DEPT_ID")
  private Department department;

  // ...
}
```

Egy-egy kapcsolat (1)

- Példa:





Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

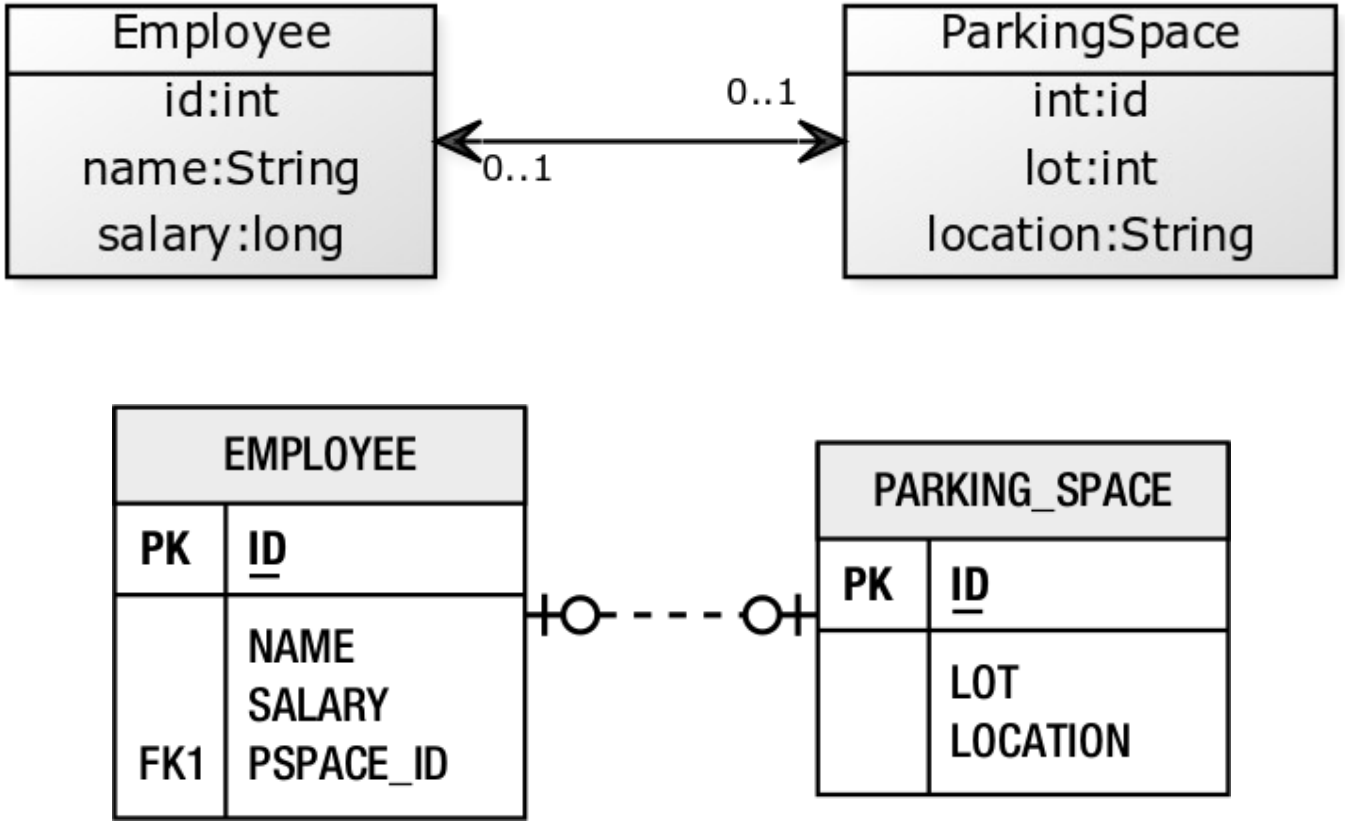
Egy-egy kapcsolat (2)

- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;

  @OneToOne
  @JoinColumn(name="PSPACE_ID")
  private ParkingSpace parkingSpace;

  // ...
}
```

Kétirányú egy-egy kapcsolat (1)

- Példa:



Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

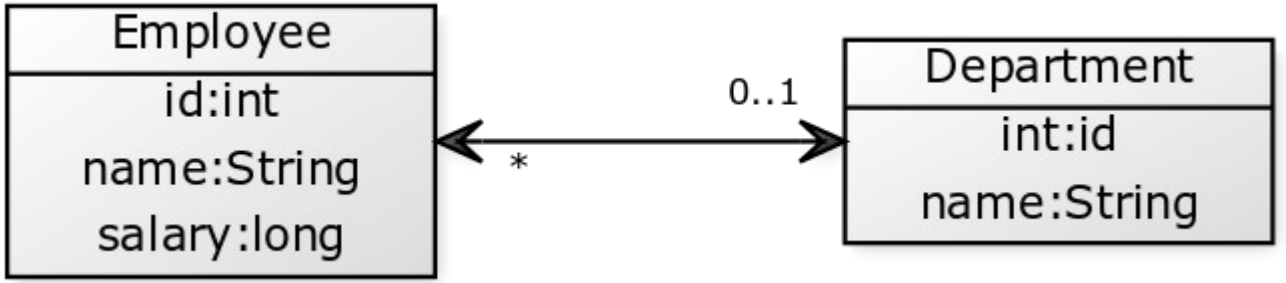
Kétirányú egy-egy kapcsolat (2)

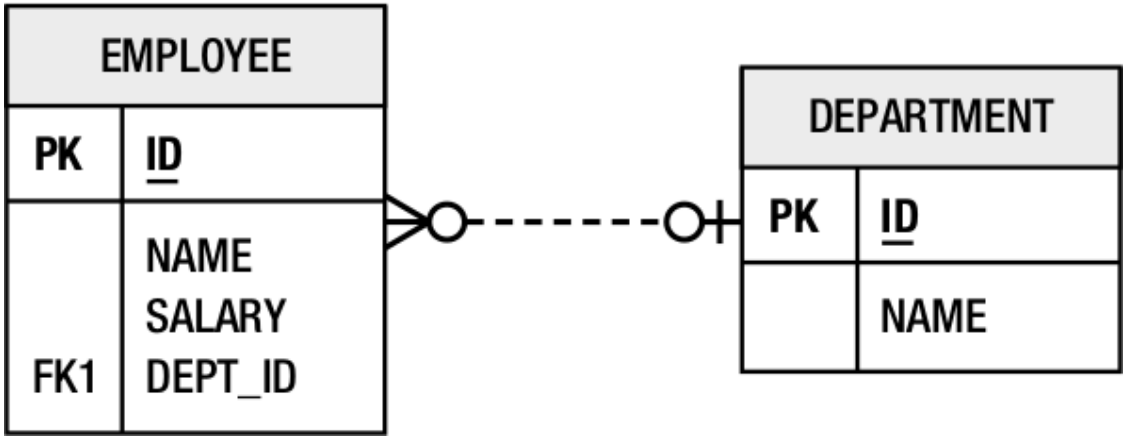
- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class ParkingSpace {

  @Id
  private int id;

  private int lot;
  private String location;

  @OneToOne(mappedBy="parkingSpace")
  private Employee employee;

  // ...
}
```

Egy-több kapcsolat (1)

- Példa:





Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

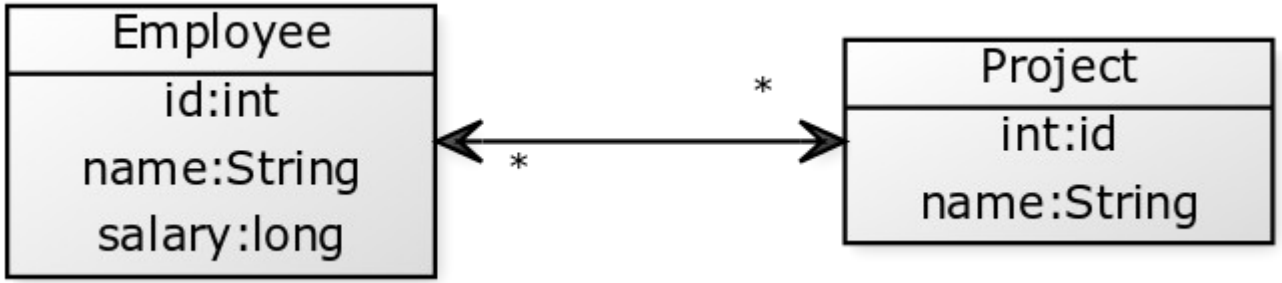
Egy-több kapcsolat (2)

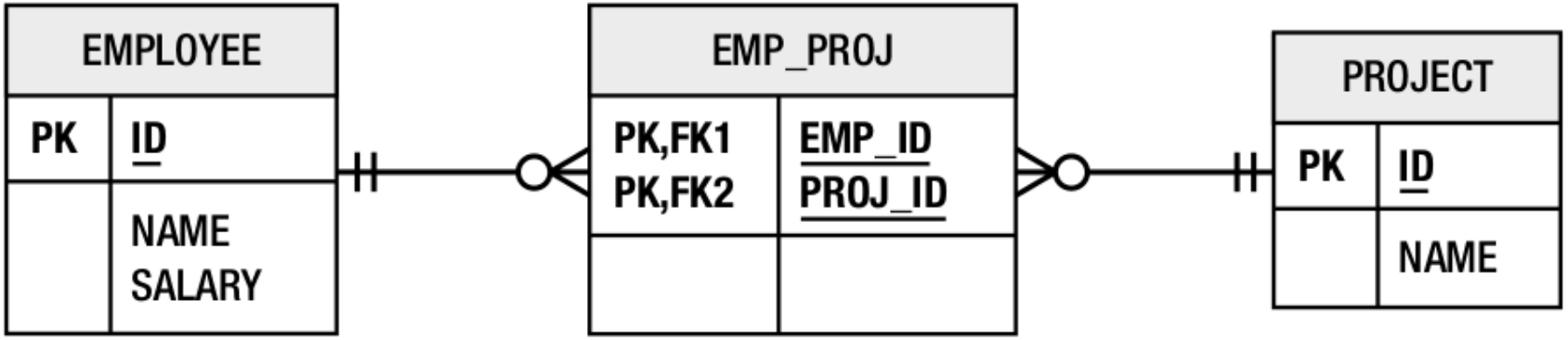
- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class Department {

  @Id
  private int id;

  private String name;

  @OneToMany(mappedBy="department")
  private Collection<Employee> employees;

  // ...
}
```

Több-több kapcsolat (1)

- Példa:





Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

Több-több kapcsolat (2)

- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;

  private String name;

  @ManyToMany
  @JoinTable(name="EMP_PROJ",
    joinColumns=@JoinColumn(name="EMP_ID"),
    inverseJoinColumns=@JoinColumn(name="PROJ_ID"))
  private Collection<Project> projects;

  // ...
}
```

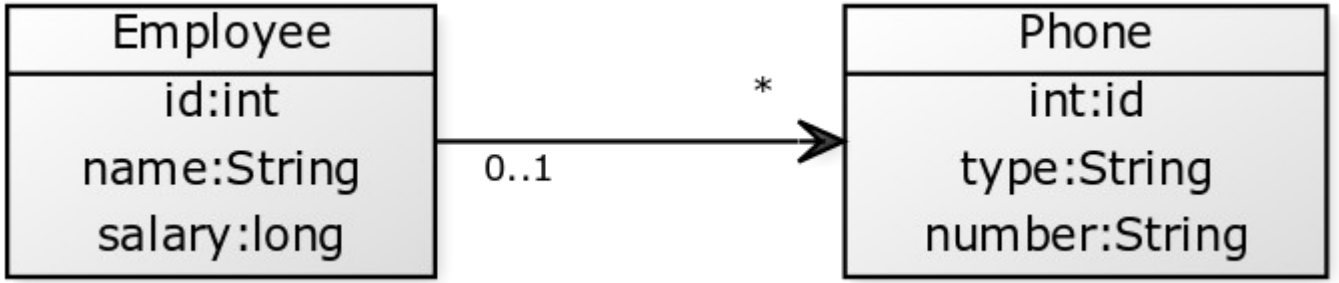
Több-több kapcsolat (3)

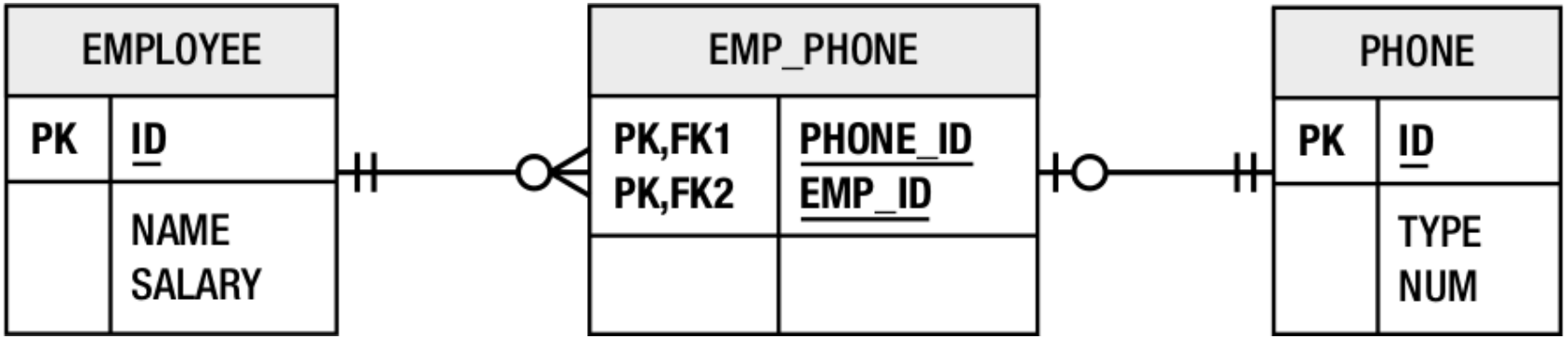
- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class Project {

  @Id
  private int id;

  private String name;

  @ManyToMany(mappedBy="projects")
  private Collection<Employee> employees;

  // ...
}
```

Egyirányú egy-több kapcsolat (1)

- Példa:





Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

Egyirányú egy-több kapcsolat (2)

- Példa:

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private long id;

  private String name;

  @OneToMany
  @JoinTable(name="EMP_PHONE",
    joinColumns=@JoinColumn(name="EMP_ID"),
    inverseJoinColumns=@JoinColumn(name="PHONE_ID"))
  private Collection<Phone> phones;

  // ...
}
```

Elemkollekciók (1)

- Az elemkollekciók beágyazható objektumokból vagy elemi típusú elemekből álló kollekciók.
  - Elemi típusok: például `java.lang.String`, `java.lang.Integer`, `java.math.BigDecimal`, …
    - Lásd: *Basic Types* http://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#basic
- Az elemkollekciók nem kapcsolatok.
  - A kapcsolatok független entitások közötti viszonyok, ellenben az elemkollekciók olyan objektumokat tartalmaznak, melyek a hivatkozó entitástól függenek és kizárólag az őket tartalmazó entitáson keresztül érhetők el.

Elemkollekciók (2)

- Példa:

```
import javax.persistence.*;

@Embeddable
public class VacationEntry {

  private LocalDate startDate;

  @Column(name="DAYS")
  private int daysTaken;

  // ...
}
```

Elemkollekciók (3)

- Példa (folytatás):

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;

  private String name;
  private long salary;

  @ElementCollection(targetClass=VacationEntry.class)
  private Collection vacationBookings;

  @ElementCollection
  private Set<String> nickNames;

  // ...
}
```

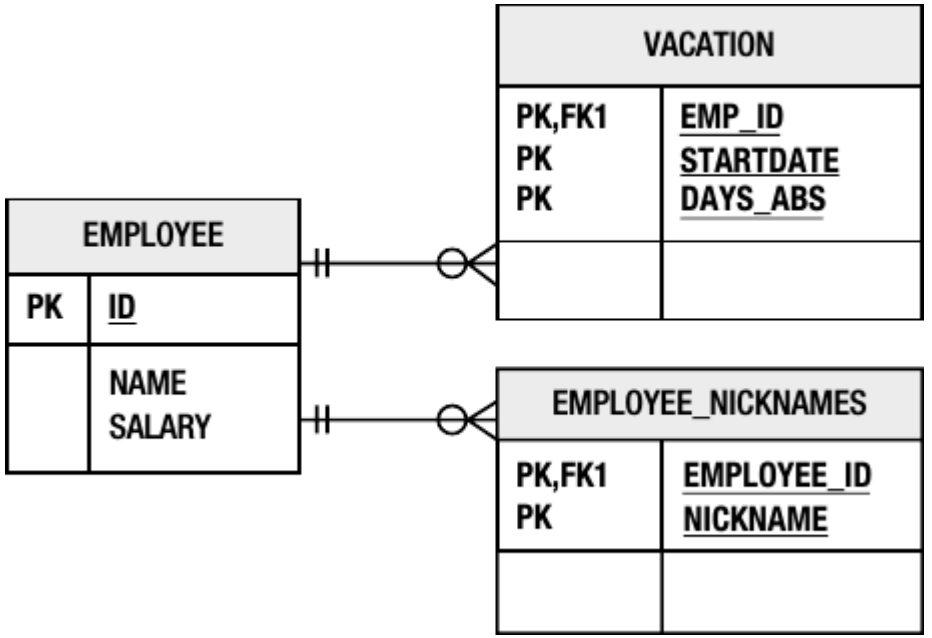
Elemkollekciók (4)

- Példa (folytatás):

```
@Entity
public class Employee {

  @Id
  private int id;
  private String name;
  private long salary;

  @ElementCollection(targetClass=VacationEntry.class)
  @CollectionTable(name="VACATION",
    joinColumns=@JoinColumn(name="EMP_ID"))
  @AttributeOverride(name="daysTaken",
    column=@Column(name="DAYS_ABS"))
  private Collection vacationBookings;

  @ElementCollection
  @Column(name="NICKNAME")
  private Set<String> nickNames;

  // ...
}
```

Elemkollekciók (5)

- Példa (folytatás):



Forrás: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

Elemkollekciók (6)

- Példa lista használatára:

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;
  private String name;
  private long salary;

  @ElementCollection
  @Column(name="NICKNAME")
  @OrderBy
  private List<String> nickNames;

  // ...
}
```

Elemkollekciók (7)

- Példa lista használatára:

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;
  private String name;
  private long salary;

  @ElementCollection
  @Column(name="NICKNAME")
  @OrderColumn(name="NICKNAME_INDEX")
  private List<String> nickNames;

  // ...
}
```

Naplózás (1)

- A Hibernate a **JBoss Logging** keretrendszert használja a naplózáshoz. https://github.com/jboss-logging/jboss-logging
  - Bedugható alá több naplózó keretrendszer, például a `java.util.logging` csomag vagy az SLF4J.
- További információk:
  - Thorben Janssen. *Hibernate Logging Guide – Use the right config for development and production*. December, 2015. https://thoughts-on-java.org/hibernate-logging-guide/
  - *Hibernate ORM User Guide – Statement logging and statistics* https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#configurations-logging
  - *Logging Guide* https://docs.jboss.org/hibernate/orm/5.4/topical/html_single/logging/Logging.html

Naplózás (2)

- A naplóüzenetek kategóriákba történő csoportosítása, ahol minden egyes kategóriához beállítható naplózási szint.
  - A kategóriák gyakorlatilag egy azonos nevű `Logger` objektumnak felelnek meg.
- Érdeklődésre számot tartó naplózási kategóriák:

Kategória	Leírás
`org.hibernate`	Az összes naplóüzenetet tartalmazza
`org.hibernate.SQL`	Végrehajtott SQL utasítások
`org.hibernate.type.descriptor.sql`	JDBC paraméterek értékei és JDBC eredményekből kinyert értékek
`org.hibernate.tool.hbm2ddl`	Végrehajtott DDL utasítások

Naplózás (3)

- Az SQL üzenetek formázása a `persistence.xml` állományban írható elő az alábbi módon:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        version="2.2">
  <persistence-unit name="...">
    <!-- ... -->
    <properties>
      <!-- ... -->
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.use_sql_comments" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

További ajánlott irodalom

- Christian Bauer, Gavin King, Gary Gregory. *Java Persistence with Hibernate*. 2nd edition. Manning, 2015. https://www.manning.com/books/java-persistence-with-hibernate-second-edition
- Mike Keith, Merrick Schincariol. *Pro JPA 2*. 2nd edition. Apress, 2013. https://www.apress.com/gp/book/9781430249276
- Mike Keith, Merrick Schincariol, Massimo Nardone. *Pro JPA 2 in Java EE 8*. Apress, 2018. https://www.apress.com/us/book/9781484234198
- Yogesh Prajapati, Vishal Ranapariya. *Java Hibernate Cookbook*. Packt Publishing, 2015. https://www.packtpub.com/application-development/java-hibernate-cookbook
- James Sutherland, Doug Clarke. *Java Persistence*. Wikibooks.org, 2013. https://en.wikibooks.org/wiki/Java_Persistence