Medii de proiectare și programare

2023-2024
Curs 9

Conținut

- Object-Relational Mapping (ORM)
  - Hibernate
  - Entity Framework
- Servicii Web

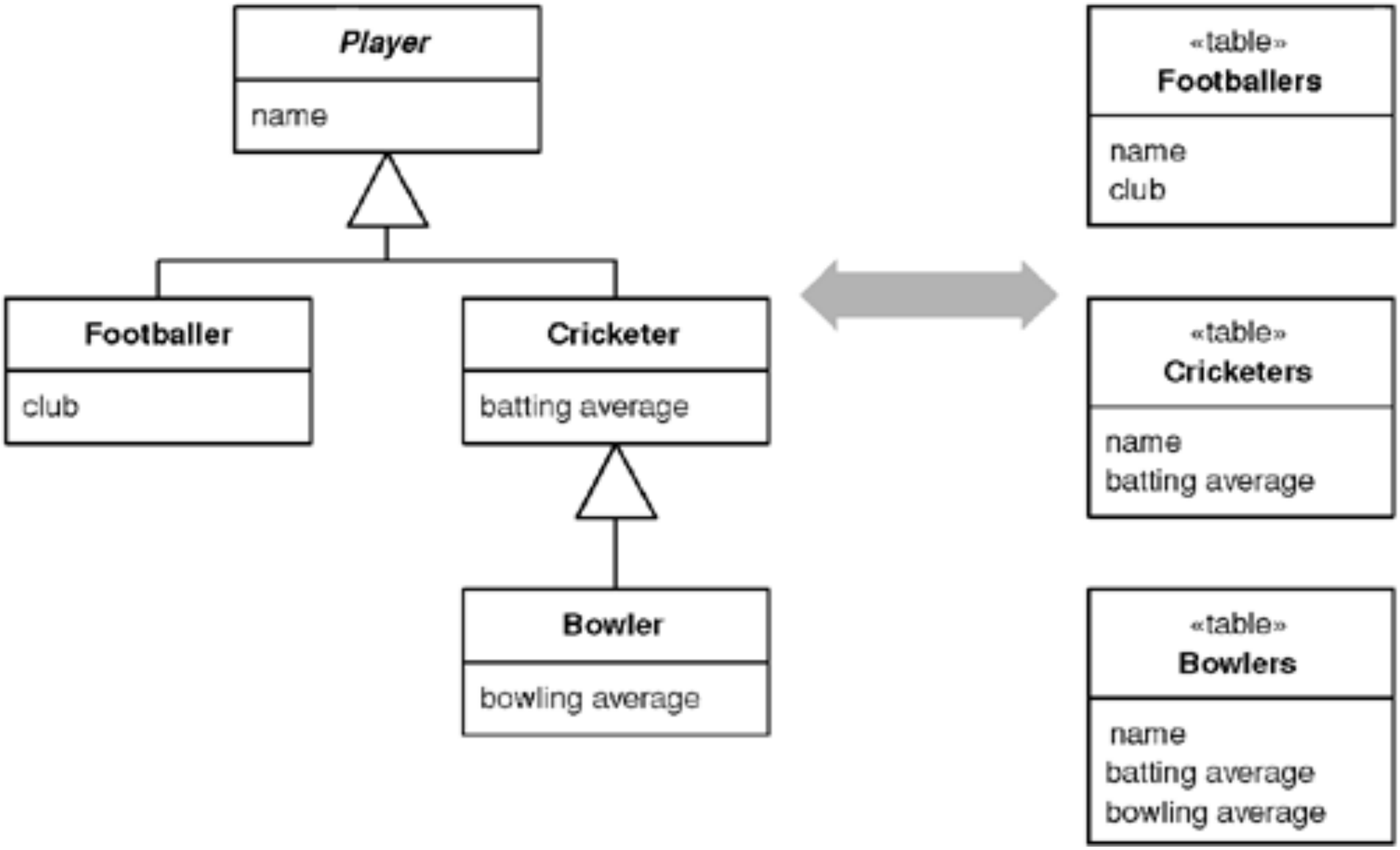
Maparea moștenirii

- Bazele de date relaționale nu suportă moștenirea.
- Programatorul trebuie să mapeze moștenirea dintre entitățile din modelul orientat obiect într-o bază de date relațională.
- Tehnici:
  - Maparea ierarhiei de clase într-o singură tabelă.
  - Maparea fiecărei clase într-o tabelă.
  - Maparea fiecărei clase concrete în tabela ei.
  - Maparea claselor într-o structura de tabele generică.

Tabelă pentru fiecare clasă concretă

- Fiecare clasă concretă (non-abstract) din ierarhie are tabela ei.
- Fiecare tabelă conține coloane pentru toate proprietățile din ierarhie până la ea. ***Atributele din clasa de bază sunt duplicate în tabelele corespunzătoare subclaselor***.
- Este responsabilitatea programatorului de a se asigura că cheile sunt unice nu doar în tabela corespunzătoare clasei dar și între toate tabelele asociate ierarhiei.

Tabelă pentru fiecare clasă concretă



Tabelă pentru fiecare clasă concretă

```
Footballer fb=new Footballer("A A", "ABC")
Cricketer cr=new Cricketer("C C", 23);
Bowler bw=new Bowler("B B",21, 47);
Footballer fb2=new Footballer("D D", "BGD");
Bowler bw2=new Bowler("H H",12, 23);
```

Footballers

PK	Name	Club
1	A A	ABC
4	D D	BGD

Cricketers

PK	Name	BattlingAvg
2	C C	23

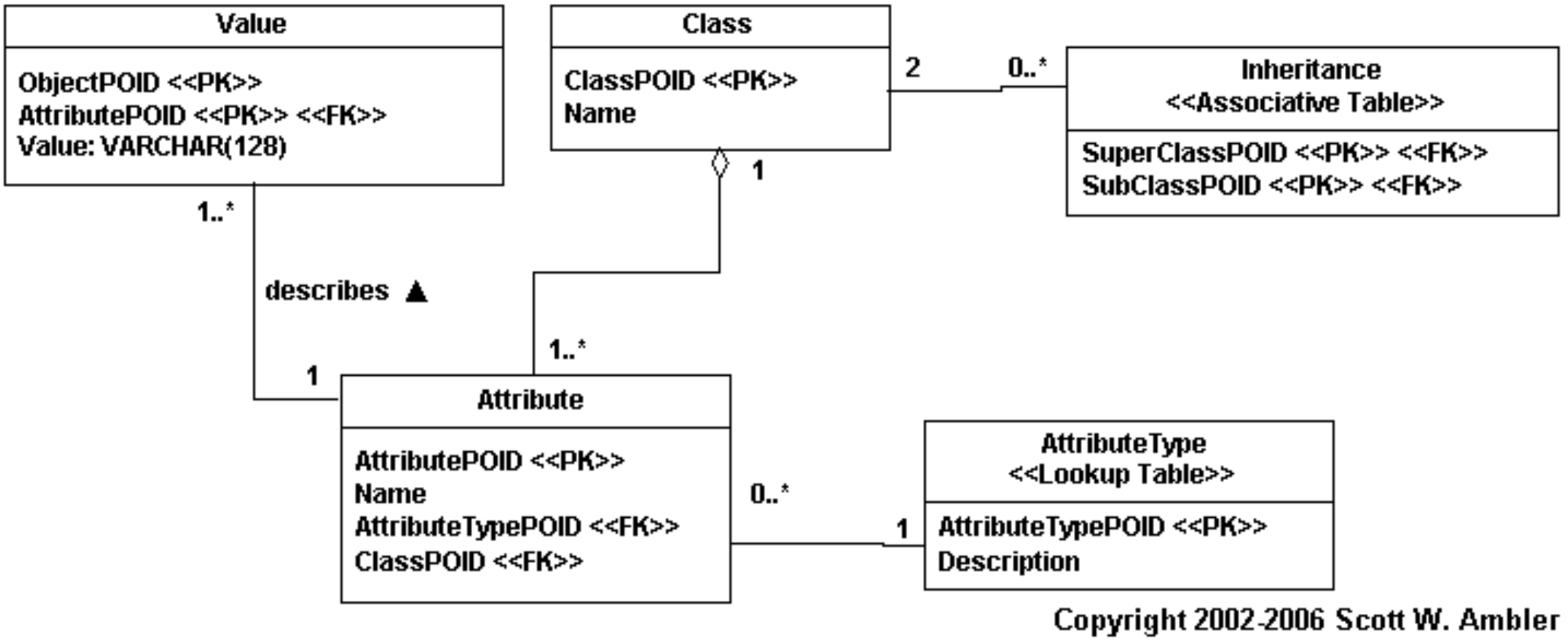
Bowlers

PK	Name	BattlingAvg	BowlingAvg
3	B B	21	47
5	H H	12	23

Tabelă pentru fiecare clasă concretă

- Avantaje:
  - Fiecare tabelă păstrează toate informațiile relevante și nu are câmpuri irelevante. Este ușor de înțeles și de alte aplicații care nu folosesc obiecte.
  - Nu este nevoie de operații join pentru citirea datelor.
  - Fiecare tabelă este accesată doar când clasa respectivă este accesată. Performanța este mai bună.
- Dezavantaje:
  - Gestiunea dificilă a cheilor primare.
  - Nu pot fi constrânse relațiile către clasele abstracte.
  - Dacă câmpurile din modelul obiectual sunt mutate în ierarhie, trebuie modificate definițiile tabelelor.
  - Dacă se modifică un câmp dintr-o clasă de bază, trebuie modificate toate tabelele corespunzătoare subclaselor, pentru că aceste câmpuri sunt duplicate.
  - O operație de căutare folosind clasa de bază, necesită căutari în toate tabelele (accesări multiple ale bazei de date sau o operație de join complicată).

Tabele generice



Tabele generice

- Avantaje:
  - Poate fi extinsă pentru a oferi suport pentru o gamă largă de mapări, inclusiv maparea relațiilor.
  - Este flexibilă, permite modificarea ușoară a modului în care sunt păstrate obiectele (trebuie modificate doar metadatele din tabelele *Class, Inheritance, Attribute* și *AttributeType*).
- Dezavantaje:
  - Este fezabilă doar pentru date de dimensiuni mici, deoarece necesită accesări dese ale bazei de date doar pentru reconstruirea unui singur obiect).
  - Interogările pot fi dificile deoarece necesită accesarea mai multor înregistrări pentru un singur obiect.

Maparea relațiilor dintre obiecte

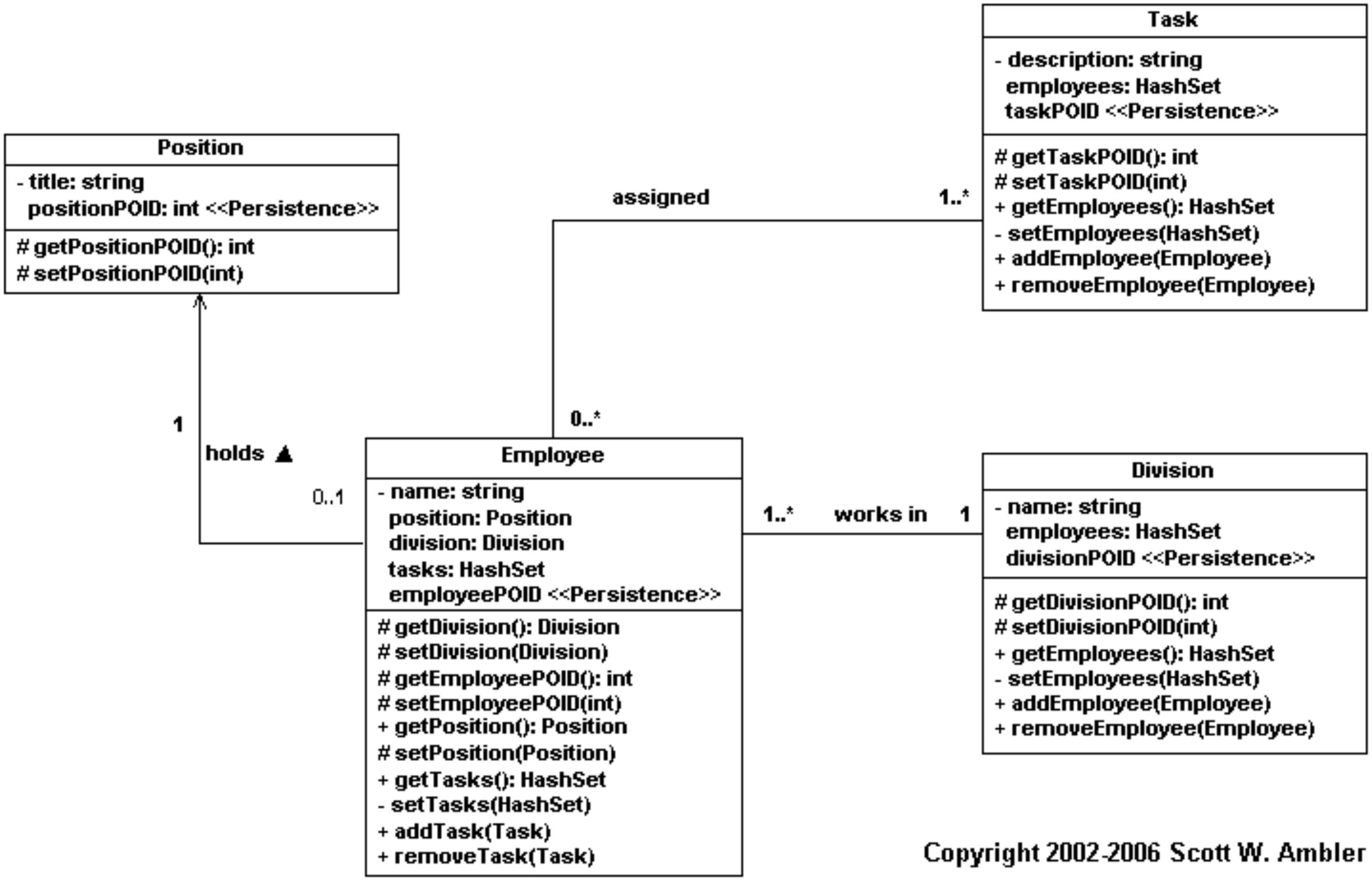
- *Problema*: modul în care paradigma orientată obiect și modelul relațional tratează legăturile dintre obiecte/înregistrari, care cauzează două situații.
  1. *Diferența reprezentării.* Obiectele tratează legăturile prin păstrarea referințelor care există în timpul execuției (referințele sunt temporare). Bazele de date relaționale tratează legăturile prin păstrarea cheilor în tabele (cheile sunt permanente).
  2. *Obiectele pot folosi colecții pentru a gestiona mai multe referințe într-un singur atribut.* Normalizarea obligă ca toate legăturile/valorile să nu fie multiple. Se inversează structura de date dintre obiecte și tabele.
- Exemplu: Un obiect *Order* are o colecție de obiecte de tip *line item* care nu păstrează o referință către obiectul de tip order.
  - ★ Structura tabelei este inversă, înregistrările *line item* includ o cheie străină către înregistrarea *order* corespunzătoare (câmpurile dintr-o înregistrare nu pot fi multivaloare).

Tipuri de relații

- Două categorii de relații între obiecte importante la mapare:
- ***Multiplicitatea***. Include 3 tipuri:
  - Relații *unu-la-unu*. Maximul multiplicității la ambele capete este 1.
  - Relații *unu-la-n (sau n-la-unu)*. Multiplicitatea la unul dintre capete este 1, la celălalt capăt este n.
  - Relații *n-la-n*. Maximul multiplicității la ambele capete este mai mare decât 1.

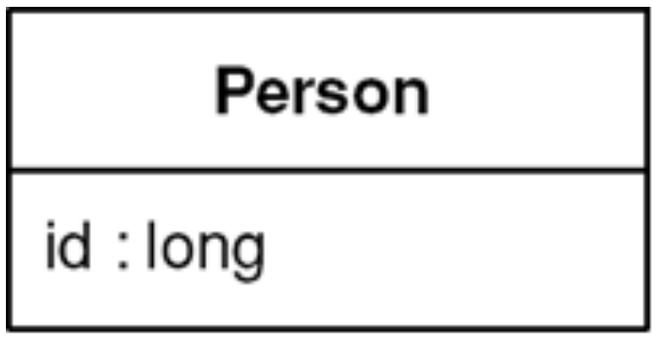
- ***Direcția***. Include 2 tipuri:
  - Relații *unidirecționale*. Un obiect știe de obiectul (obiectele) cu care are o legătură, dar celălalt obiect (celelalte obiecte) nu știe (nu știu) de el.
  - Relații *bidirecționale*. Ambele obiecte aflate într-o relație știu unul de celălalt.

Tipuri de relații



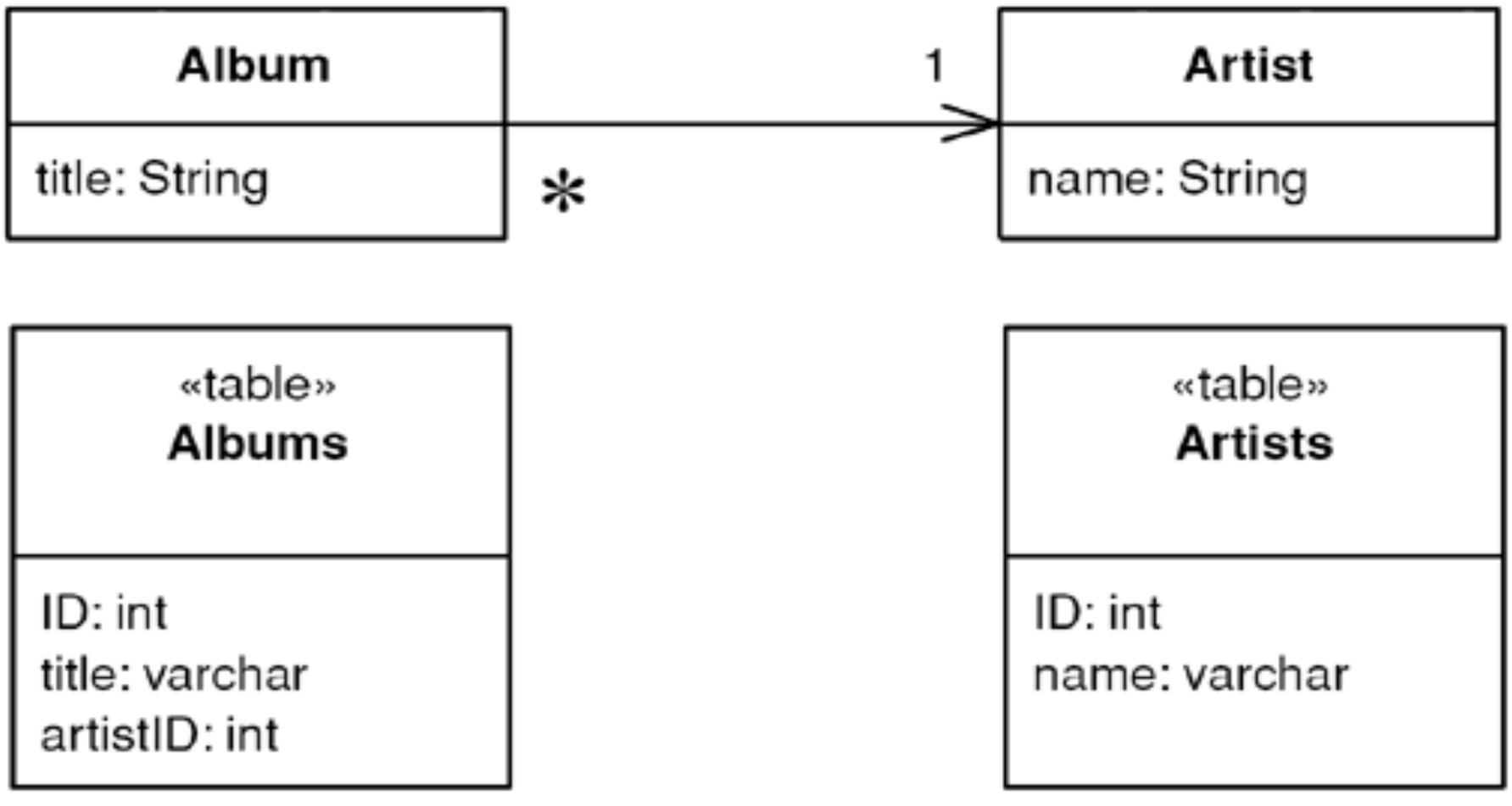
Șablonul *Identity*

- Salvează un identificator (ID) din baza de date într-un obiect pentru a păstra legătura dintre un obiect în memorie și o înregistrare din baza de date.
- Cheia primară dintr-o bază de date relațională este păstrată printre atributele obiectului.
- Șablonul ar trebui folosit când există o mapare între obiectele din memorie și înregistrările din baza de date (cheia primară este diferită de atributele din obiect).



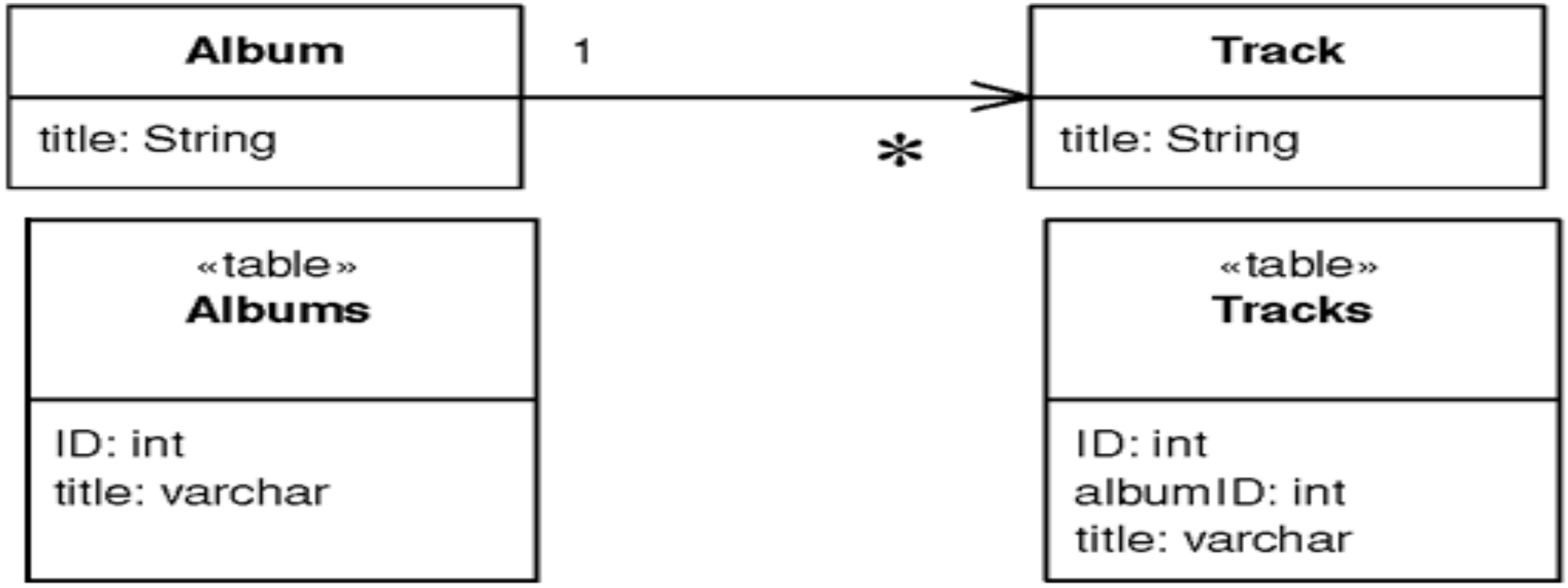
Maparea cheii străine

- Mapează o asociere dintre obiecte ca și cheie străină între tabelele dintr-o bază de date relațională.
- Fiecare obiect conține cheia din tabela corespunzătoare.
- Dacă două obiecte sunt legate printr-o relație de asociere, relația poate fi înlocuită printr-o cheie străină în baza de date.



Maparea cheii străine

- Maparea unei colecții de obiecte.
- Într-o bază de date relaționala nu se poate salva o colecție, trebuie inversată direcția referinței.
- Maparea cheii străine poate fi folosită pentru aproape toate asocierile dintre clase. Nu poate fi folosită pentru asocierea *n-la-n*.

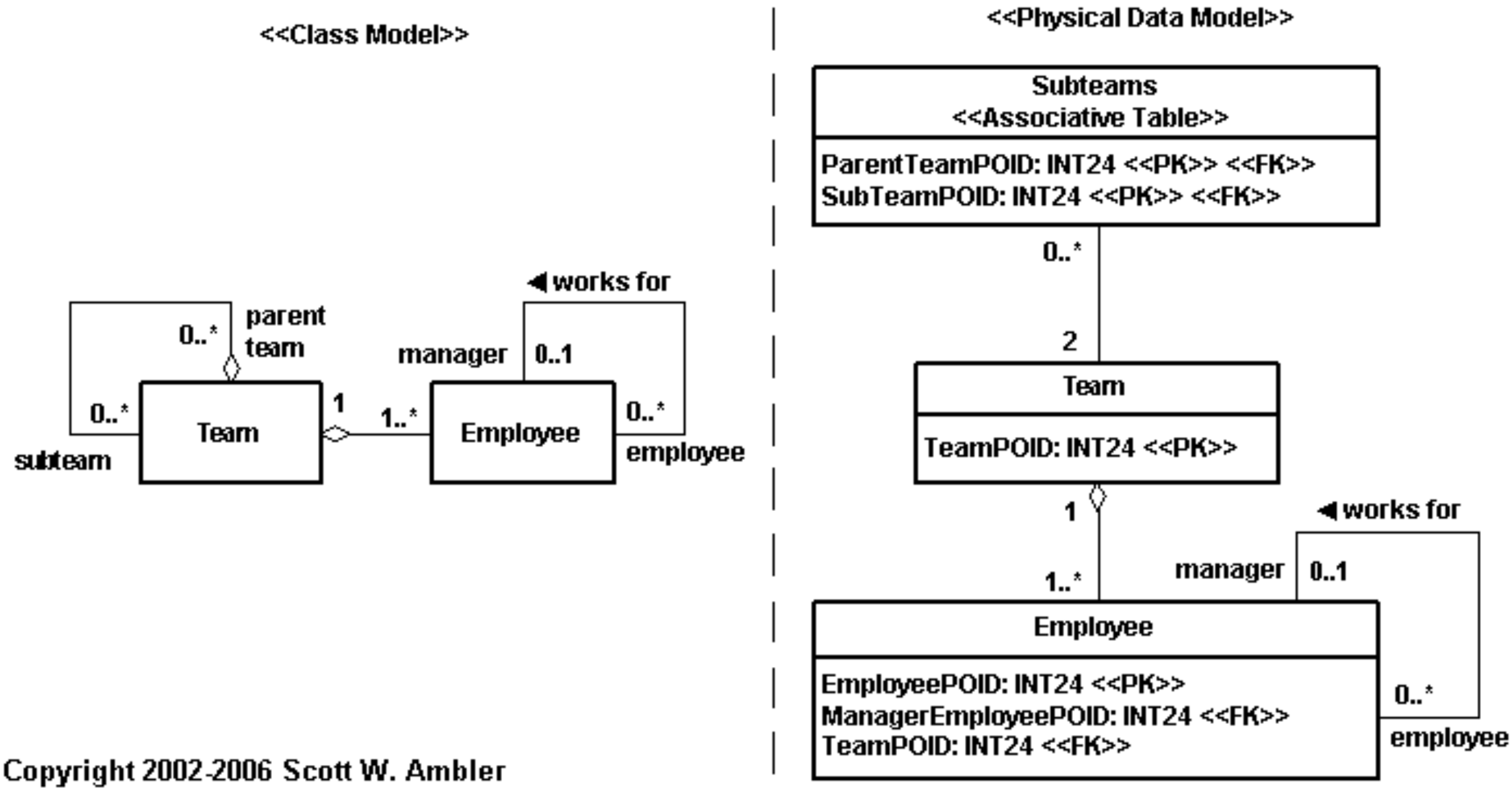


Maparea folosind o tabelă de asociere

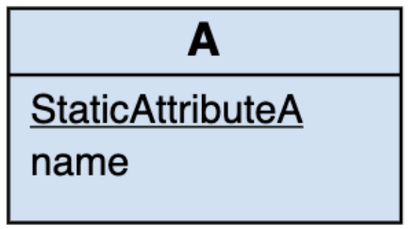
- Salvează o asociere ca o tabelă cu chei străine către tabelele legate prin asociere.
- Obiectele pot păstra mulțimi de valori folosind colecții. Bazele de date relaționale nu au această caracteristică și sunt restrictionate la câmpuri cu o singură valoare.
- Ideea este de a crea o tabelă de legătură/asociere pentru a stoca asocierea.
- Tabela are doar două coloane corespunzătoare cheilor străine, conține câte o înregistrare pentru fiecare pereche de obiecte asociate.
- Tabela de legătura nu are echivalentul unui obiect în memorie (nu are ID). Cheia primară este compusă din cheile primare ale celor două tabele asociate.
- Tabela de asociere este folosită cel mai des pentru maparea asocierii n-la-n, dar poate fi folosită și pentru alte tipuri de asocieri (mai dificil, complex).

Tabela de asociere

Maparea asocierilor recursive



Maparea proprietăților statice



TableA

PK	Name	StaticAttributeA
	AA	1
	BB	2
	CC	1

TableA

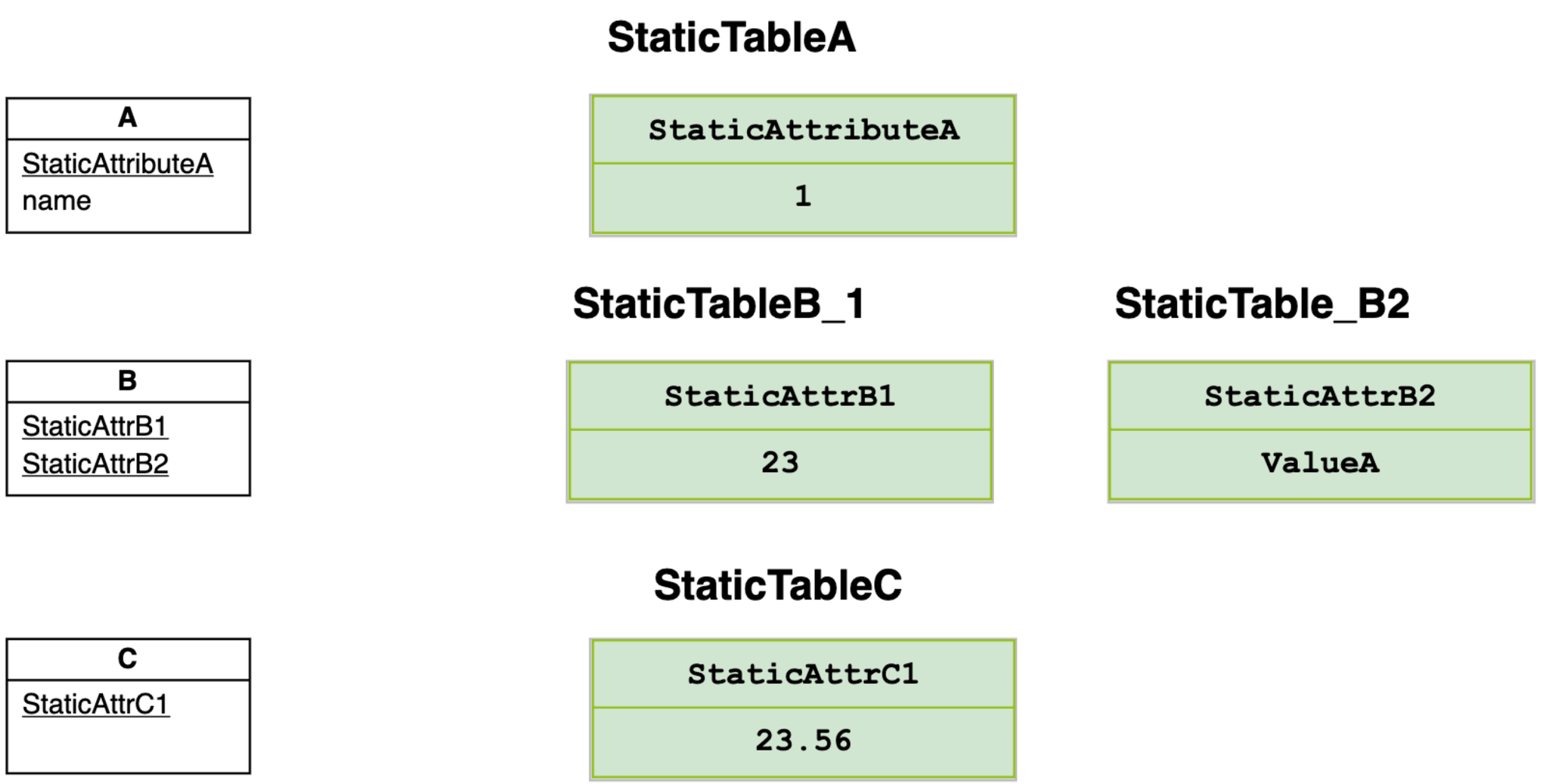
PK	Name
	AA
	BB
	CC

StaticTableA

StaticAttributeA
1

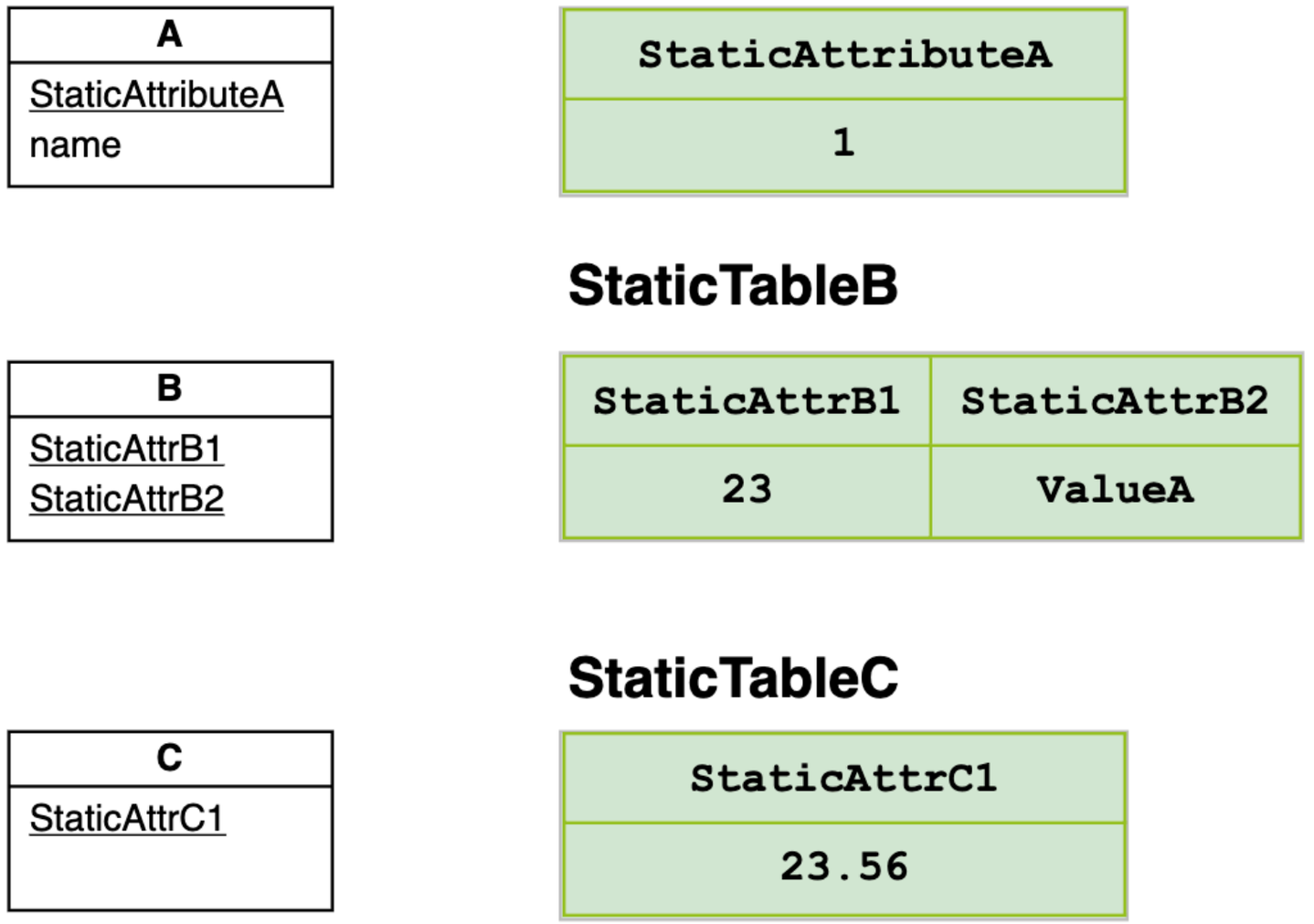
Maparea proprietăților statice - Strategii (1)

- O tabelă cu o singură înregistrare, o singură coloană pentru fiecare proprietate statică:
  - Pro: Simplu, acces rapid
  - Con: multe tabele mici



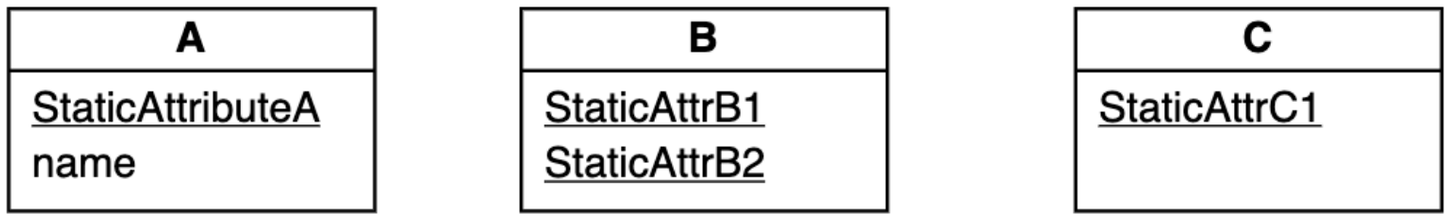
Maparea proprietăților statice - Strategii (2)

- O tabelă cu mai multe coloane, o singură înregistrare pentru fiecare clasă:
  - Pro: Simplu, acces rapid
  - Con: multe tabele mici, dar mai puține decât la strategia precedentă



Maparea proprietăților statice - Strategii (3)

- O singură tabelă cu mai multe coloane - o singură înregistrare pentru toate clasele:
  - Pro: număr minim de tabele introdus
  - Con: potențiale probleme de concurență dacă mai multe clase trebuie să acceseze datele în același timp.



StaticAttributesTable

A.StaticAttributeA	B.StaticAttrB1	B.StaticAttrB2	C.StaticAttrC1
1	23	ValueA	23.56

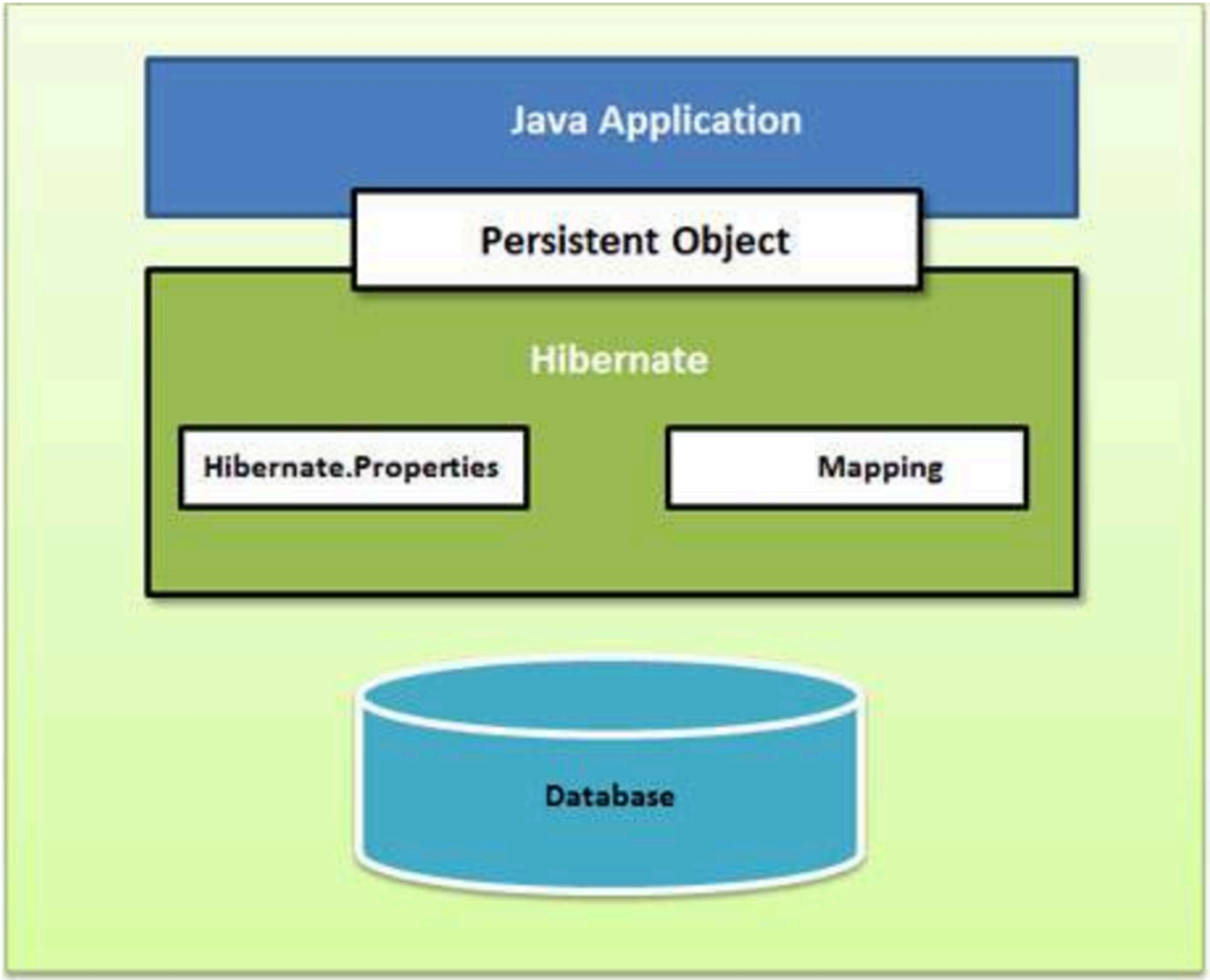
Maparea proprietăților statice

- Strategii (cont):
  - Schemă generică cu mai multe înregistrări pentru toate clasele:
    - Pro: număr minim de tabele introdus. Reduce problemele legate de concurență.
    - Con: Necesitatea convertirii între tipuri de date. Schema este asociată cu numele claselor și a proprietăților statice.

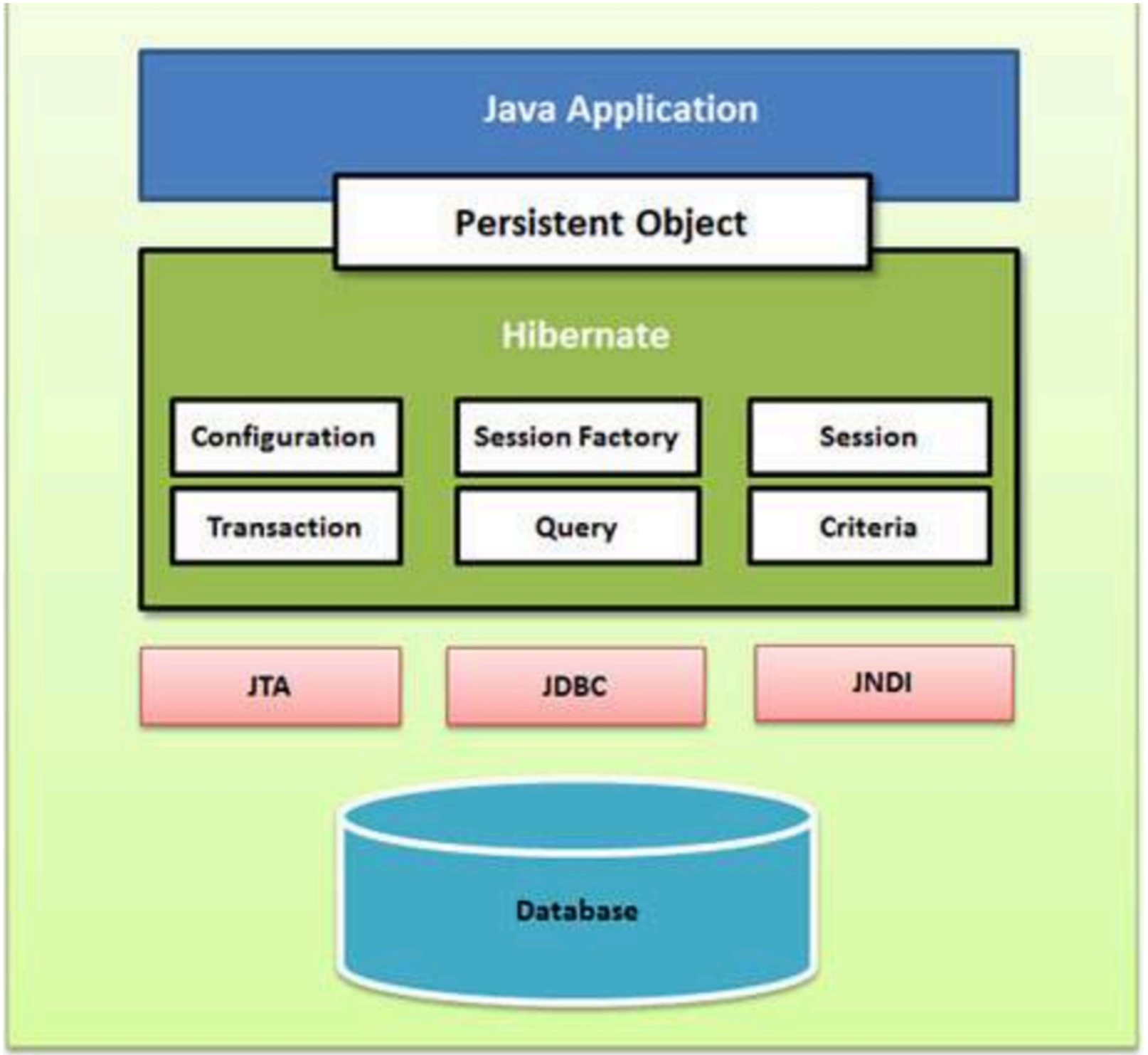
Hibernate

- Instrument open source pentru ORM.
- Aplicațiile care folosesc Hibernate definesc/specifică clasele persistente ce vor fi mapate în tabele într-o bază de date relațională.
- Toate instrucțiunile SQL sunt generate automat în timpul execuției.
- Informațiile despre mapare sunt transmise prin adnotări (vers<6, ca și un document de mapare în format XML).
- Adnotările specifică:
  - cum vor fi mapate proprietățile la coloanele din tabelă (tabele).
  - strategia aleasă de dezvoltator pentru mapare (unde este cazul).

Arhitectura



Arhitectura



Arhitectura

- Interfețele/Clasele definite de Hibernate pot fi clasificate astfel:
  - Interfețe folosite de aplicații pentru a efectua operații CRUD și interogări. Logica aplicației corespunzătoare nivelului de persistență va fi strâns cuplată de acestea: *Session*, *Transaction* și *Query*.
  - Clase apelate de aplicație pentru configurarea instrumentului Hibernate: clasa *Configuration*.
  - Interfețe callback care permit aplicațiilor să reacționeze la evenimente ce apar în timpul execuției instrumentului Hibernate: *Interceptor*, *Lifecycle* și *Validatable*.
  - Interfețe care permit extinderea Hibernate: *UserType*, *CompositeUserType* și *IdentifierGenerator*.
- Hibernate folosește Java API existent: JDBC, Java Transaction API (JTA) și Java Naming and Directory Interface (JNDI).

Interfețe de bază

- Interfața *Session*: este cea mai folosită interfață de către aplicațiile care folosesc Hibernate. O instanță de tip Session nu consumă multe resurse și poate fi ușor creată/distrusă.
- Interfața *SessionFactory*. Aplicațiile obțin instanțe de tip Session folosind un obiect de tip SessionFactory. Păstrează în cache instrucțiunile SQL generate dinamic și alte metadate legate de mapări folosite în timpul execuției.
- Clasa *Configuration*. Un obiect de tip Configuration este folosit pentru configurarea și pornirea Hibernate. Aplicațiile pot folosi acest obiect pentru a specifica locația bazei de date și a altor proprietăți specifice Hibernate.
- Interfața *Transaction*. Abstractizează codul corespunzător unei tranzacții de implementarea specifică folosită: o tranzacție JDBC, o tranzacție JTA UserTransaction, etc.
- Interfața *Query*. Permite efectuarea de interogări asupra bazei de date și controlează modul în care este executată interogarea. Interogările pot fi scrise în HQL sau folosind direct limbajul SQL corespunzător bazei de date.

Exemplu

- Etape:
  - Crearea claselor Java corespunzătoare entităților
  - Crearea fișierelor de mapare (adnotări)
  - Crearea fișierului de configurare Hibernate
  - Implementarea nivelului de persistență folosind funcțiile corespunzătoare
  - Testarea claselor

Exemplu - Entitatea

```
@Entity
public class Book {
    private  Long  id;
    private String title;

    public Book() {} //obligatoriu
    public Book(String title){ this.title=title;}

    @Id
    @GeneratedValue(strategy=IDENTITY)
    public Long getId() { return id;}

    public void setId(Long id) {this.id = id;}

    @NotNull
    public String getTitle() { return title;}

    public void setTitle(String title) {this.title = title;}
}
```

Exemplu - Entitatea

```
@Entity
@Table( name = "Messages")
public class Message {
    @Id
    @Column(name = "idMessage")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

  @Column(name = "Message_Text")
  private String text;

  public Message() {} //obligatoriu
  public Message(String text) {  this.text = text; }

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

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

Fișierul de configurare *hibernate.properties*

```
#hibernate.properties –numele implicit căutat de Hibernate

# URL-ul de conectare la baza de date (JDBC)
jakarta.persistence.jdbc.url=jdbc:sqlite:/Users/grigo/HibernateExample/hibernate6.db
# Credentials
#jakarta.persistence.jdbc.user=
#jakarta.persistence.jdbc.password=

jakarta.persistence.schema-generation.database.action=update
#none/create/etc

# SQL statement logging
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.highlight_sql=true
```

Exemplu – Salvarea unei entități

```
//INSERT
void addBook(Book b){
    sessionFactory.inTransaction(session -> {
        session.persist(b);
        System.out.println("Book added "+b);
    });
}

//Book b=new Book("Test Book ");

//dupa rulare
//Book added Book{id='8', title='Test Book '}
```

Exemplu – Modificarea unei entități

```
  //FindOne
Book findBook(int id){
    try(Session session=sessionFactory.openSession()){
       return session.find(Book.class, id);
    } //se apelează session.close()
    catch (Exception e){
        System.err.println("Eroare la findBook " +e);
    }
    return null;
}
//update
Book update(int id){
     Book b=findBook(id);
     b.setTitle("New Title");
    sessionFactory.inTransaction(session->{
        session.merge(b);
        //session.flush()
    });
     return b;
}
```

Exemplu – ștergerea unei entități

```
//DELETE
void delete(Book book){
    sessionFactory.inTransaction(session -> {
        session.remove(book);
    });
}
```

Exemplu – selectarea unei entități

```
//SELECT
 List<Book> getBooks(){
    try( Session session= sessionFactory.openSession()) {
        return session.createQuery("from Book where title like 'Test %'",
   Book.class).setFirstResult(0).
                setMaxResults(5).list();
    } //se apelează session.close()
}


//Book - numele entității, NU a tabelei
//title - proprietate a clasei Book, NU numele coloanei din tabela
```

Exemplu – MainBook

```
public class MainBook {
    static SessionFactory sessionFactory;
    public static void main(String[] args) {
        // A SessionFactory is set up once for an application!
         sessionFactory = new Configuration() //cauta fișierul hibernate.properties
                .addAnnotatedClass(Book.class)
                .buildSessionFactory();


        Book b=new Book("Test Book ");
        // persist an entity
        addBook(b);

         System.out.println("Searching book having id 1 "+findBook(1));
         update(6);

       // query data using HQL
         for(Book bb:getBooks())
             System.out.println(bb);

         //delete an entity
         delete(b);

        //close sessionFactory
        sessionFactory.close();

    }
   //… codul metodelor addBook, findBook, update, getBooks, delete
}
```

Configurare Gradle

```
plugins {
    id 'java'
    id 'application'
}

group = 'ro.mpp'
version = '1.0'

repositories {
    mavenCentral()
}

dependencies {

    implementation 'org.hibernate.orm:hibernate-core:6.4.4.Final'

    // Hibernate Validator
    implementation 'org.hibernate.validator:hibernate-validator:8.0.0.Final'
    implementation 'org.glassfish:jakarta.el:4.0.2'

    //pentru lucrul la diferite baze de date: Sqlite, MariaDb, etc…
   implementation 'org.hibernate.orm:hibernate-community-dialects:6.4.4.Final'


    testImplementation platform('org.junit:junit-bom:5.9.1')
    testImplementation 'org.junit.jupiter:junit-jupiter'

    // driverul pentru conectarea la o baza de date Sqlite
   runtimeOnly 'org.xerial:sqlite-jdbc:3.45.3.0'

}
```

Interogări ale bazei de date

- Două posibilități:
  - Hibernate Query Language

```
session.createQuery("from Category c where c.name like
   'Laptop%'");
```

  - SQL

```
session.createNativeQuery(
"select {c.*} from CATEGORY {c} where NAME like 'Laptop%'",
   "c", Category.class);
```

Obținerea rezultatelor

- Metodele `list()/getResultList()` execută interogarea și returnează rezultatul ca și o listă:

```
List<User> result = session.createQuery("from User”,
   User.class).getResultList();
```

- Un singur obiect ca și rezultat :

```
Bid maxBid =session.createQuery("from Bid b order by
        b.amount desc”, Bid.class)
        .setMaxResults(1)
        .uniqueResult();
```

Interogări cu parametri

- Parametrii cu nume

```
String queryString = "from Item item where item.description
   like :searchString and item.date > :minDate";
List result = session.createQuery(queryString)
.setParameter("searchString", searchS)
.setParameter("minDate", minD).list();
```

- Parametrii cu poziție:

```
String queryString = "from Item item where item.description
   like ?1 and item.date > ?2”;
List result = session.createQuery(queryString)
.setParameter(1, searchString)
.setParameter(2, minDate)
.list();
```

Hibernate Query Language (HQL)

Suportă aproape toate funcțiile și operațiile SQL:

- from clause: `from Cat as cat`
- select clause: `select foo from Foo foo, Bar bar where foo.startDate = bar.date`
- where clause: `from Cat as cat where cat.name='Fritz'`
- aggregate functions:

```
select cat.color, sum(cat.weight), count(cat) from Cat cat
   group by cat.color
```

- order by clause
- group by clause
- expressions
- etc.

Maparea relațiilor

Adnotări pentru maparea relațiilor dintre clase:

- **`@ManyToMany`**
- **`@ManyToOne`**
- **`@OneToOne`**
- **`@JoinTable`** - tabela de asociere
- **`@JoinColumn`**

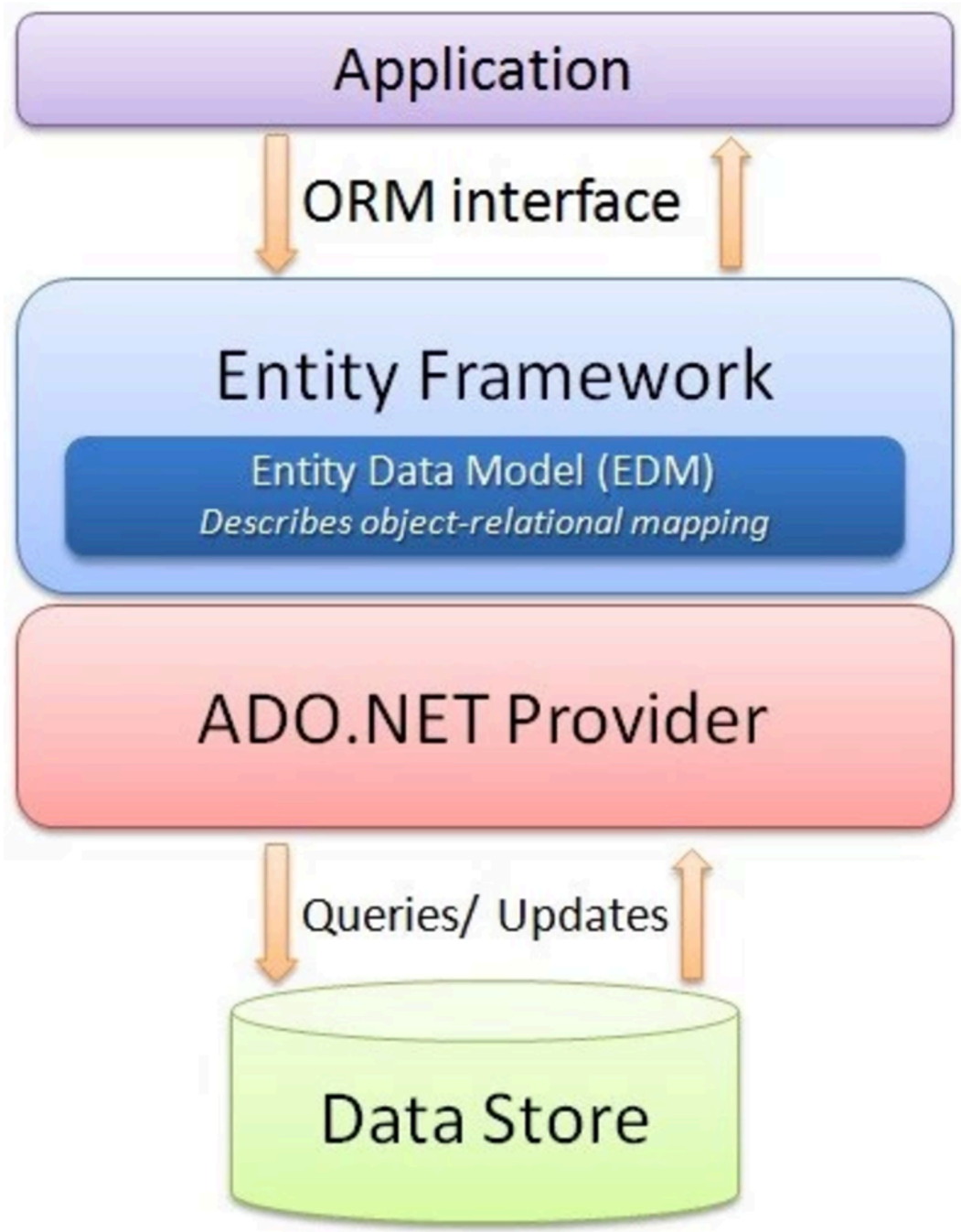
Fetching - lazy, eager

https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html

Exemplu Hibernate

.NET ORM

- Instrumente ORM bazate pe LINQ:
  - LINQ to SQL (L2S)
  - Entity Framework (EF)
- EF permite o mai bună decuplare a claselor de modelul relațional



.NET L2S

- Decorarea claselor folosind atribute .NET
- Spațiul de nume System.Data.Linq.Mapping

```
[Table (Name="Customers")]
public class Customer
{
   [Column(IsPrimaryKey=true)]
    public int ID {get;set};
   [Column (Name="FullName")]
    public string Name {get; set};
}
```

.NET L2S

```
var context = new DataContext ("database connection string”);

 Table<Customer> customers = context.GetTable <Customer>();

  // numărul de înregistrări din tabelă.
     Console.WriteLine (customers.Count());

// Clientul cu Id-ul 2.
   Customer cust = customers.Single (c => c.ID == 2);

   Customer cust = customers.OrderBy (c => c.Name).First();
   cust.Name = "Updated Name";
   context.SubmitChanges();
```

.NET EF

- Decorarea claselor folosind atribute .NET
- Referință către System.Data.Entity.dll

```
[EdmEntityType (NamespaceName = "EFModel", Name = "Customer")]
public partial class Customer
{
[EdmScalarPropertyAttribute (EntityKeyProperty=true,
  IsNullable=false)]
public int ID { get; set; }

[EdmScalarProperty (EntityKeyProperty = false, IsNullable = false)]
     public string Name { get; set; }
}
```

.NET EF

```
var context = new ObjectContext ("entity connection string");
   context.DefaultContainerName = "EntitiesContainer";
   ObjectSet<Customer> customers =
   context.CreateObjectSet<Customer>();

// numărul de înregistrări din tabelă
    Console.WriteLine (customers.Count());

// Clientul cu ID-ul 2
Customer cust = customers.Single (c => c.ID == 2);


Customer cust = customers.OrderBy (c => c.Name).First();
   cust.Name = "Updated Name";
   context.SaveChanges();
```

Servicii Web & REST

Servicii Web

- Definiții:

1. Un **serviciu web** este o aplicație/componentă soft disponibilă pe internet și care folosește un sistem stardardizat de transmitere a mesajelor în format XML. XML este folosit pentru comunicarea datelor către/de la un serviciu web.
    - Un client apelează un serviciu web trimițând un mesaj în format XML și așteaptă un răspuns în format XML.
    - Comunicarea are loc folosind XML, serviciile web nu sunt dependente de un anumit sistem de operare sau de un anumit limbaj de programare.
2. **Serviciile web** sunt aplicații *self-contained*, modulare și distribuite care pot fi descrise, publicate, localizate și apelate prin rețea pentru a crea produse, procese, etc. Aceste aplicații pot fi locale, distribuite sau pe web. Serviciile web sunt dezvoltate folosind standarde ca TCP/IP, HTTP și XML.

3. **Serviciile web** sunt sisteme de schimbare a mesajelor bazate pe XML care folosesc Internetul pentru interacțiunea dintre aplicații. Aceste sisteme pot fi: programe, obiecte, mesaje, documente, etc.

4. Un **serviciu web** este o colecție de protocoale și standarde folosite pentru transmiterea datelor între aplicații sau sisteme. Aplicații dezvoltate în diferite limbaje de programare și rulând pe diferite platforme (sisteme de operare) pot folosi serviciile web pentru a schimba date în rețea (ex. Internet) într-o manieră similară comunicării între procese pe același calculator. Interoperabilitatea acestora este posibilă datorită folosirii standardelor.

Servicii Web

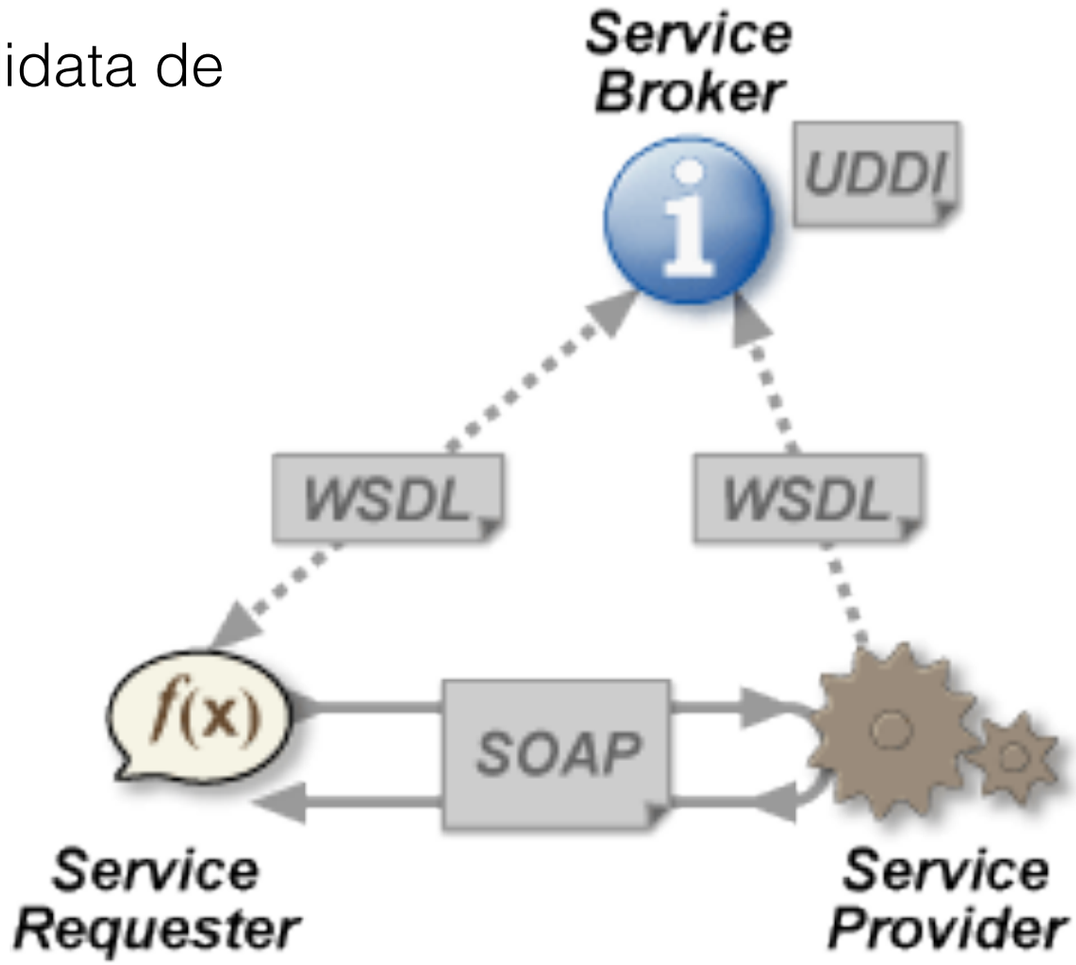
- Un serviciu web este un serviciu:
  - Disponibil pe Internet sau într-o rețea privată.
  - Folosește un sistem standardizat de schimbare a mesajelor bazat pe XML.
  - Nu este dependent de un anumit limbaj de programare sau de un anumit sistem de operare.
  - Este self-describing folosind o gramatică XML (DTD, XML Schema).
  - Poate fi descoperit folosind mecanisme simple.
- Componentele: structura de baza pentru serviciile web este XML + HTTP. Toate serviciile web standard funcționează folosind componentele:
  - XML pentru marcarea informației
  - SOAP (Simple Object Access Protocol) pentru transmiterea mesajelor
  - UDDI (Universal Description, Discovery and Integration)
  - WSDL (Web Services Description Language) pentru a descrie disponibilitatea serviciului

Exemplu WSDL

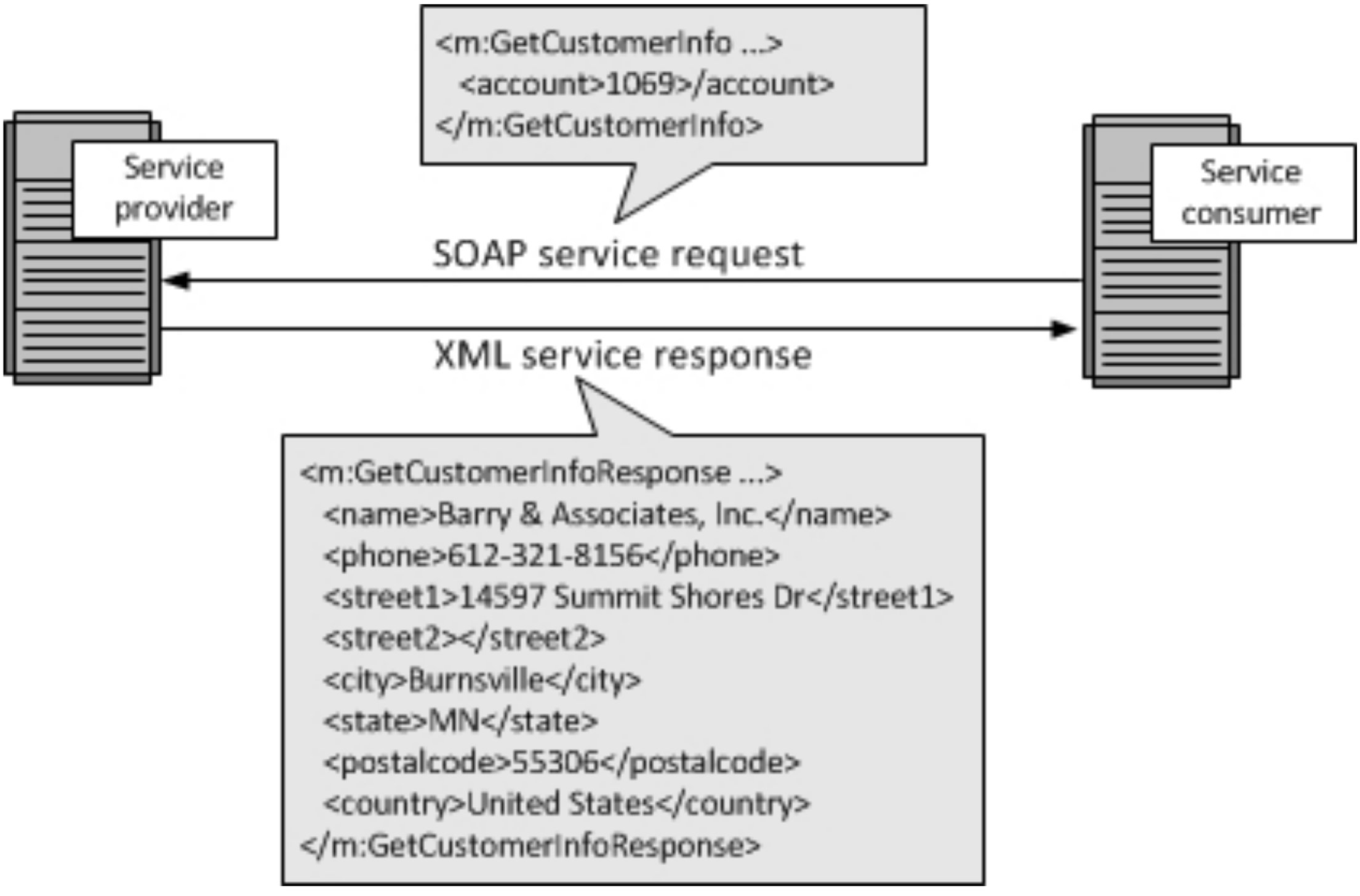
```
<definitions name = "HelloService"
   targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>

   <message name = "SayHelloResponse">
      <part name = "greeting" type = "xsd:string"/>
   </message>

   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
         <output message = "tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name = "Hello_Binding" type = "tns:Hello_PortType">
      <soap:binding style = "rpc"
         transport = "http://schemas.xmlsoap.org/soap/http"/>
      <operation name = "sayHello">
         <soap:operation soapAction = "sayHello"/>
         <input>
            <soap:body
               encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
               namespace = "urn:examples:helloservice"
               use = "encoded"/>
         </input>

         <output>
            <soap:body
               encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
               namespace = "urn:examples:helloservice"
               use = "encoded"/>
         </output>
      </operation>
   </binding>

   <service name = "Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding = "tns:Hello_Binding" name = "Hello_Port">
         <soap:address
            location = "http://www.examples.com/SayHello/" />
      </port>
   </service>
</definitions>
```

Servicii Web - Arhitectura

- Furnizorul serviciului trimite un fișier WSDL serviciului UDDI.
- Clientul serviciului (consumatorul) contactează UDDI pentru a descoperi cine este furnizorul datelor de care are nevoie, iar apoi contacteaza furnizorul serviciului folosind protocolul SOAP.
- Furnizorul serviciului validează cererea și trimite datele cerute în format XML folosind protocolul SOAP.
- Structura datelor în format XML ar trebui validata de client folosind un fișier XSD (XML Schema).



Servicii Web - Arhitectura



REST

- REpresentational State Transfer (REST).
- Este un stil arhitectural pentru sisteme distribuite hipermedia.
- A fost propus de Roy T. Fielding în teza sa de doctorat "*Architectural Styles and the Design of Network-based Software Architectures*" (2000).
- Conține 6 constrângeri ce trebuie satisfăcute pentru a fi numit sistem REST veritabil:

1. Client–server
2. Fără stare (eng. *stateless*)
3. Cacheable
4. Interfață uniformă (eng. *uniform interface*)
5. Sistem stratificat (eng. *layered system*)
6. Cod la cerere (eng. *code on demand*) -opțională

REST - Resursa

- Conceptul de bază în REST este *resursa* (eng. *resource*).
- Orice informație care poate fi numită poate fi **resursă**: un document, o imagine, o colecție de alte resurse, un obiect real (ex. persoană, mașină), etc.
- REST folosește identificatorul resursei pentru a identifica resursa în momentul interacțiunii între componente.
- Starea unei resurse la un anumit moment de timp este numită ***reprezentarea resursei***.
- Reprezentarea constă din date, metadata care descriu datele și legături hipermedia care ajută clienții în tranziția la următoarea stare dorită.
- Formatul datelor dintr-o reprezentare este numit ***tip media*** (eng. *media type*).
- Tipul media identifică o specificație care definește modul în care reprezentarea va fi procesată.
- Un API RESTful veritabil seamănă cu hypertext. Fiecare informație ce poate fi obținută are o *adresă explicită* (prin legături și atribute id) sau *implicită* (obținută din definiția tipului media și structura reprezentării).
- Reprezentările resurselor trebuie să se descrie pe ele (eng. *self-descriptive*): clientul nu trebuie să știe că resursa este persoană sau mașină. Datele trebuie să poate fi procesate pe baza tipului media asociat resursei.