Java Persistence API

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

Last modified: January 21, 2020

Fundamental Concepts

- Persistence
- Data access object (DAO)
- Domain model
- Anemic domain model
- Plain old Java object (POJO)
- JavaBean
- Object-relational mapping (ORM)

Persistence

- Meaning: https://en.wiktionary.org/wiki/persistence
- In Computer Science:
  - Persistent data is data that survives the process that created it.

Implementing Persistence

- Java provides various ways to implement persistence:
  - File I/O
  - Java Architecture for XML Binding (JAXB)
  - JDBC
  - Object serialization: see the `java.io.Serializable` interface https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/Serializable.html
  - …
- In the following, we restrict ourselves to storing data in relational databases.

Data access object (DAO)

- An object that provides access to a data source in such a way that completely hides implementation details of the data source behind the object's interface.
  - Defines an interface for performing persistence operations related to a particular entity.
- See:
  - Deepak Alur, Dan Malks, John Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd edition. Prentice Hall, 2003. http://corej2eepatterns.com/
    - *Core J2EE Patterns – Data Access Object* http://corej2eepatterns.com/DataAccessObject.htm

Domain Model

- An object model of the domain that incorporates both behavior and data. https://martinfowler.com/eaaCatalog/domainModel.html
  - See:
    - Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

Anemic Domain Model

- A domain model void of behavior.
  - According to Martin Fowler, it is an anti-pattern.
- See:
  - Martin Fowler: *AnemicDomainModel*. https://www.martinfowler.com/bliki/AnemicDomainModel.html

POJO

- A term coined by Rebecca Parsons, Josh MacKenzie, and Martin Fowler.
- An ordinary Java object on which no restrictions are imposed.
  - For example, it does not implement certain interfaces.
- See:
  - Martin Fowler: *POJO*. https://www.martinfowler.com/bliki/POJO.html

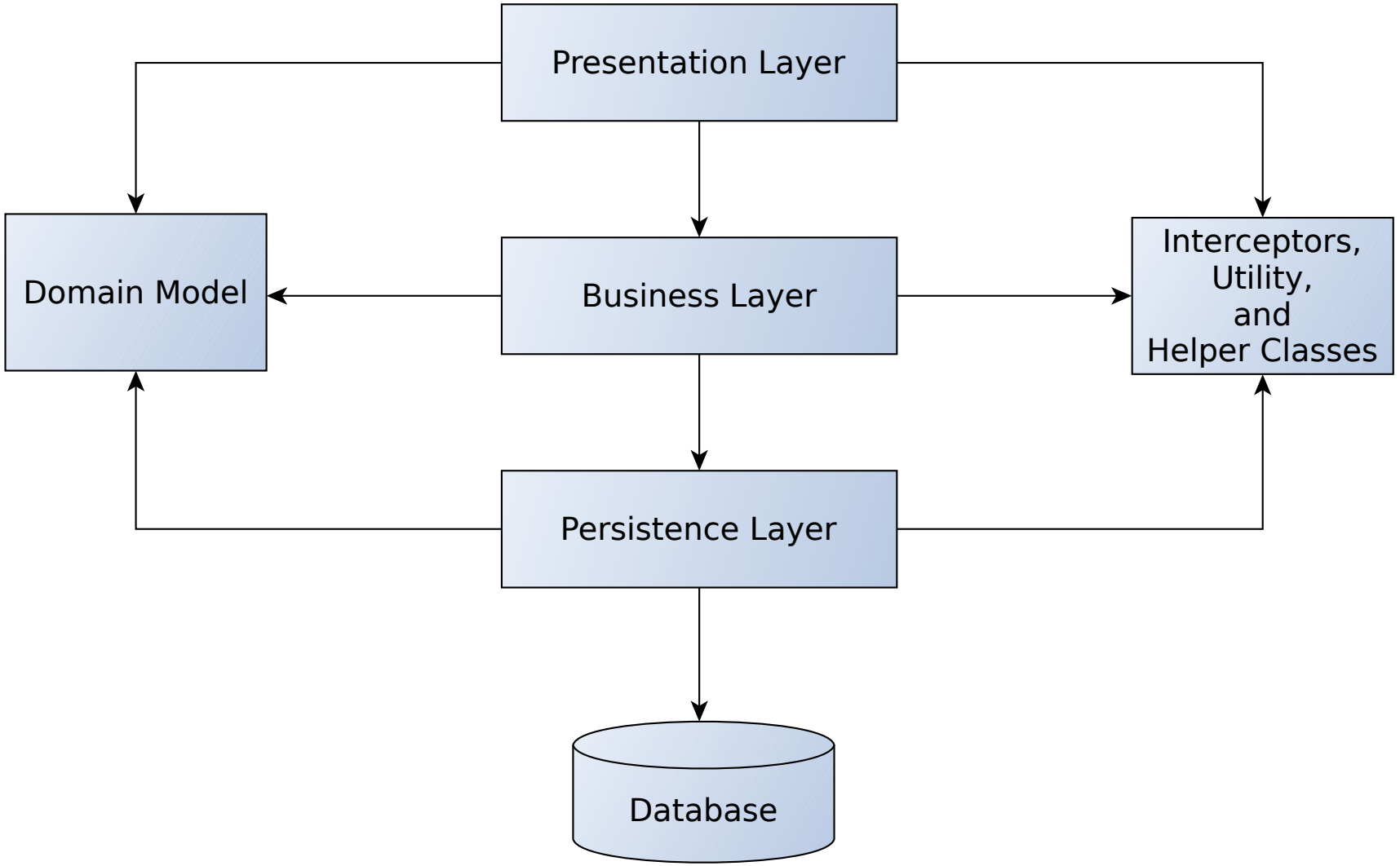
JavaBean

- A class that implements the `java.io.Serializable` interface, has a no-argument constructor and has getters and setters for its properties.
  - See:
    - *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

Object-Relational Mapping

- Conversion between objects of an object oriented programming language and tables of a relational database.
  - Storing domain model objects in relational database tables.

Layered Application Architecture

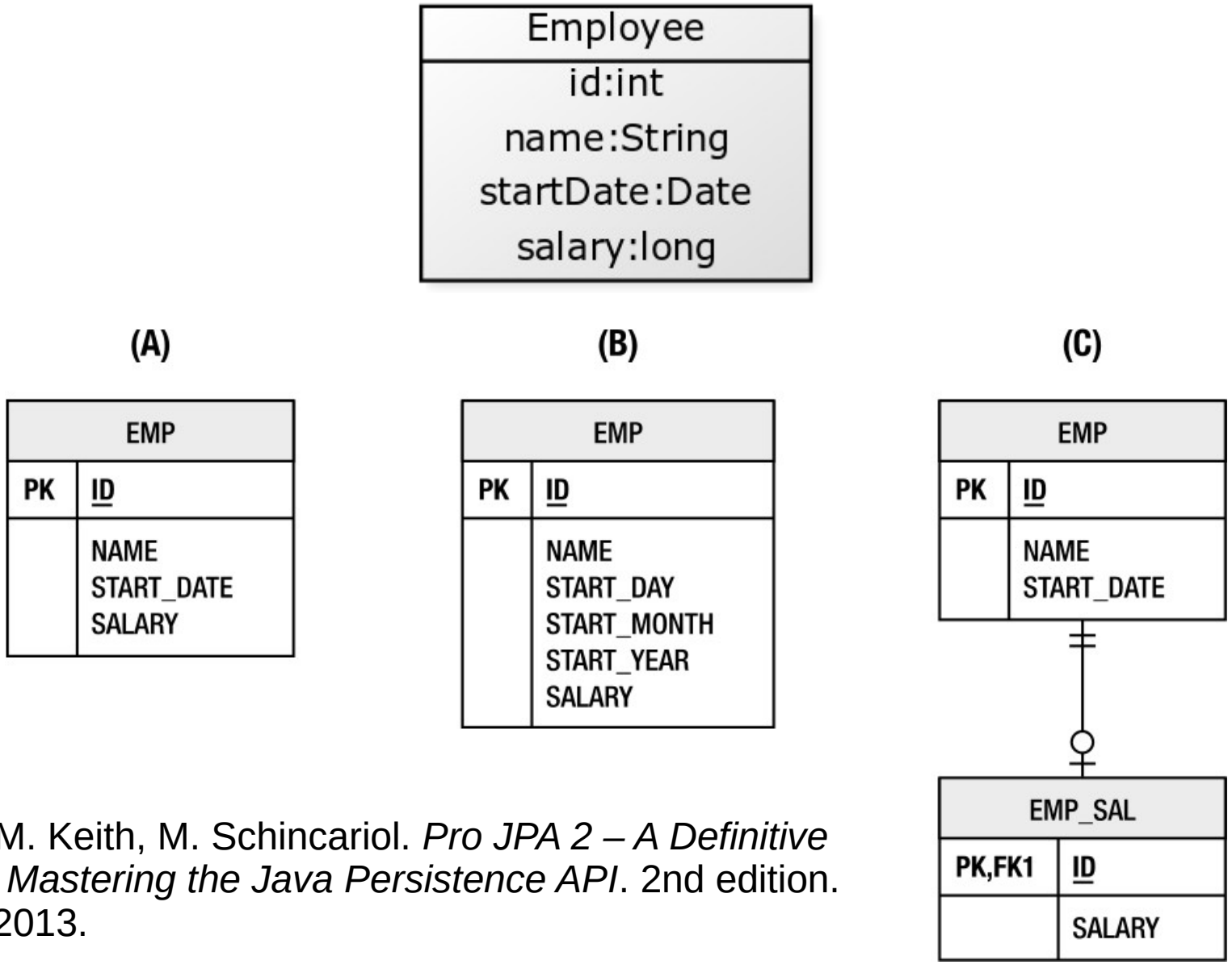


Source: Christian Bauer, Gavin King, Gary Gregory. *Java Persistence with Hibernate*. 2nd edition. Manning, 2015.

Impedance Mismatch

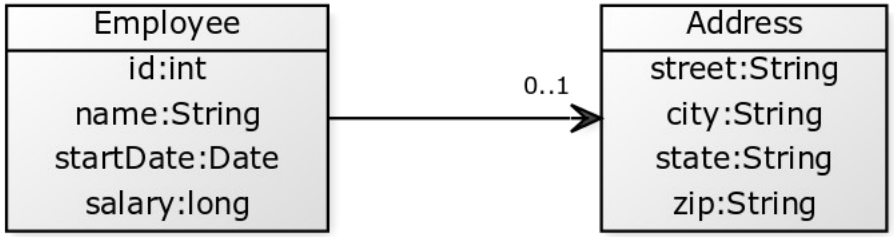
- Object-relational impedance mismatch:
  - The term expresses the problems and difficulties arising from the differences between the two data model.
  - The two data models are based on different concepts, some of which do not have a counterpart in the other.
    - Example:
      - Associations: representing an *N*:*M* relationship requires a join table that is not present in the object model.

Implementing Object-Relational Mapping (1)

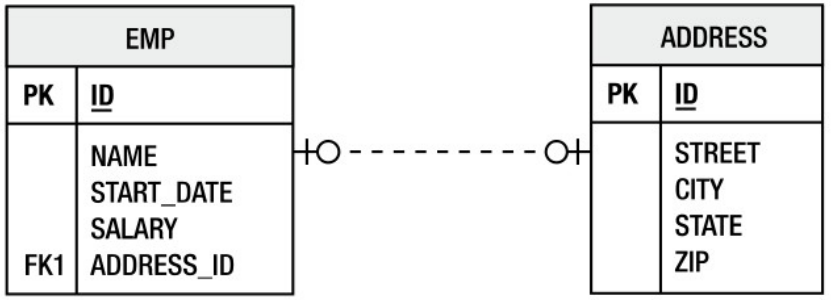


Source: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

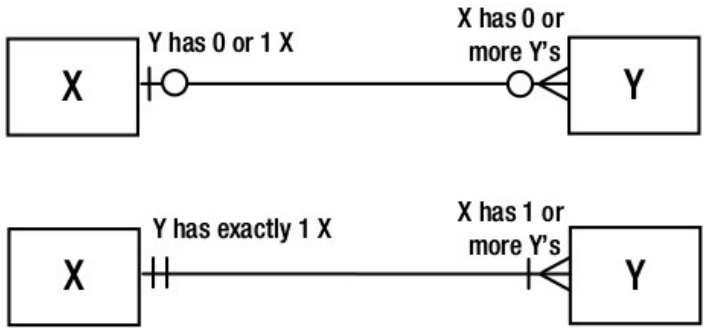
Implementing Object-Relational Mapping (2)



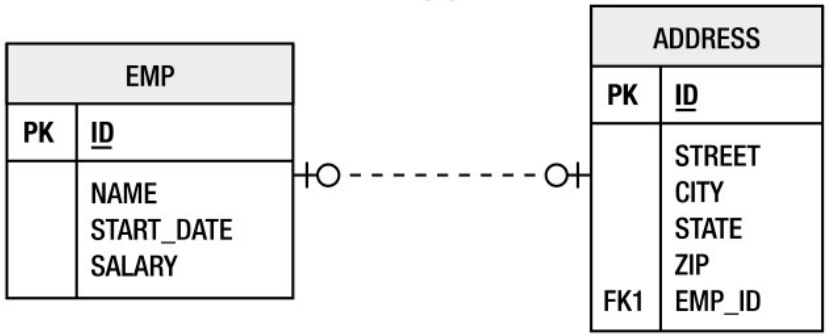
(A)



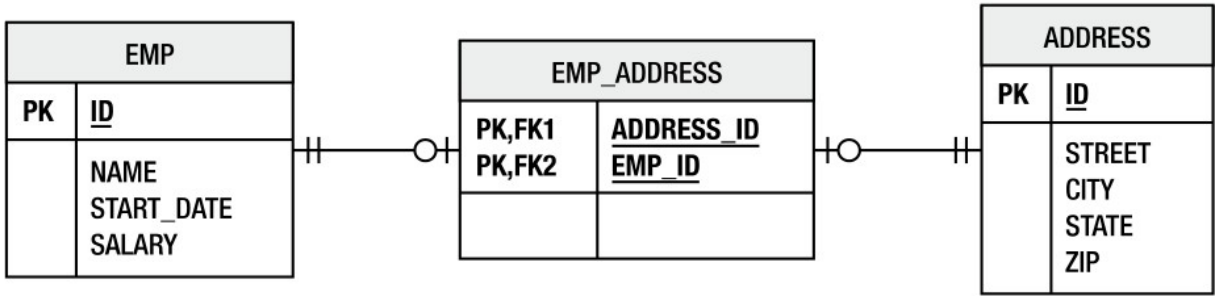
Legend:



(B)

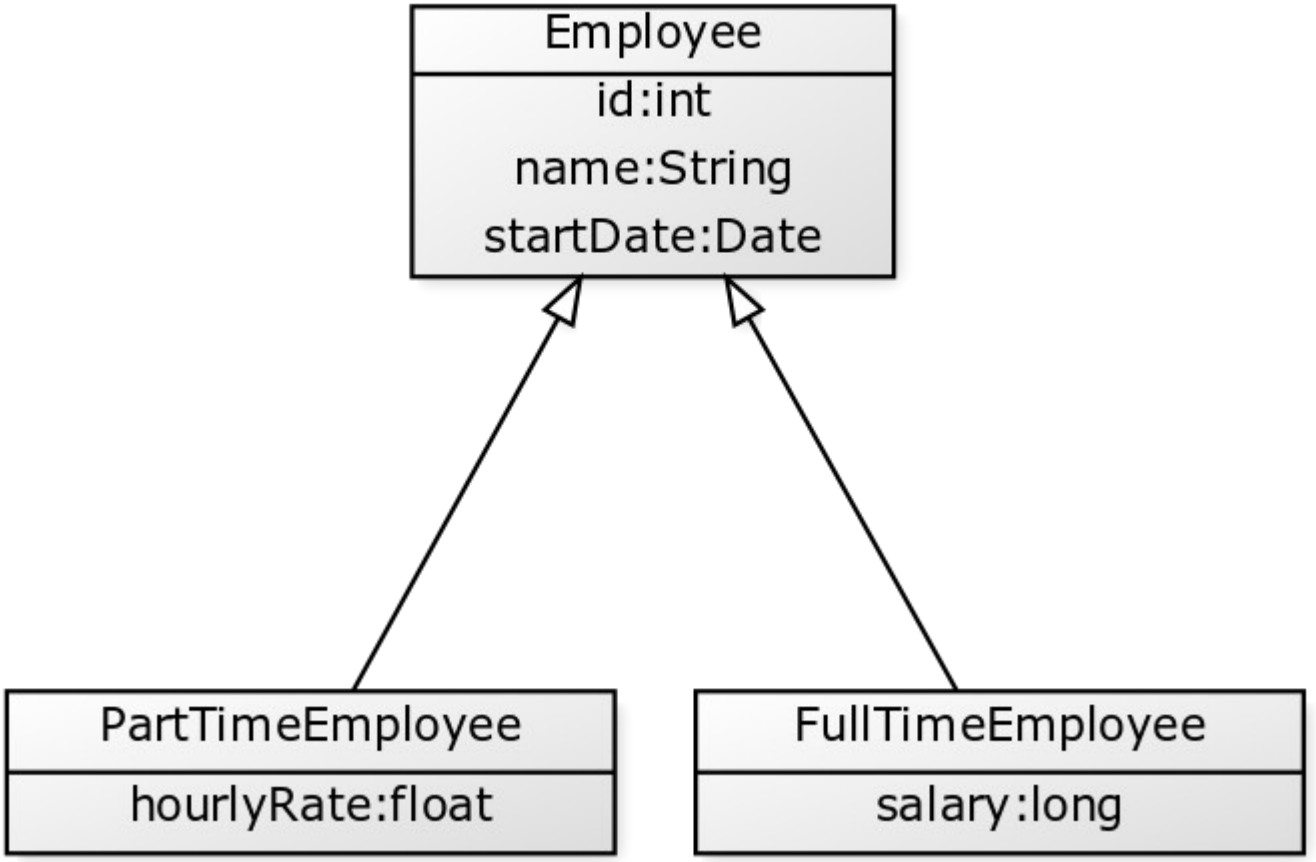


(C)



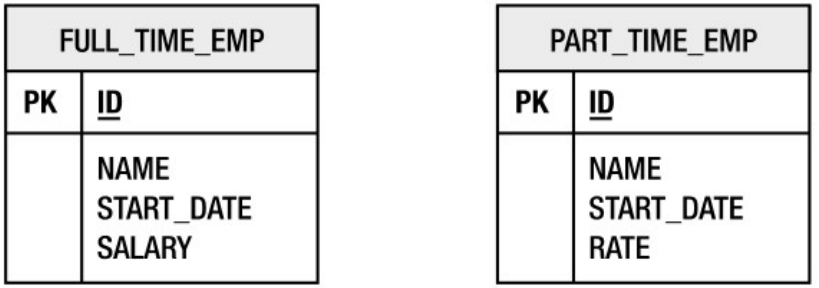
Source: M. Keith, M. Schincariol. *Pro JPA 2 – A Definitive Guide to Mastering the Java Persistence API*. 2nd edition. Apress, 2013.

Implementing Object-Relational Mapping (3)

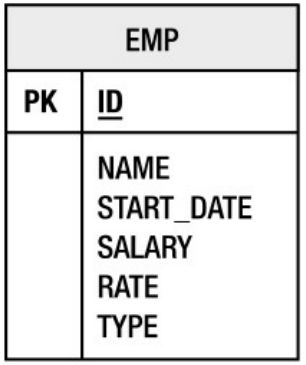


Implementing Object-Relational Mapping (4)

(A)

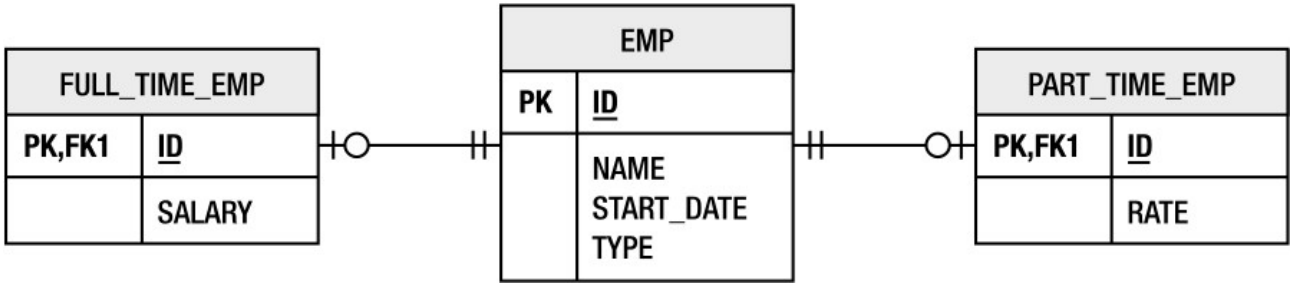


(B)



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

(C)



Earlier Solutions for Object-Relational Mapping (1)

- **JDBC**
- **Vendor-specific solutions**:
  - Free and open source software:
    - Hibernate ORM (license: LGPLv2.1) https://hibernate.org/orm/
  - Non-free software:
    - Oracle TopLink (license: proprietary) https://www.oracle.com/middleware/technologies/top-link.html

Earlier Solutions for Object-Relational Mapping (2)

- **Data mappers**:
  - They represent a partial solution halfway between JDBC and full ORM solutions, where the applications developer is responsible for providinig the SQL statements.
  - See:
    - Martin Fowler: *Data Mapper*. https://martinfowler.com/eaaCatalog/dataMapper.html
    - Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
  - Free and open source software:
    - Jdbi (license: Apache License 2.0) http://jdbi.org/
    - MyBatis (license: Apache License 2.0) http://www.mybatis.org/mybatis-3/

Earlier Solutions for Object-Relational Mapping (3)

- **Java Data Objects** (**JDO**):
  - Specification:
    - *JSR 243: Java Data Objects 3.0 (Maintenance Release 3)* (April 9, 2010) https://www.jcp.org/en/jsr/detail?id=243
  - Free and open source software:
    - Apache JDO (license: Apache License 2.0) https://db.apache.org/jdo/
    - DataNucleus Access Platform (license: Apache License 2.0) http://www.datanucleus.org/

Alternative ORM Solutions

- Free and open source software:
  - Apache Cayenne (license: Apache License 2.0)
    https://cayenne.apache.org/
  - Speedment (license: Apache License 2.0)
    https://github.com/speedment/speedment

Java Persistence API (1)

- A POJO-based ORM solution for Java object persistence.
  - Originally, it was introduced the *Enterprise JavaBeans* (EJB) 3.0 specification in 2006 as part of Java EE 5.
  - The current version is 2.2 that was published in 2017.

Java Persistence API (2)

- It is contained in the following packages (Java EE 8, Jakarta 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
    https://jakarta.ee/specifications/platform/8/apidocs/javax/persistence/spi/pa

Specification

- *JSR 338: Java Persistence 2.2 (Maintenance Release)* (July 17, 2017) https://jcp.org/en/jsr/detail?id=338
  - Major changes:
    - Several annotation types, e.g., `AttributeOverride`, `JoinColumn`, `NamedQuery`, are now marked as repeatable.
    - Support for the following types of the `java.time` package:
      - `java.time.LocalDate`
      - `java.time.LocalTime`
      - `java.time.LocalDateTime`
      - `java.time.OffsetTime`
      - `java.time.OffsetDateTime`
    - Adding the `Stream getResultStream()` default method to the `javax.persistence.Query` interface, and the `Stream<`*X*`> getResultStream()` method to the `javax.persistence.TypedQuery` interface, respectively.
- *Jakarta Persistence 2.2* https://jakarta.ee/specifications/persistence/2.2/

Characteristics

- **POJO-based**
- **Non-intrusiveness**
  - The persistence API is separated from persistent classes.
    - Methods of the persistence API are called by the application business logic, passing persistent objects as parameters.
- **Object-oriented query language**
  - Java Persistence Query Language (JPQL)
- **Mobile entities**
  - Persistent objects can be moved from one JVM to another.
- **Simple configuration**
  - The default configurations settings are applicable in most cases, settings must only be provided explicitly when the defaults must be overridden (configuration by exception).
  - ORM metadata can be provided by annotations or externally in XML documents.
- **No need for an application server**
  - Can be used in Java SE applications.

JPA 2.2 Implementations

- Free and open source software:
  - Apache OpenJPA (license: Apache License 2.0) http://openjpa.apache.org/
  - DataNucleus Access Platform (license: Apache License 2.0) http://www.datanucleus.org/
  - EclipseLink (license: Eclipse Public License v1.0/Eclipse Distribution License v1.0) http://www.eclipse.org/eclipselink/
  - Hibernate ORM (license: LGPLv2.1) http://hibernate.org/
    - It is the default persistence provider used by the WildFly applications server.
      - See: *WildFly Developer Guide – JPA Reference Guide* http://docs.wildfly.org/18/Developer_Guide.html#JPA_Reference_Guide

JPA Support for NoSQL Databases

- Free and open source software:
  - EclipseLink (license: Eclipse Public License v1.0/Eclipse Distribution License v1.0) https://www.eclipse.org/eclipselink/
    - See: *EclipseLink NoSQL* http://www.eclipse.org/eclipselink/documentation/2.7/concepts/nosql.htm
  - Hibernate OGM (license: LGPLv2.1) http://hibernate.org/ogm/
  - Kundera (license: Apache License 2.0) https://github.com/Impetus/Kundera
- Non-free software:
  - ObjectDB https://www.objectdb.com/

IDE Support

- **Eclipse**:
  - Dali Java Persistence Tools (license: Eclipse Public License v1.0) https://eclipse.org/webtools/dali/
    - Included in the Eclipse IDE for Java EE Developers.
- **IntelliJ IDEA**:
  - Available only in the Ultimate edition!
    - See: *Enabling JPA Support* https://www.jetbrains.com/help/idea/enabling-jpa-support.html
- **NetBeans**:
  - Jeddict (license: Apache License 2.0) https://jeddict.github.io/ https://github.com/jeddict/jeddict http://plugins.netbeans.org/plugin/53057/jpa-modeler

Hibernate Components

- Hibernate OGM http://hibernate.org/ogm/
  - Provides JPA support for NoSQL databases (e.g., MongoDB, Neo4j)
- Hibernate ORM http://hibernate.org/orm/
  - JPA implementation
- Hibernate Search http://hibernate.org/search/
  - Provides full-text search suppor
- Hibernate Validator http://hibernate.org/validator/
  - An annotation-based solution that allows to express and validate constraints on objects
- Hibernate Tools http://hibernate.org/tools/
  - Developer tools (Eclipse plugins)
- …

Supported Database Management Systems

- See the `org.hibernate.dialect` package.
  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, …

Availability

- Available in the Maven central repository:
  - Artifacts:
    - `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 Support

- hibernate-tools-maven-plugin (license: LGPLv2.1)
  https://github.com/hibernate/hibernate-tools/tree/master/maven
  - Available in the Maven central repository: `org.hibernate:hibernate-tools-maven-plugin`
    https://javalibs.com/artifact/org.hibernate/hibernate-tools-maven-plugin
  - Plugin goals:
    - `hbm2ddl`:
    - `hbm2java`:

NHibernate

- NHibernate (written in: C#, license: LGPLv2.1) https://nhibernate.info/
  - A port of Hibernate to the .NET platform.

Entity

- An entity is a lightweight persistent domain object.

Entity Classes (1)

- An entity class must be annotated with the `javax.persistence.Entity` annotation or denoted in the XML descriptor as an entity.
  - It must have a no-argument constructor that must be `public` or `protected`.
  - It must be a top-level class, an `enum` or interface must not be designated as an entity.
  - It must not be `final`. No methods or persistent instance variables of an entity class may be `final`.

Entity Classes (2)

- Both abstract and concrete classes can be entities.
- Entities may extend non-entity classes as well as entity classes, and non-entity classes may extend entity classes.

Fields and Properties (1)

- The persistent state of an entity is represented by instance variables, which may correspond to JavaBeans properties.
- The state of the entity is available to clients only through the entity's methods, i.e., accessor methods (getter/setter methods) or other business methods.
- The persistent state of an entity is accessed by the persistence provider runtime either via JavaBeans style property accessors ("property access") or via instance variables ("field access").
  - The instance variables of a class must be `private`, `protected`, or `package` visibility independent of whether field access or property access is used.
  - When property access is used, the property accessor methods must be `public` or `protected`.

Fields and Properties (2)

- Collection-valued persistent fields and properties must be of type `java.util.Collection`, `java.util.List`, `java.util.Map` or `java.util.Set`.
  - Any collection implementation type may be used by the application to initialize fields or properties before the entity is made persistent.

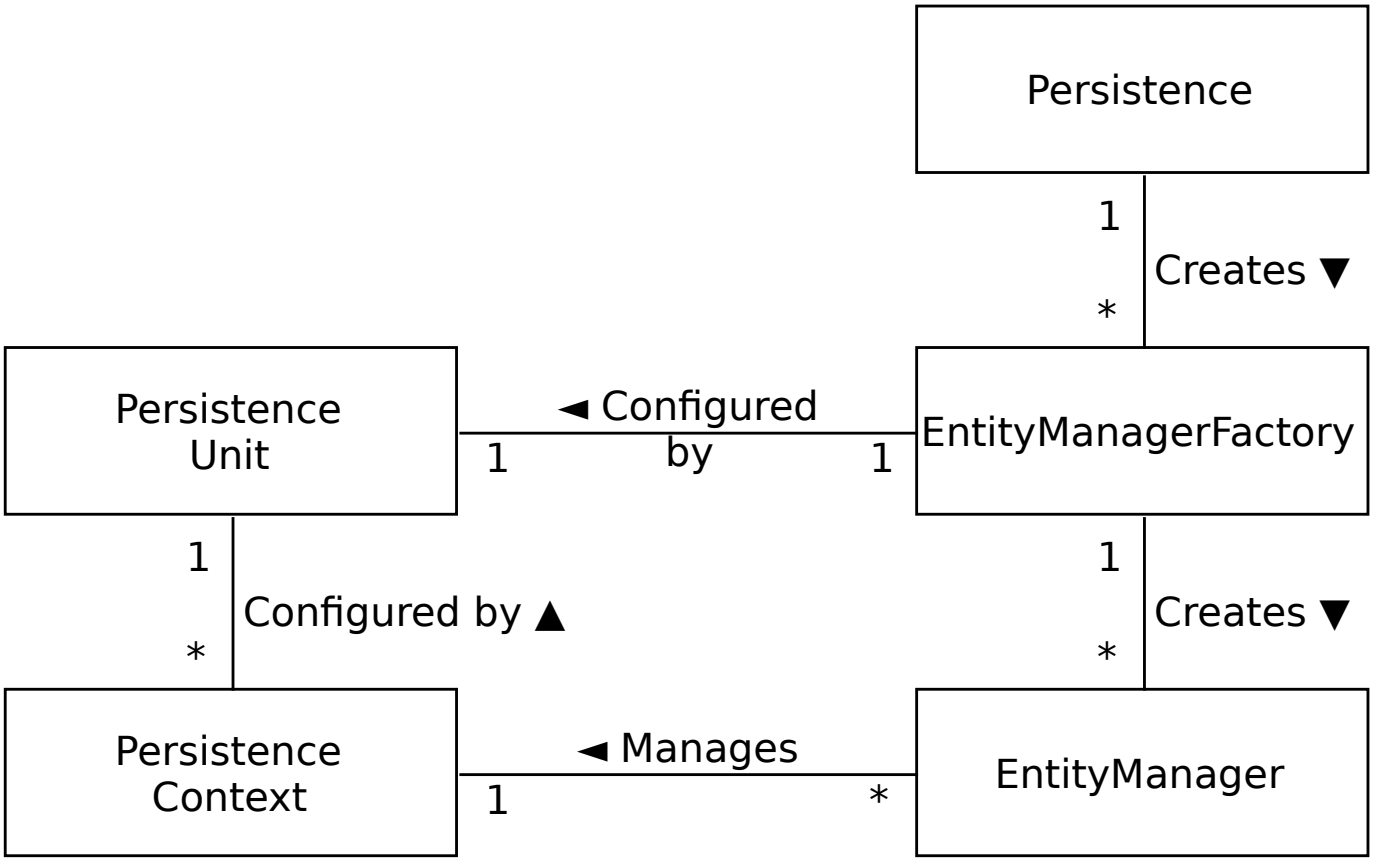
Fields and Properties (3)

- The persistent fields or properties of an entity may be of the following types:
  - Any primitive types
  - `java.lang.String`
  - Other types that implement the `java.io.Serializable` interface (primitive wrapper types, `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`)
  - User-defined types that implement the `java.io.Serializable` interface
  - Enums
  - Entity types
  - Collections of entity types
  - Embeddable classes
  - Collections of basic and embeddable types

Primary Keys and Entity Identity

- Every entity must have a primary key.
- A primary key corresponds to one or more fields or properties of the entity class.
  - A primary key is either simple or compound.
- A simple primary key or a field or property of a composite primary key should be one of the following types:
  - Any Java primitive type, any primitive wrapper type, `java.lang.String`, `java.util.Date`, `java.sql.Date`, `java.math.BigDecimal`, `java.math.BigInteger`.

JPA Concepts and Their Relationship



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

EntityManager (1)

- The `javax.persistence.EntityManager` interface provides an API for persistence operations. https://javaee.github.io/javaee-spec/javadocs/javax/persistence/EntityManager.html
  - See, for example, the `find()`, `persist()`, and `remove()` methods.

EntityManager (2)

- When an `EntityManager` obtains a reference to an entity object, that object is said to be **managed by** the `EntityManager`.
  - The `EntityManager` obtains a reference either by having it explicitly passed as an argument to a method call or because it was read from the database.
- Source: Mike Keith, Merrick Schincariol. *Pro JPA 2*. 2nd edition. Apress, 2013.

Persistence Context

- The set of entity instances managed by an EntityManager is called its **persistence context**.
  - Within a persistence context, each entity instance must have a unique persistent identity.
- The set of entity classes used by an `EntityManager` is defined by a persistence unit.

Persistence Unit

- A persistence unit defines the set of all classes that are related or grouped by the application, and which must be mapped to a single database.

EntityManagerFactory

- The `javax.persistence.EntityManagerFactory` interface is provided for obtaining `EntityManager` objects for a named persistence unit. https://javaee.github.io/javaee-spec/javadocs/javax/persistence/EntityManagerFactory.html
  - Java SE applications must use the `createEntityManagerFactory()` method of the `javax.persistence.Persistence` class to obtain an `EntityManagerFactory` instance. https://javaee.github.io/javaee-spec/javadocs/javax/persistence/Persistence.html

Persistence Unit (1)

- A persistence unit is a logical grouping that includes:
  - An `EntityManagerFactory` and its `EntityManagers`, together with their configuration information.
  - The set of managed classes included in the persistence unit and managed by the `EntityManagers` of the `EntityManagerFactory`.
  - Mapping metadata (in the form of metadata annotations and/or XML metadata) that specifies the mapping of the classes to the database.
- It is defined by a `persistence.xml` file.

Persistence Unit (2)

- Every persistence unit must have a name.
  - Names of persistence units must be unique within their scope.

persistence.xml (1)

- An XML document that defines one ore more persistence unit.
  - XML schema: http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd
- The file must be placed in the `META-INF` directory of a JAR file or a directory.
  - The JAR file or directory whose `META-INF` directory contains the `persistence.xml` file is termed the **root** of the persistence unit.

persistence.xml (2)

- Document structure:

```
<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">
  <!-- One or more persistence-unit elements: -->
  <persistence-unit name="név">
    <!-- ... -->
  </persistence-unit>
  <!-- ... -->
</persistence>
```

persistence.xml (3)

- Example:

```
<?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 Descriptor (1)

- An XML document that contains ORM metadata and serves as both an alternative to and an overriding mechanism for Java language metadata annotations.
  - When the `persistence-unit-metadata/xml-mapping-metadata-complete` elem is present, then the persistence annotations must be completely ignored.
    - Otherwise, the document overrides/supplements the metadata provided by annotations.
- XML schema: http://xmlns.jcp.org/xml/ns/persistence/orm_2_2.xsd

XML ORM Descriptor (2)

- Document structure:

```
<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 “Hello, world!” 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 “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)

- `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 “Hello, world!” 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 “Hello, world!” Program (5)

- Update in the database:

```
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();
```

The Previous 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 Previous Example Without Annotations (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>
```

The Previous Example Without Annotations (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>
```

A More Complex Example (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; }

}
```

A More Complex Example (2)

- `EmployeeService.java`:

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

public class EmployeeService {

  protected EntityManager em;

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

  // creating an employee and persisting it to the database
  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` (continued):

```
  // removing an employee from the database
  public void removeEmployee(int id) {
    Employee emp = findEmployee(id);
    if (emp != null) {
      em.remove(emp);
    }
  }

  // updating an employee in the database
  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` (continued):

```
  // finding an employee in the database
  public Employee findEmployee(int id) {
    return em.find(Employee.class, id);
  }

  // finding all employees in the database
  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);

    // creating and persisting 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` (continued):

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

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

A More Complex Example (7)

- `EmployeeTest.java` (continued):

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

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

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

}
```

Support for Java 8 Date/Time Types (1)

- Starting with version 5.2.0, Hibernate ORM supports Java 8 date/time types by default. Previously, Java 8 support required the `hibernate-java8` module to be included in the classpath during runtime.
  - See: *Hibernate ORM 5.2 release*. Jun 1, 2016. http://in.relation.to/2016/06/01/hibernate-orm-520-final-release/
  - JPA 2.2 supports the use of the classes in the `java.time` package.
    - Although Hibernate also supports the `java.time.Duration`, `java.time.Instant`, and `java.time.ZonedDateTime` types, these types are not supported in JPA 2.2!

Support for Java 8 Date/Time Types (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; // there is nothing to be done

  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; }
}
```

Overriding Default Table Name

- `Employee.java`:

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

   // ...

}
```

Overriding Default Column Names

- `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;
  // ...
}
```

Lazy Loading

- `Employee.java`:

```
import javax.persistence.*;

@Entity
public class Employee {

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

Enumerated Types (1)

- Mapping an `enum` to an integer (`FULL_TIME_EMPLOYEE`: 1, `PART_TIME_EMPLOYEE`: 2, `CONTRACT_EMPLOYEE`: 3):
  - Changes to the `enum` can corrupt the numbering!

```
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
}
```

Enumerated Types (2)

- Mapping an `enum` to a string:

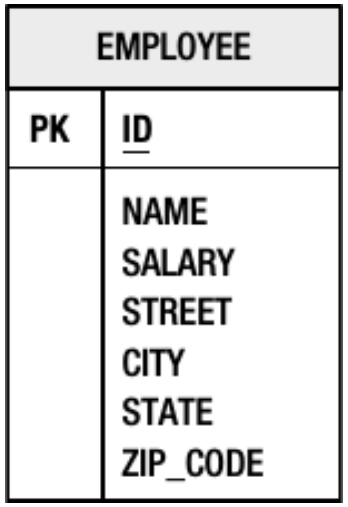
```
import javax.persistence.*;

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;
  // ...
}
```

Embedded Objects (1)

- Instances of embeddable classes do not have persistent identity of their own. Instead, they exist only as part of the state of an entity to which they belong.

Embedded Objects (2)

- Example:



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

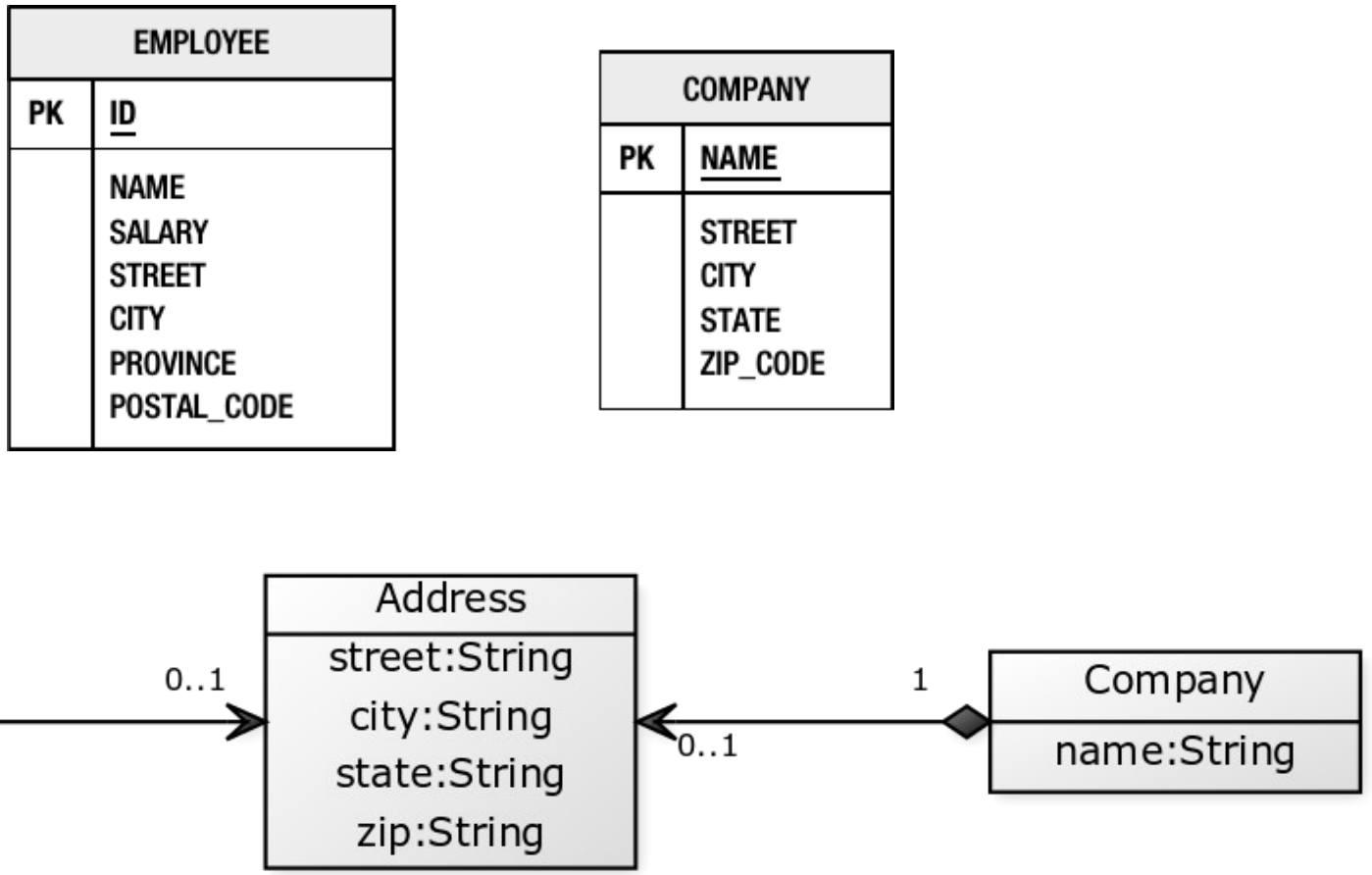
Embedded Objects (3)

- Example (continued):

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

Embedded Objects (4)

- Example:



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

Embedded Objects (5)

- Example (continued):

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

Embedded Objects (6)

- Example (continued): Starting with JPA 2.2, the `@AttributeOverride` annotation is marked as repeatable (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;
  // ...
}
```

Constraints (1)

- Constraints of the `javax.validation.constraints` package are available to be used.
  - Bean Validation https://beanvalidation.org/
    - It is part of Java EE (see the `javax.validation` package and its sub-packages).
      - *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/
    - Reference implementation: Hibernate Validator http://hibernate.org/validator/

Constraints (2)

- Requires an implementation of the Unified Expression Language (EL) that allows the use of dynamic expressions in constraint violation messages.
  - *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/
- In Java EE containers, EL is available by default.

Constraints (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>
```

Constraints (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;
  // ...
}
```

Constraints (5)

- `Employee.java`: using message parameters and message expressions

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

Constraints (6)

- Validation can be accomplished:
  - Automatically before performing database operations.
    - The default behavior is to perform automatic validation only before `INSERT` and `UPDATE` operations.
  - Calling the `validate()` method of a `javax.validation.Validator` object.
    - The object to be validated must be passed as an argument to the method.

Constraints (7)

- Enabling automatic validation in the `persistence.xml` file:

```
<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>
```

Constraints (8)

- The values of the `validation-mode` element:
  - **`AUTO`** (default): If a Bean Validation provider is present in the environment, the persistence provider must perform the automatic validation of entities. If no Bean Validation provider is present in the environment, no automatic validation takes place.
  - **`CALLBACK`**: The persistence provider must perform automatic validation. It is an error if there is no Bean Validation provider present in the environment.
  - **`NONE`**: The persistence provider must not perform automatic validation.

Constraints (9)

- The persistence provider must throw an `javax.validation.ConstraintViolation`nException when constraint violations are detected during the validation of an object.
- Hibernate uses the constraints for generating the database schema.

EntityManager Operations (1)

- `void persist(Object entity)`:
  - Makes an instance managed and persistent.
    - If the entity is already managed by the persistence context, it is ignored.
    - An `EntityExistsException` is thrown if the entity already exists in the database.

EntityManager Operations (2)

- `<T> T find(Class<T> entityClass, Object primaryKey)`:
  - Searches for an entity of the specified class and primary key.
    - The method returns the found entity instance, or `null` if the entity does not exist.

EntityManager Operations (3)

- `void remove(Object entity)`:
  - Removes the entity instance from the database.
    - Only a managed instance can be removed, otherwise the method throws an `IllegalArgumentException`.

EntityManager Operations (4)

- `void detach(Object entity)`:
  - Removes the given entity from the persistence context, causing it to become **detached**.
    - Changes made to entity (including removal of the entity) that have not been flushed to the database will not be persisted.

EntityManager Operations (5)

- `<T> T merge(T entity)`:
  - Merges the state of the given (detached) entity into the current persistence context. It is the inverse of the `detach()` operation.
    - The argument to `merge()` remains detached.
    - The method returns a managed instance.
      - The return value is a new managed instance or a managed instance that already exists in the persistence context.

EntityManager Operations (6)

- `void clear()`:
  - Clears the persistence context, causing all managed entities to become detached.
  - Changes made to entities that have not been flushed to the database will not be persisted.
- `void flush()`:
  - Synchronizes the persistence context to the underlying database.

Cascading Persistence Operations (1)

- By default, every entity manager operation applies only to the entity provided as an argument to the operation.
  - Thus, the operation will not be performed on other entities that have a relationship with the entity that is being passed to the operation.
  - This default behavior can be overridden by the `cascade` element of annotations defining logical relationships (`@ManyToMany`, `@ManyToOne`, `@OneToMany`, `@OneToOne`).

Cascading Persistence Operations (2)

- Cascadable operations are represented by the constants of the `javax.persistence.CascadeType` enum:
  - `ALL`
  - `DETACH`
  - `MERGE`
  - `PERSIST`
  - `REFRESH`
  - `REMOVE`

Cascading Persistence Operations (3)

- Example:

```
import javax.persistence.*;

@Entity
public class Employee {

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

Cascading Persistence Operations (4)

- Cascading can be enabled globally only for the `persist()` operation in the XML descriptor:

```
<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>
```

Detached Entities (1)

- A detached entity instance is an instance with a persistent identity that is not (or no longer) associated with a persistence context.
  - Any changes made on the entity won't be persisted to the database.
- An entity may become detached in many ways:
  - When a persistence context is closed, all its managed entities become detached.
  - The `clear()` method of the `EntityManager` interface detaches all managed entities of the persistence context.
  - The `detach()` method of the `EntityManager` interface detaches the the entity provided as an argument.
  - …

Detached Entities (2)

- Working with a detached entity (does not work as expected):
  - When the transaction is committed, the entity is not updated in the database.

```
EntityManager em;

Employee emp; // reference to a detached entity

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

Detached Entities (3)

- Working with a detached entity:
  - When the transaction is committed, the entity is updated in the database.

```
EntityManager em;

Employee emp; // reference to a detached entity

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

Synchronization to the Database

- When a persistence provider generates SQL statements and executes them on the database, the persistence context is said to be being **flushed**.
  - The persistence context can be flushed at any time when the persistence provider deems it necessary.
  - The persistence context is guaranteed to be flushed in the following two situations:
    - When a transaction is committed.
    - When the `flush()` method of the `EntityManager` is invoked.

Primary Key Generation (1)

- The `javax.persistence.GeneratedValue` annotation is provided to specify a generation strategy for the values of primary keys.
  - A persistence provider is only required to support the use of the annotation for simple primary keys.
  - The generated key may not be available until the entity has been inserted in the database.

Primary Key Generation (2)

- Four primary key generation strategies are available that are represented by the constants of the `javax.persistence.GenerationType` enum:
  - **`AUTO`**: the persistence provider should pick an appropriate strategy for the particular database.
  - **`IDENTITY`**: the persistence provider must assign primary keys for entities using an identity column.
  - **`SEQUENCE`**: the persistence provider must assign primary keys for entities using a database sequence.
  - **`TABLE`**: the persistence provider must assign primary keys for entities using an underlying database table.
    - This is the most flexible and portable strategy.

Primary Key Generation (3)

- Example:

```
import javax.persistence.*;

@Entity
public class Employee {

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

Primary Key Generation (4)

- Example (continued):
  - The DDL statement that creates the `Employee` database table (H2):

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

Queries (1)

- Java Persistence Query Language (JPQL):
  - The Java Persistence Query Language is a platform-independent object-oriented query language defined as part of JPA. It enables to to define queries over the persistent entities stored in a relational database.
  - JPQL can be compiled to a target language, such as SQL.
  - The following three statements are supported: SELECT, UPDATE, DELETE.

Queries (2)

- Static and dynamic queries:
  - Static queries are defined in annotations or in XML descriptors.
    - They have a name, thus, they are also called as named queries.
  - Dynamic queries are provided at runtime.

Queries (3)

- Queries are represented by the `javax.persistence.Query` and `javax.persistence.TypedQuery<`*X*`>` interfaces.
  - Queries can be created by invoking the `createNamedQuery()`, `createNativeQuery()`, or `createQuery()` method of an `EntityManager`.

Queries (4)

- Examples for SELECT queries:

```
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')
```

Queries (5)

- Examples for SELECT queries (continued):

```
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
```

Queries (6)

- An example for execution a dynamic query:

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

Queries (7)

- An example for using a dynamic query:

```
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();
  }

  // ...
}
```

Queries (8)

- Example for using a static query:

```
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 {

  // ...
}
```

Queries (9)

- An example for using a static query (continued):

```
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();
  }

}
```

Queries (10)

- Criteria API:
  - The Criteria API provides a means to create queries programmatically.
  - It is contained in the `javax.persistence.criteria` package.

Queries (11)

- Example for using the Criteria API:
  - Executing the `SELECT e FROM Employee e WHERE e.name = 'John Smith'` query:

```
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();
```

Relationships (1)

- A relationship is an association between two entities.
  - An entity may participate in many relationships of different types.
- Relationship attributes:
  - Directionality: unidirectional, bidirectional
    - Every bidirectional relationship can be viewed as a pair of unidirectional relationships.
  - Cardinality: `0..1`, `1`, `*`

Relationships (2)

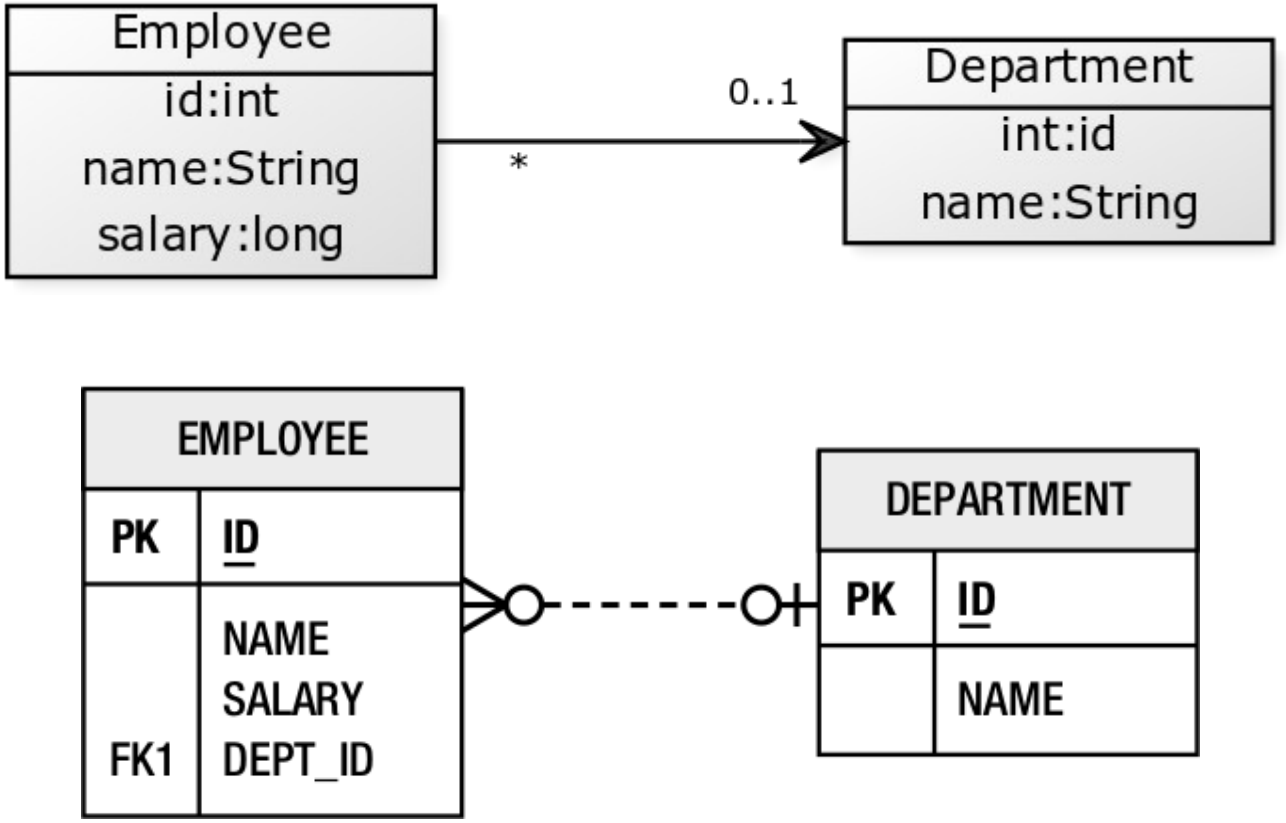
- The following types of relationships are available:
  - Many-to-one
  - One-to-one
  - One-to-many
  - Many-to-many

Relationships (3)

- Single-valued associations:
  - Many-to-one
  - One-to-one
    - Bidirectional one-to-one relationship is discussed as a special case.
- Collection-valued association:
  - One-to-many
    - Unidirectional one-to-many relationship is discussed as a special case.
  - Many-to-many

Many-to-One Relationship (1)

- Example:



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

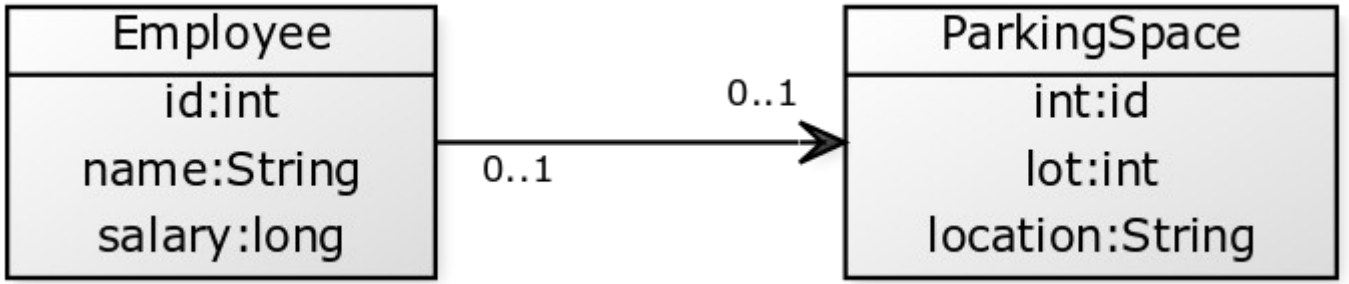
Many-to-One Relationship (2)

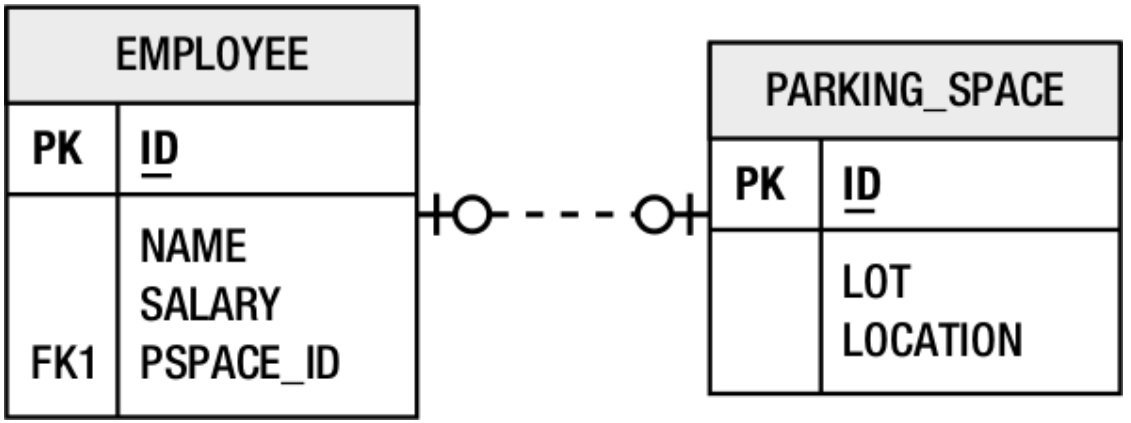
- Example (continued):

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;

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

  // ...
}
```

One-to-one Relationship (1)

- Example:





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

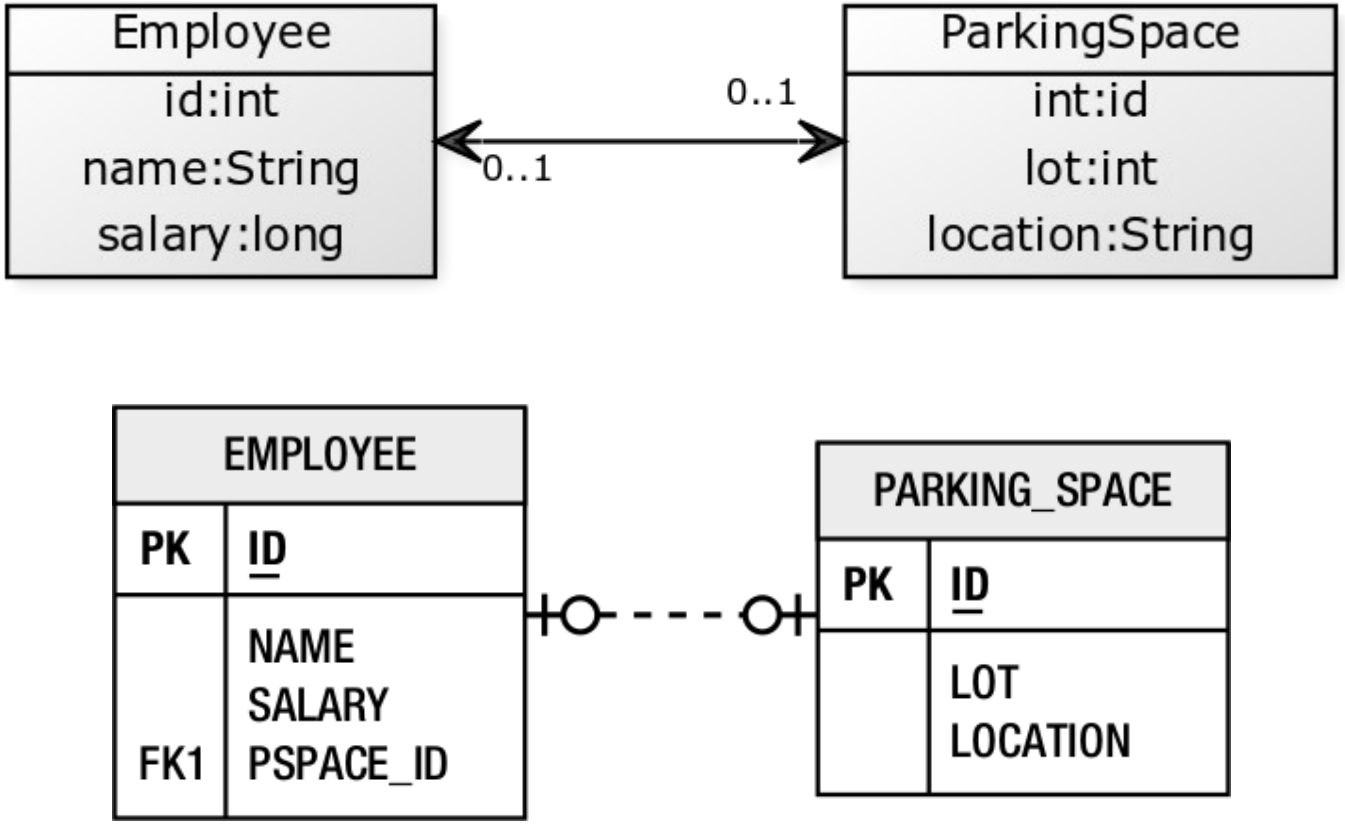
One-to-one Relationship (2)

- Example (continued):

```
import javax.persistence.*;

@Entity
public class Employee {

  @Id
  private int id;

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

  // ...
}
```

Bidirectional One-to-One Relationship (1)

- Example:



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

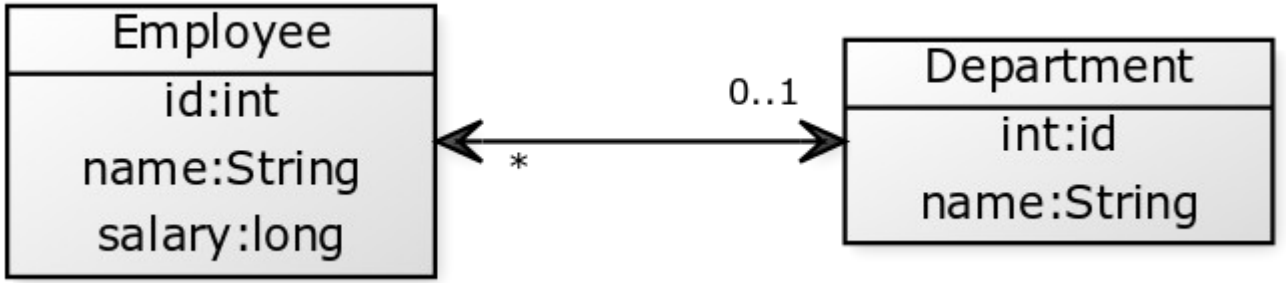
Bidirectional One-to-One Relationship (2)

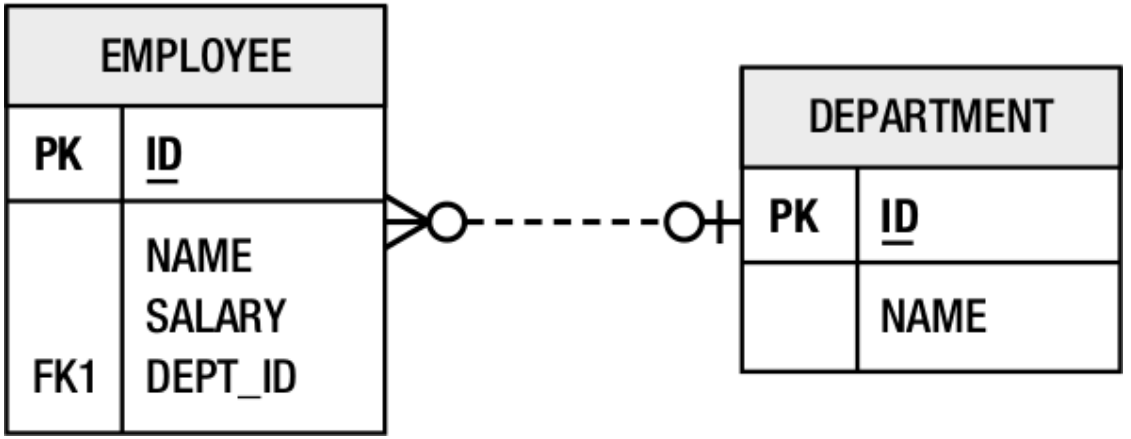
- Example (continued):

```
import javax.persistence.*;

@Entity
public class ParkingSpace {

  @Id
  private int id;

  private int lot;
  private String location;

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

  // ...
}
```

One-to-Many Relationship (1)

- Example:





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

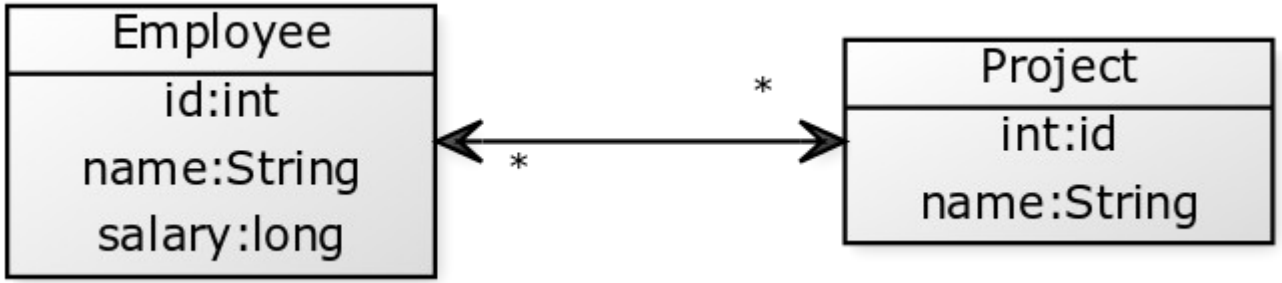
One-to-Many Relationship (2)

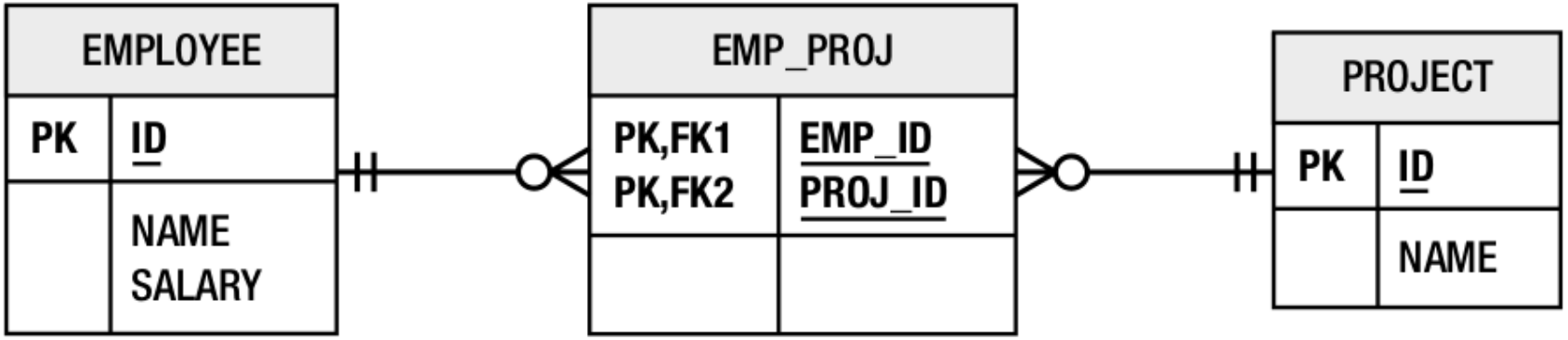
- Example (continued):

```
import javax.persistence.*;

@Entity
public class Department {

  @Id
  private int id;

  private String name;

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

  // ...
}
```

Many-to-Many Relationship (1)

- Example:





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

Many-to-Many Relationship (2)

- Example (continued):

```
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;

  // ...
}
```

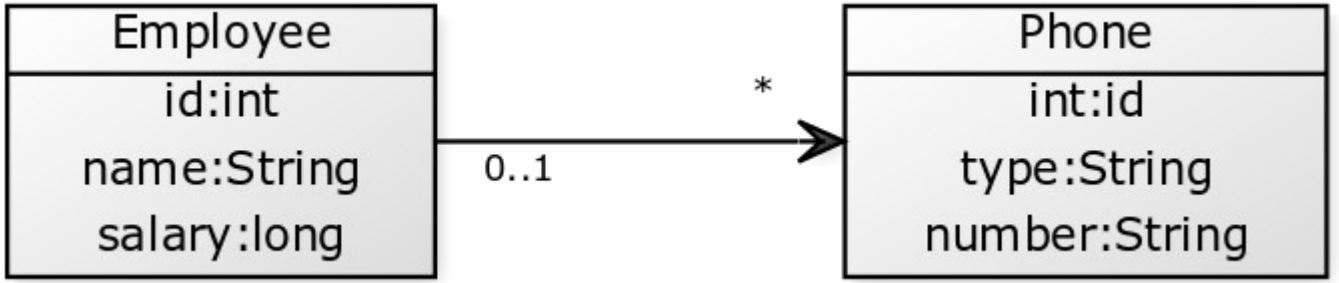
Many-to-Many Relationship (3)

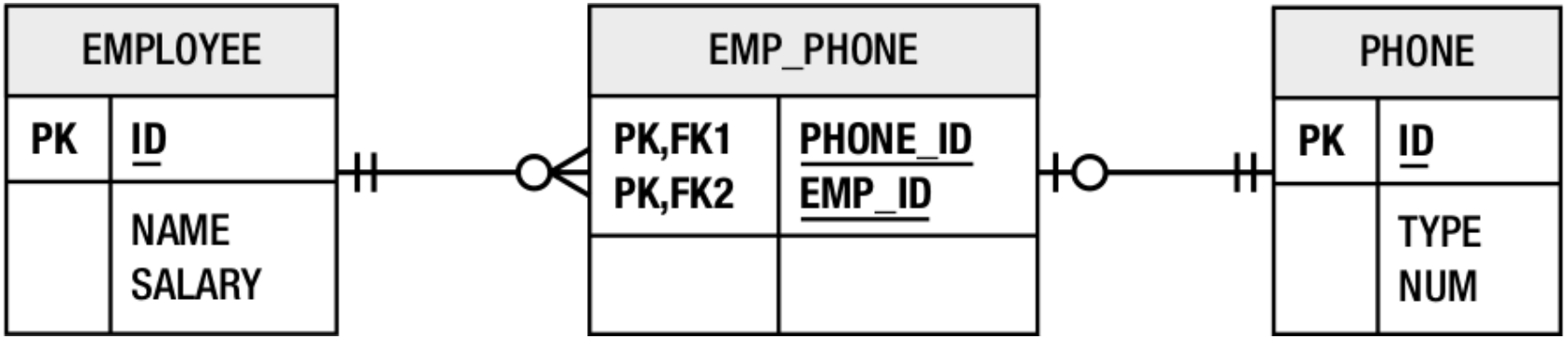
- Example (continued):

```
import javax.persistence.*;

@Entity
public class Project {

  @Id
  private int id;

  private String name;

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

  // ...
}
```

Unidirectional One-to-Many Relationship (1)

- Example:





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

Unidirectional One-to-Many Relationship (2)

- Example:

```
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;

  // ...
}
```

Element Collections (1)

- An element collection is a collection of a basic type or an embeddable type.
  - Basic types: e.g., `java.lang.String`, `java.lang.Integer`, `java.math.BigDecimal`, …
    - See: *Basic Types* http://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#basic
- Element collections are not relationships.
  - Relationships define associations between independent entities, whereas element collections contain objects that are dependent upon the referencing entity, and can be retrieved only through the entity that contains them.

Element Collections (2)

- Example:

```
import javax.persistence.*;

@Embeddable
public class VacationEntry {

  private LocalDate startDate;

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

  // ...
}
```

Element Collections (3)

- Example (continued):

```
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;

  // ...
}
```

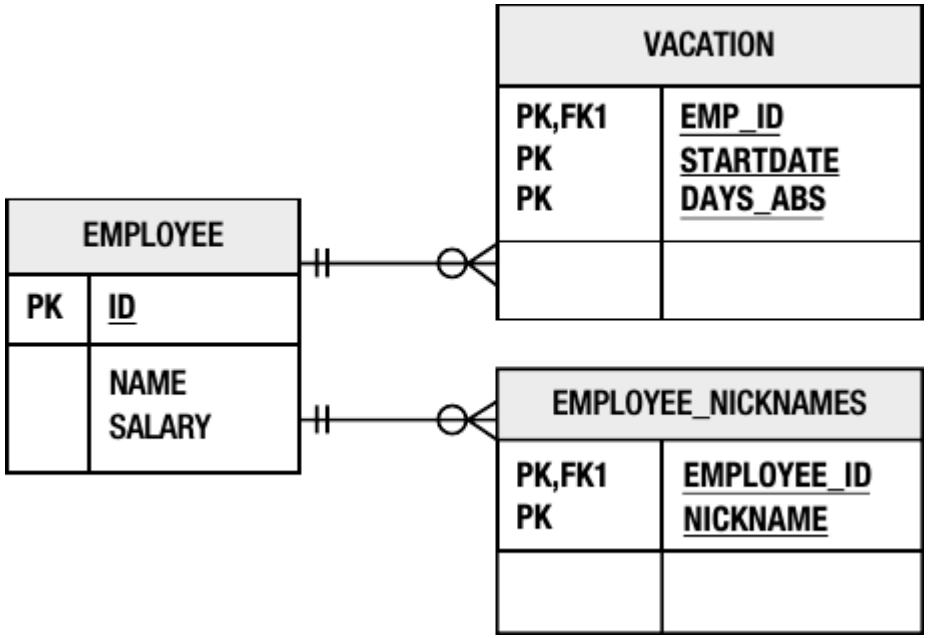
Element Collections (4)

- Example (continued):

```
@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;

  // ...
}
```

Element Collections (5)

- Example (continued):



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

Element Collections (6)

- An example for using a list:

```
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;

  // ...
}
```

Element Collections (7)

- An example for using a list:

```
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;

  // ...
}
```

Logging (1)

- Hibernate uses the **JBoss Logging** framework for logging. https://github.com/jboss-logging/jboss-logging
  - It supports numerous logging frameworks (back-ends), such as the `java.util.logging` package or SLF4J.
- Further information:
  - 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

Logging (2)

- Log messages are grouped into categories, to each of which a log level can be assigned.
  - Practically, each category correspond to a `Logger` object with the same name.
- Log categories of interest:

Category	Description
`org.hibernate`	Contains all log messages
`org.hibernate.SQL`	SQL statements executed
`org.hibernate.type.descriptor.sql`	Values bound to JDBC parameters and extracted from JDBC results
`org.hibernate.tool.hbm2ddl`	SQL DDL statements executed

Logging (3)

- The formatting of SQL messages can be controlled via the `persistence.xml` file as follows:

```
<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>
```

Further Recommended 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*. 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