Java Persistence API

Jeszenszky, Péter
University of Debrecen, Faculty of Informatics
jeszenszky.peter@inf.unideb.hu

Kocsis, Gergely
University of Debrecen, Faculty of Informatics
kocsis.gergely@inf.unideb.hu

Definitions

- *Persistence*
- *Data access object* – DAO
- *Domain model*
- *Anemic domain model*
- *Plain old Java object* – POJO
- JavaBean
- *object-relational mapping* – ORM

Persistence

- Meaning: the state of occurring or existing beyond the usual, expected, or normal time (http://www.merriam-webster.com/dictionary/persistence)
- In computer science it is used to data that overlives the process that created it.

How to implement persistence

- Java provides several different ways to implement persistence:
  - File management
  - Java Architecture for XML Binding (JAXB)
  - JDBC
  - Object serialization: see `java.io.Serializable` interface http://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
  - …
- Hereafter we focus on storing data in relational databases

Data access object (DAO)

- It defines an interface which through persistence operations can be executed on a given entity
- See:
  - Deepak Alur, Dan Malks, John Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd edition. Prentice Hall, 2003.
  - *Core J2EE Patterns – Data Access Object* http://corej2eepatterns.com/DataAccessObject.htm

Domain model

- Object model of a specific domain. It contains data and behavion as well. http://martinfowler.com/eaaCatalog/domainModel.html
  - See:
    - Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

Anemic domain model

- A domain model that does not contain behavior
  - Sometimes it is treated as an antipattern

Martin Fowler

- See:
  - Martin Fowler: *AnemicDomainModel*. http://www.martinfowler.com/bliki/AnemicDomainModel.html

POJO – Plain Old Java Object

- A concept by Rebecca Parsons, Josh MacKenzie and Martin Fowler (2000)
- It is a simple Java object for which no restrictions are given
  - E.g. There no mandatory interfaces to implement
- See:
  - Martin Fowler: *POJO*. http://www.martinfowler.com/bliki/POJO.html

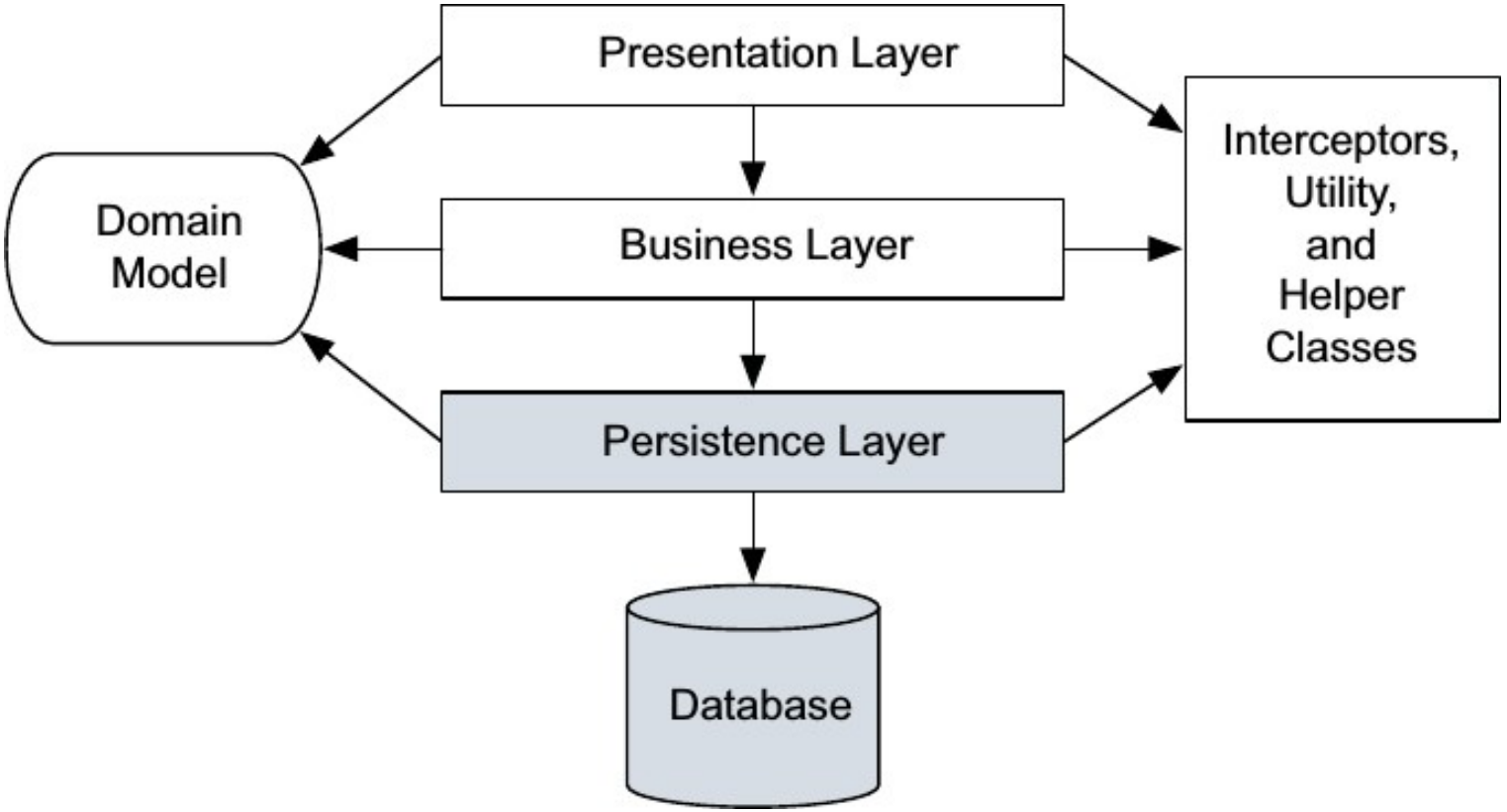
JavaBean

- A class implementing the `java.io.Serializable` interface. It has to contain a default constructor (parameterless) and has to provide getters and setters for its fields
  - See:
    - *JavaBeans Specification 1.01 (Final Release)* (August 8, 1997) http://www.oracle.com/technetwork/articles/javaee/spec-136004.html
    - *The Java Tutorials – Trail: JavaBeans* http://docs.oracle.com/javase/tutorial/javabeans/
    - Stephen Colebourne. *The JavaBeans specification*. November 28, 2014. http://blog.joda.org/2014/11/the-javabeans-specification.html

Object-relational mapping – ORM

- It provides conversion between object oriented programming languages and relational database tables
  - An implementation of storing objects of domain models in relational database tables

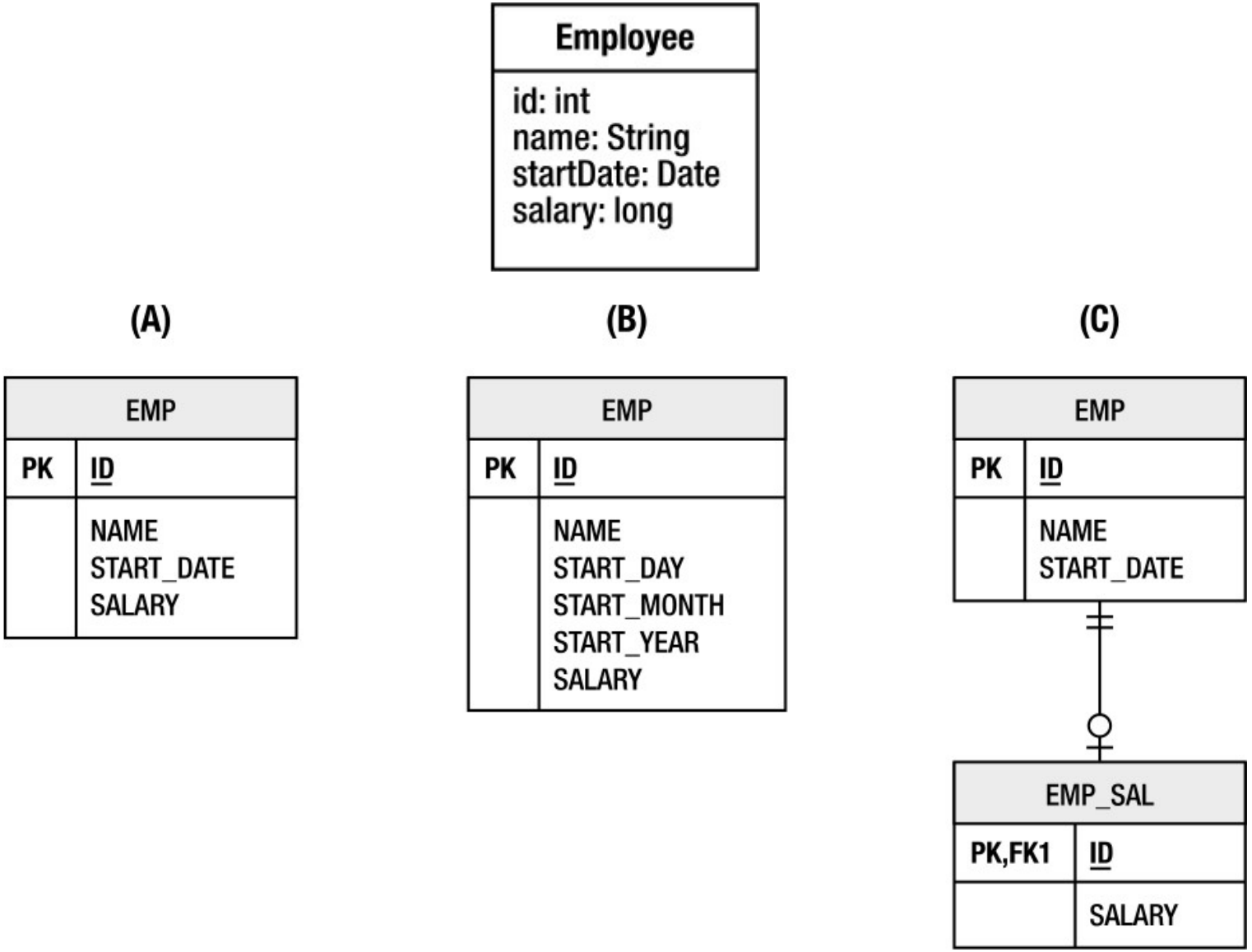
Layered application architecture



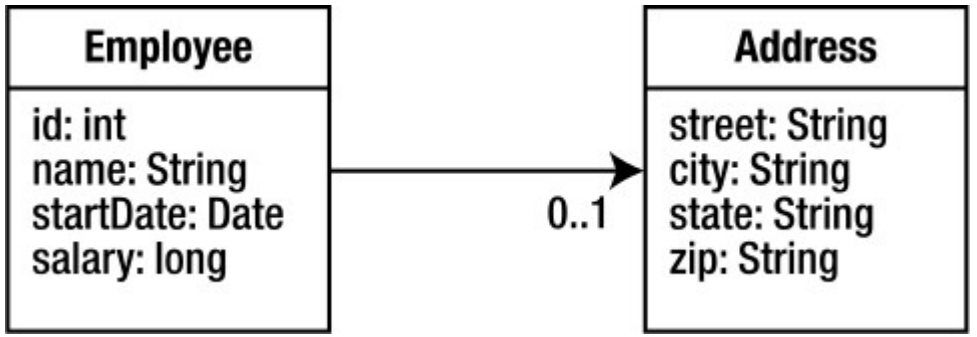
Paradign collision

- *Object-relational impedance mismatch*:
  - It means the difficulties arising as a result of the differences between the two different world
  - There are different concepts and sometimes there is no suitable pair of one in the other world
    - Example:
      - Associations: In order to represent N:M connections a connecting table is required. However in the object model this concept is not present

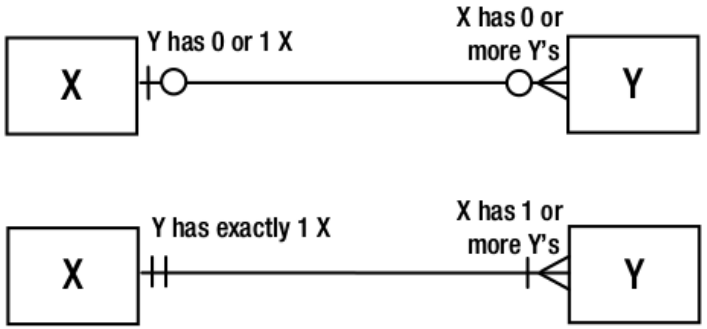
Implementation of Object-relational mapping (1)



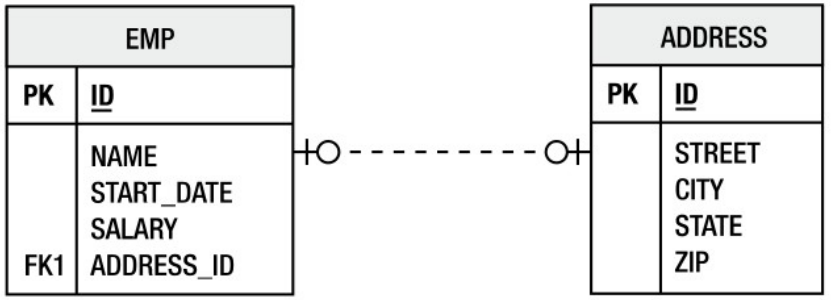
Implementation of Object-relational mapping (2)



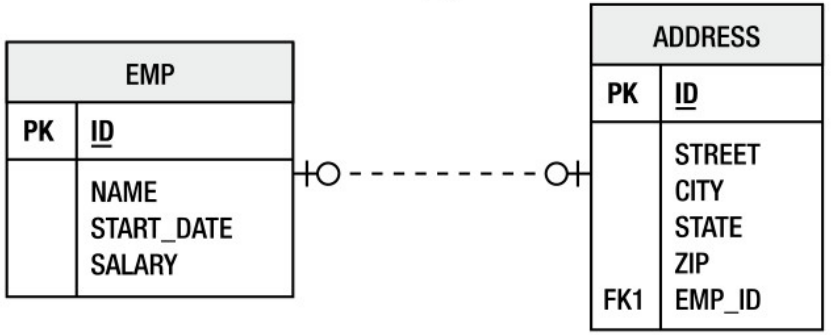
Legend:



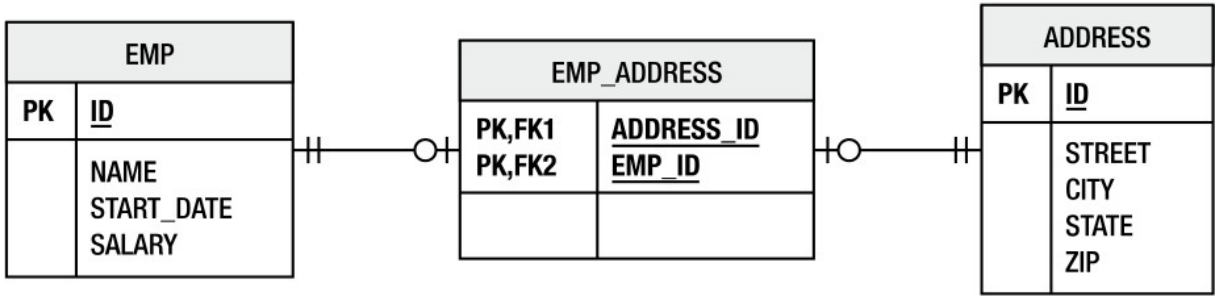
(A)



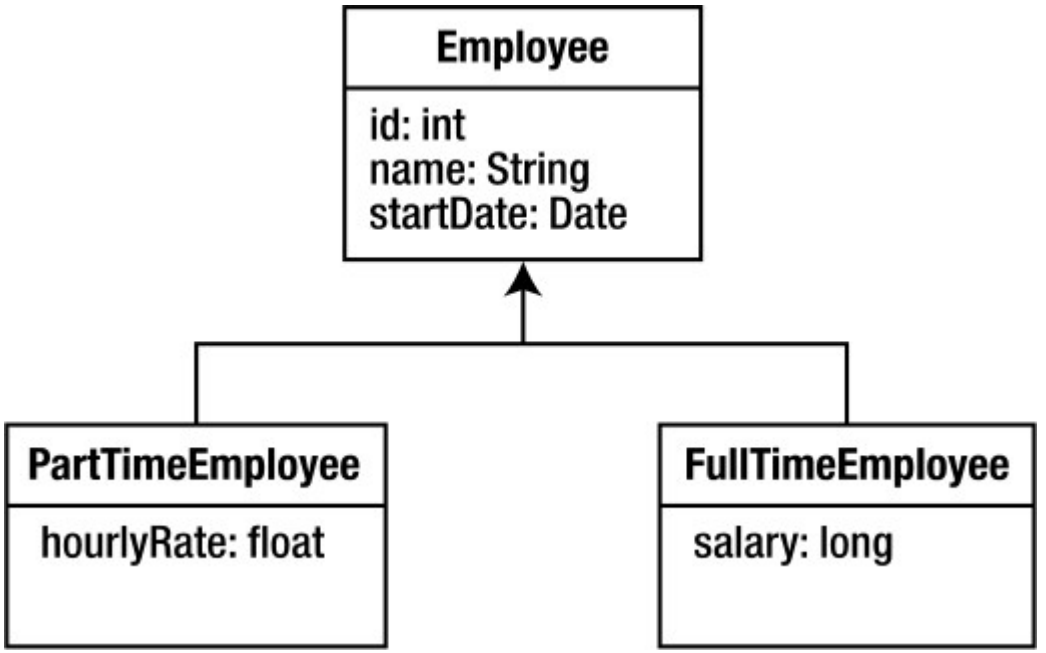
(B)



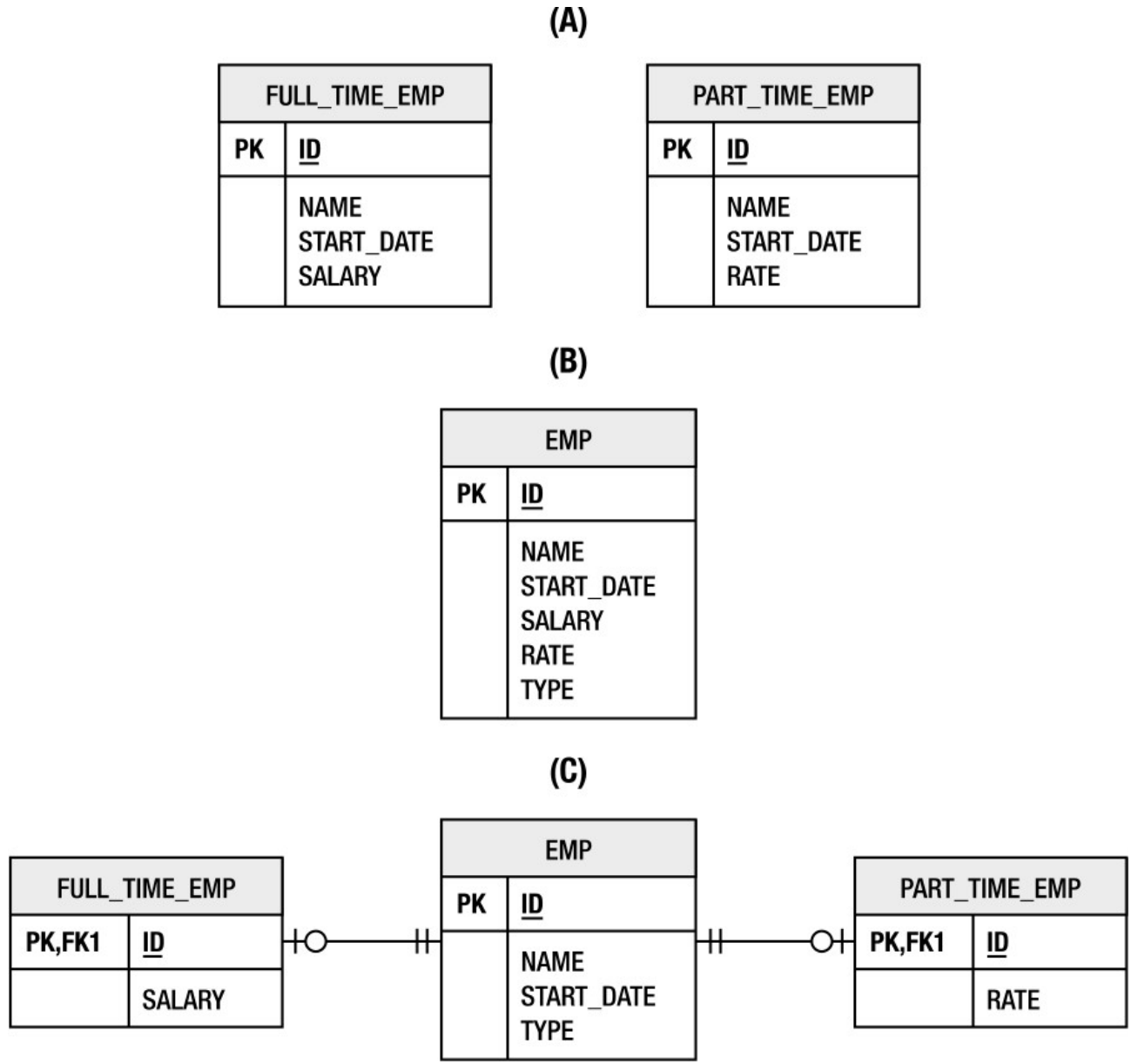
(C)



Implementation of Object-relational mapping (3)



Implementation of Object-relational mapping (4)



Earlier implementations of object persistence in Java (1)

- **Producer dependent solutioins**:
  - Open source Free solutions (e.g. Castor, Hibernate)
  - Commercial solutions (TopLink)
- **JDBC**
- ***Data mappers***:
  - Parial solutions half way between JDBC and full ORM solutions. The applicaion developer provides the SQL statements
    - Example: MyBatis (license: Apache Software License) http://www.mybatis.org/mybatis-3/
  - See:
    - Martin Fowler: *Data Mapper*. http://martinfowler.com/eaaCatalog/dataMapper.html
    - Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

Earlier implementations of object persistence in Java (2)

- ***Enterprise JavaBeans – Entity Beans****:*
  - Since the introduction of EJB 3.0 in 2006 it is deprecated and obsolated by JPA
- ***Java Data Objects*** (***JDO***):
  - Example:
    - Apache JDO (licenc: Apache License v2) https://db.apache.org/jdo/
    - DataNucleus Access Platform (licenc: Apache License v2) http://www.datanucleus.org/

Java Persistence API (1)

- POJO-based ORM solution of Java object persistence
  - Originally introduced in Enterprise JavaBeans (EJB) 3.0 specification in 2006 as a part of Java EE 5
  - Current version: 2.1 introduced in 2013

Java Persistence API (2)

- Contained by the following packages:
  - `javax.persistence`
    https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html
  - `javax.persistence.criteria`
    https://docs.oracle.com/javaee/7/api/javax/persistence/criteria/package-summary.html
  - `javax.persistence.metamodel`
    https://docs.oracle.com/javaee/7/api/javax/persistence/metamodel/package-summary.html
  - `javax.persistence.spi`
    https://docs.oracle.com/javaee/7/api/javax/persistence/spi/package-summary.html

Specification

- *JSR 338: Java Persistence 2.1 (Final Release)* (22 May, 2013) https://jcp.org/en/jsr/detail?id=338

Properties (1)

- **POJO-based**
- ***Non-intrusiveness***
  - The persistence API is separated from the persistent classes
    - The business logic of the application calls it. It passes the persistent objects as parameters to the API.
- **Object oriented query language**
  - Java Persistence Query Language (JPQL)

Properties (2)

Mobil entities

- The persistent objects can be transmitted from one Java virtual machine to another

- **Simple configuration**
  - There are default settings for everything. Only the exceptional cases needed to be configured (*configuration by exception*)
  - For the mapping meta-data can be provided by annotations or in a separate XML document
- **No application server is needed**
  - Can be used in Java SE as well (not only in Java EE)

JPA implementations

- Open Source Free softwares:
  - DataNucleus (licenc: Apache License v2) http://www.datanucleus.org/
  - EclipseLink (license: Eclipse Public License) http://www.eclipse.org/eclipselink/
    - Reference implementation of JPA 2.1
    - GlassFish Server uses it a default persistence solution
  - Hibernate (licenc: GNU LGPL v2.1) http://hibernate.org/
    - WildFly uses it a default persistence solution https://docs.jboss.org/author/display/WFLY10/JPA+Reference+Guide

IDE support

- Eclipse:
  - Dali Java Persistence Tools (license: Eclipse Public License) https://eclipse.org/webtools/dali/
    - A parrt of Eclipse IDE for Java EE Developers
- NetBeans:
  - NetBeans JPA Modeler (license: Apache License v2) http://jpamodeler.github.io/ http://plugins.netbeans.org/plugin/53057/jpa-modeler

Hibernate components

- Hibernate OGM http://hibernate.org/ogm/
  - JPA support for NoSQL databases (e.g. MongoDB, Neo4j)
- Hibernate ORM http://hibernate.org/orm/
  - JPA implementation
- Hibernate Search http://hibernate.org/search/
  - Full text search support
- Hibernate Validator http://hibernate.org/validator/
  - Annotation-based solution to validate restrictions on ebjects
- Hibernate Tools http://hibernate.org/tools/
  - For developers (Eclipse extensions and Apache Ant task)
- …

Supported Database management systems

- See `org.hibernate.dialect` package
  http://docs.jboss.org/hibernate/orm/5.1/javadocs/org/hibernate/dialect/package-summary.html
  - Microsoft SQL Server, Oracle, MySQL, PostgreSQL, Apache Derby (Java DB), H2, HSQLDB, …

Availability

- Available in Maven central repository (last stable version `5.1.0.Final`)
  - Products:
    - `org.hibernate:hibernate-core:5.1.0.Final`
    - `org.hibernate:hibernate-java8:5.1.0.Final`
    - `org.hibernate:hibernate-tools:5.1.0.Alpha3`
    - `org.hibernate:hibernate-gradle-plugin:5.1.0.Final`
    - …

Maven support

- Hibernate Maven Plugin
  http://juplo.de/hibernate-maven-plugin/
  - Available in Maven central repository:
    `de.juplo:hibernate-maven-plugin:2.0.0`

NHibernate

- NHibernate (programming language: C#, license: GNU GPL v2.1) http://nhibernate.info/
  - A Hibernate port for .NET platform

Entity

- A lightweight persistent domain specific object

Entity class

- A class marked by `javax.persistence.Entity` annotation or a class marked as an entity class in the XML descriptor file
- It has a mandatory `public` or `protected` default constructor
- It has to be a top level class. Must not be `enum` or interface
- For the class, its methods and persistent instance variables `final` modifier cannot be used

Persistent fields and attributes (1)

- The persistent state of an object is represented by the instance variables that fulfill JavaBean properties.
- Instance variables can be reached only through the methods by the clients of the entity
- The instance variables of the entity can be reached by the persistence provider running environment through the `getters` and `setters`
  - The visibility of instance variables can be `private`, `protected` or package private (no modifier)
  - If the instance variables can be reached only through the getters and setters, the visibility of these have to be `public` or `protected`

Persistent fields and attributes (2)

- Collection type persistent fields and attributes have to be of type `java.util.List`, `java.util.Map` or `java.util.Set`

Elsődleges kulcsok és entitás identitás

- All entities must have a primary key
- The primary key has to be one or more instance variables or attributes of the entity class
  - The primary key can be simple or compound (composite)
- The type of the instance variables and attributes forming the primary key can be:
  - Primitive types and their respective Object types, `java.lang.String`, `java.util.Date`, `java.sql.Date`, `java.math.BigDecimal`, `java.math.BigInteger`

EntityManager (1)

- The `javax.persistence.EntityManager` interface provides an API for persistence operations https://docs.oracle.com/javaee/7/api/javax/persistence/EntityManager.html
  - Example `find()`, `persist()` and `remove()` methods

EntityManager (2)

- When an `EntityManager` gets a reference to an entity object we say that the `EntityManager` **manages** the object
  - The reference to the object can be given as a result of a method call or by reading from a database
- The objects manages by the `EntityManager` are called the **Persistence context**
- An `EntityManager` manages specific type entities stored in a specific database
  - The types of the managed entities are defined by a **persistence unit**

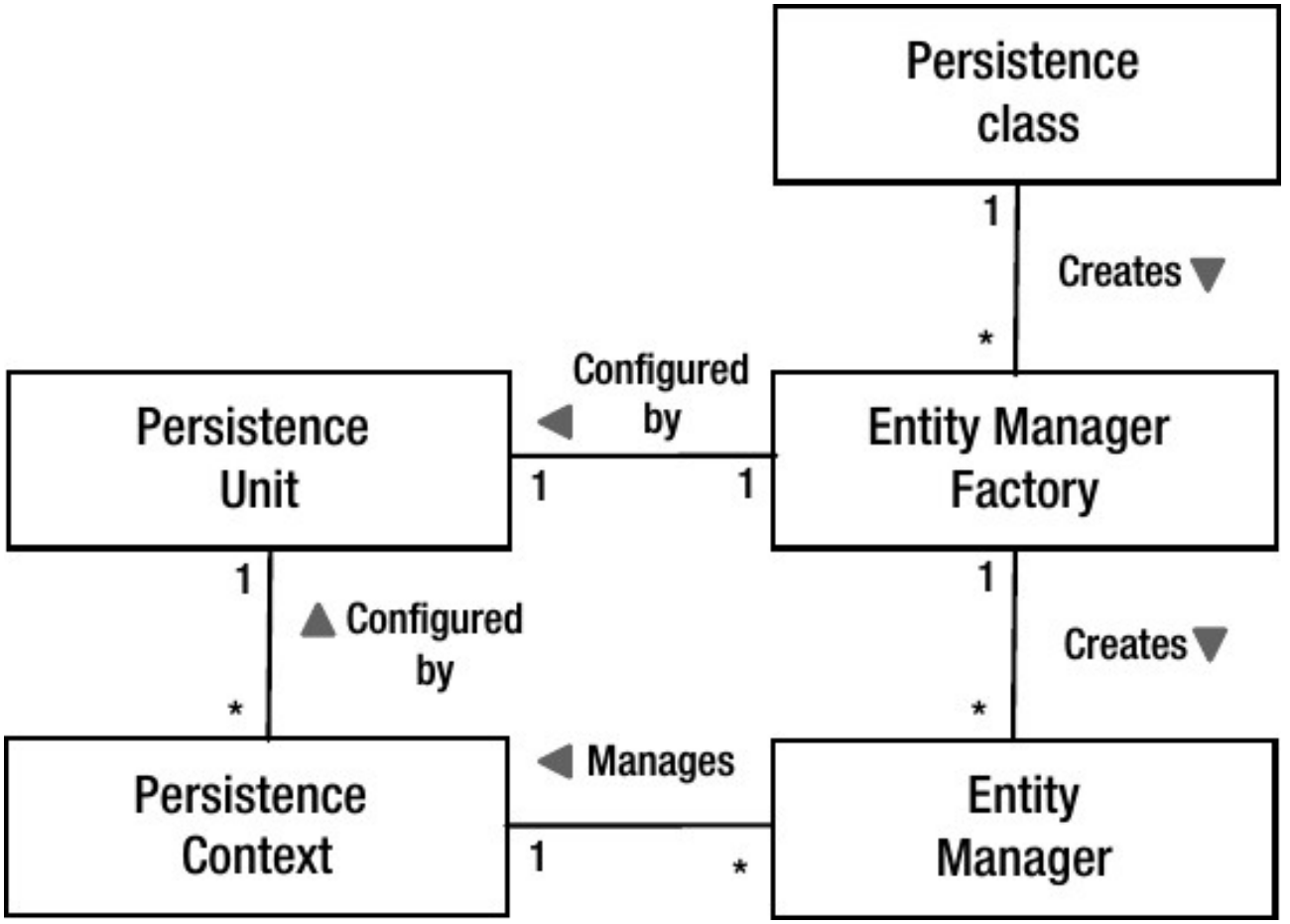
Persistence unit

- A Persistence Unit is a logical grouping of persistable classes along with their related settings that are mapped to the same database.

EntityManagerFactory

- The `javax.persistence.EntityManagerFactory` interface can provide `EntityManager` objects for a given persistence unit https://docs.oracle.com/javaee/7/api/javax/persistence/EntityManagerFactory.html
  - In Java SE applications an instance of the `EntityManagerFactory` object can be created by the use of the `createEntityManagerFactory()` method of the `javax.persistence.Persistence` class https://docs.oracle.com/javaee/7/api/javax/persistence/Persistence.html

JPA concepts



JPA „Hello, world!” program (1)

- `pom.xml`:

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.1.0.Final</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>5.1.0.Final</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.12.1.1</version>
        <scope>runtime</scope>
    </dependency>
```

JPA „Hello, world!” 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 „Hello, world!” program (3)

- `HelloJPA.java`:

```
package hello;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import hello.model.Message;

public class HelloJPA {

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

JPA „Hello, world!” program (4)

- `META-INF/persistence.xml`:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
version="2.1">
    <persistence-unit name="HelloPU">
        <class>hello.model.Message</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
                value="org.apache.derby.jdbc.EmbeddedDriver"/>
            <property name="javax.persistence.jdbc.url"
                value="jdbc:derby:testDB;create=true"/>
            <property name="javax.persistence.schema-
generation.database.action" value="drop-and-create"/>
        </properties>
    </persistence-unit>
</persistence>
```

JPA „Hello, world!” program (5)

- Modification in the database:

```
EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("HelloPU");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
List<Message> messages = em.createQuery("select m from
                  hello.model.Message m").getResultList();
assert messages.size() == 1;
assert messages.get(0).getText().equals("Hello, world!");
messages.get(0).setText("Hello, JPA!");
em.getTransaction().commit();
em.close();
emf.close();
```

The same example without annotations (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;
    }
}
```

The same example without annotations (2)

- `META-INF/persistence.xml`:

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
version="2.1">
    <persistence-unit name="HelloPU">
        <mapping-file>META-INF/orm.xml</mapping-file>
        <class>hello.model.Message</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
                value="org.apache.derby.jdbc.EmbeddedDriver"/>
            <property name="javax.persistence.jdbc.url"
                value="jdbc:derby:testDB;create=true"/>
            <property name="javax.persistence.schema-
generation.database.action" value="drop-and-create"/>
        </properties>
    </persistence-unit>
</persistence>
```

The same example without annotations (3)

- `META-INF/orm.xml`:

```
<entity-mappings
xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm",ver
sion="2.1">
    <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="name"/>
        </attributes>
    </entity>
</entity-mappings>
```

A more complex example (1)

- `Employee.java`:

```
@Entity
public class Employee {

    @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; }

}
```

A more complex example (2)

- `EmployeeService.java`:

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

public class EmployeeService {

    protected EntityManager em;

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

    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;
    }
```

A more complex example (3)

- `EmployeeService.java` (part 2):

```
public void removeEmployee(int id) {
    Employee emp = findEmployee(id);
    if (emp != null) {
        em.remove(emp);
    }
}

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

A more complex example (4)

- `EmployeeService.java` (part 3):

```
    public Employee findEmployee(int id) {
        return em.find(Employee.class, id);
    }

    public List<Employee> findAllEmployees() {
        TypedQuery<Employee> query = em.createQuery(
                "SELECT e FROM Employee e", Employee.class);
        return query.getResultList();
    }
}
```

A more complex example (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);

        // create or save an employee
        em.getTransaction().begin();
        Employee emp = service.createEmployee(158, "John Doe", 45000);
        em.getTransaction().commit();
        System.out.println("Persisted " + emp);
```

A more complex example (6)

- `EmployeeTest.java` (part 2):

```
// load an employee
emp = service.findEmployee(158);
System.out.println("Found " + emp);

// load all employees
List<Employee> emps = service.findAllEmployees();
for (Employee e : emps) {
    System.out.println("Found employee: " + e);
}
```

A more complex example (7)

- `EmployeeTest.java` (part 3):

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

        // delete employee
        em.getTransaction().begin();
        service.removeEmployee(158);
        em.getTransaction().commit();
        System.out.println("Removed Employee 158");

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

Java 8 date and time types (1)

- `pom.xml`:

```
<dependencies>
    ...
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-java8</artifactId>
        <version>5.1.0.Final</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Java 8 date and time types (2)

- `Employee.java`:

```
@Entity
public class Employee {

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

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

Change table name

- `Employee.java`:

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

Change default column name

- `Employee.xml`:

```
@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;
    // ...
}
```

Lazy loading

- `Employee.xml`:

```
@Entity
public class Employee {

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

The use of enums

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

@Entity
public class Employee {

    @Id private long id;

    private EmployeeType
type;
    // ...
}
```

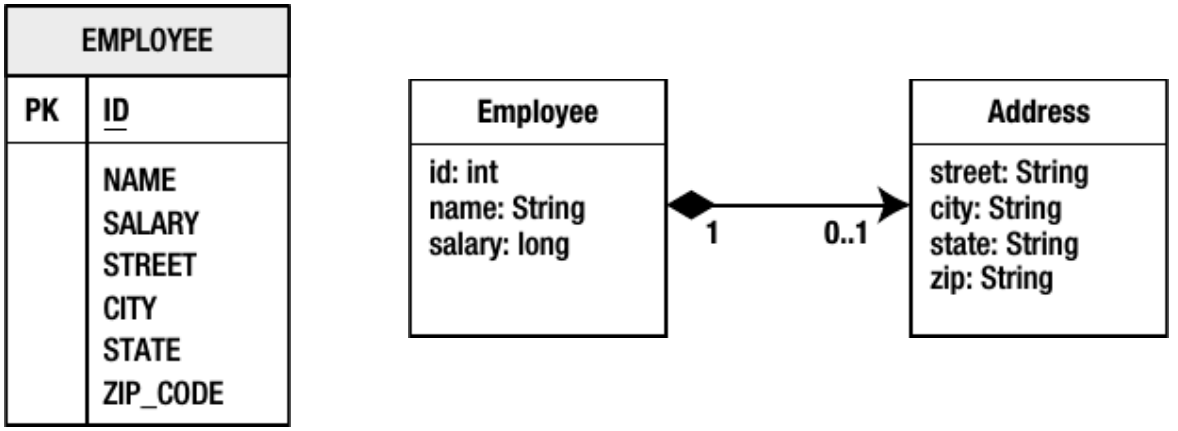
```
public enum EmployeeType {
    FULL_TIME_EMPLOYEE,
    PART_TIME_EMPLOYEE,
    CONTRACT_EMPLOYEE
}

@Entity
public class Employee {

    @Id private long id;

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

Embedded objects (1)

- Objects that do not have identity, but they are a part of another entity

Embedded objects (2)

- Example:



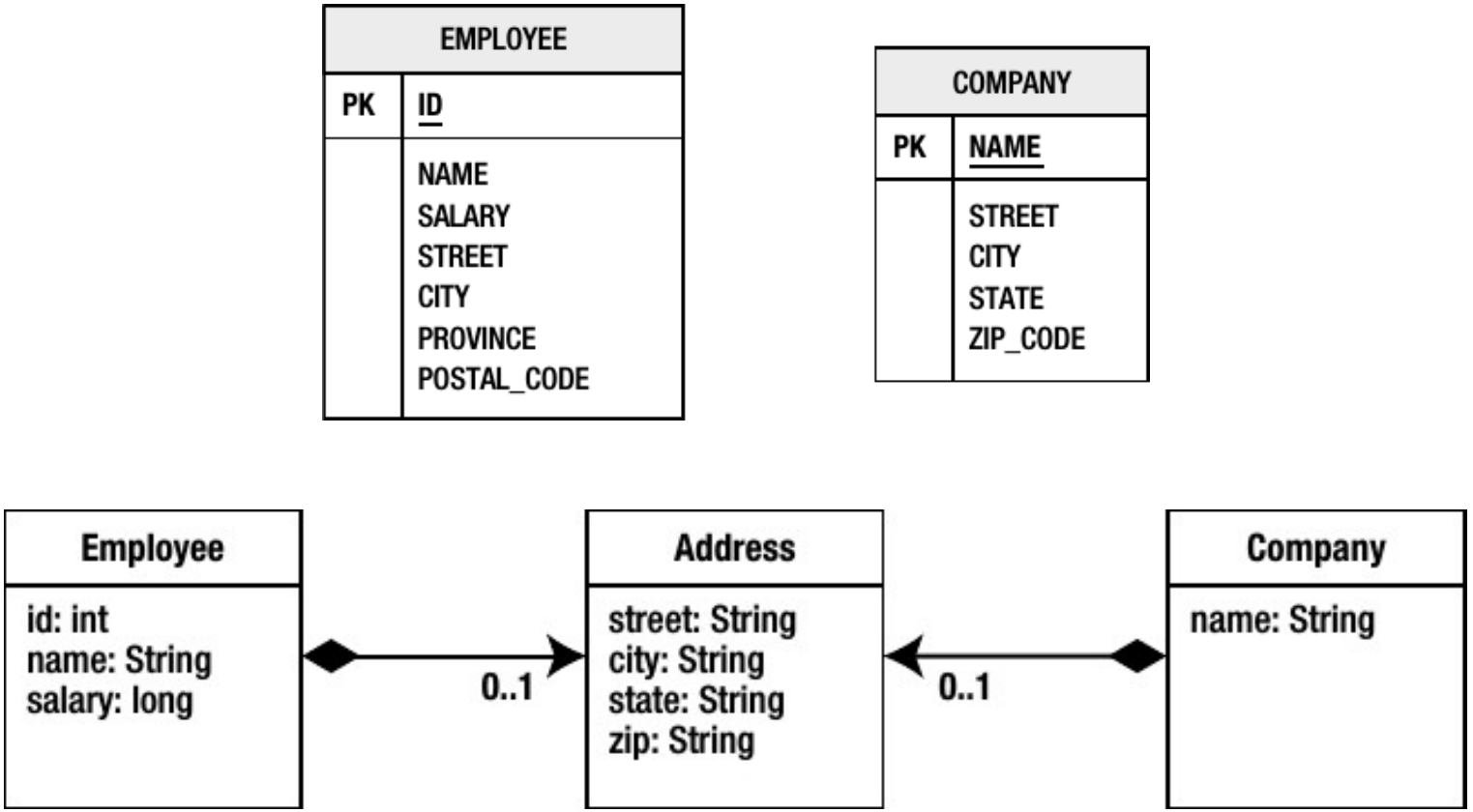
Embedded objects (3)

- Example:

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

Embedded objects (4)

- Example:



Embedded objects (5)

- Example:

```
@Entity
public class Employee {

    @Id private long 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;
    // ...
}
```

Restrictions (1)

- The restrictions of the `javax.validation.constraints` package can be used:
  - Bean Validation: a part of Java EE (see `javax.validation` package and sub-packages)
    - *JSR 349: Bean Validation 1.1 (Final Release)* (24 May 2013) https://jcp.org/en/jsr/detail?id=349
    - Referencia implementáció: Hibernate Validator http://hibernate.org/validator/

Restrictions (2)

- Example:

```
public class Employee {

    @NotNull
    private int id;

    @NotNull(message="Employee name must be specified")
    @Size(max=40)
    private String name;

    @Past
    private Date startDate;
    // ...
}
```

Further reading

- 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 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013. http://www.apress.com/9781430249269
- Yogesh Prajapati, Vishal Ranapariya. *Java Hibernate Cookbook – Over 50 recipes to help you build dynamic and powerful real-time Java Hibernate applications*. Packt Publishing, 2015.
- James Sutherland, Doug Clarke. *Java Persistence*. Wikibooks.org, 2013. https://en.wikibooks.org/wiki/Java_Persistence