# Modelowanie Systemów Informacyjnych (MSI)

dr inż. Mariusz Trzaska
*mtrzaska@pjwstk.edu.pl*

## Wykład 13

### Wykorzystanie modelu relacyjnego w obiektowych językach programowania (2)

*http://www.mtrzaska.com*

# Zagadnienia

- Niezgodność impedancji i jej konsekwencje
- Różne sposoby rozwiązania problemu
- Microsoft LINQ
- Hibernate
  - Wprowadzenie,
  - Mapowanie obiektów,
  - Mapowanie asocjacji,
  - Mapowanie atrybutów powtarzalnych.
- Podsumowanie

# Niezgodność impedancji

- Łączenie:
  - **modelu obiektowego z** języka programowania,
  - **modelu relacyjnego** ze składu danych

  skutkuje zjawiskiem zwanym niezgodnością impedancji.

```
// [...]

// Wykonaj zapytanie
ResultSet result = db_statement.executeQuery("select * from employee");

// Przetwarzanie wynikow
while (result.next() )
{
        System.out.println ("ID : " + result.getInt("ID"));
        System.out.println ("Name : " + result.getString("Name"));
}
```

W efekcie zamiast operować na obiektach (np. Pracownik) działamy w oparciu o ich elementy, np. `String` z nazwiskiem.

# Niezgodność impedancji (2)

- Rozwiązania problemu
  - Stosowanie tego samego modelu w języku programowania oraz źródle danych.
    - Mało kto zdecyduje się na programowanie w modelu relacyjnym (programowanie strukturalne).
    - Można wprowadzić istotne udogodnienia z zakresu baz danych do języka programowania m. in. język zapytań.
      - Microsoft C# i LINQ
  - Wykorzystanie dedykowanych bibliotek, np. Hibernate.

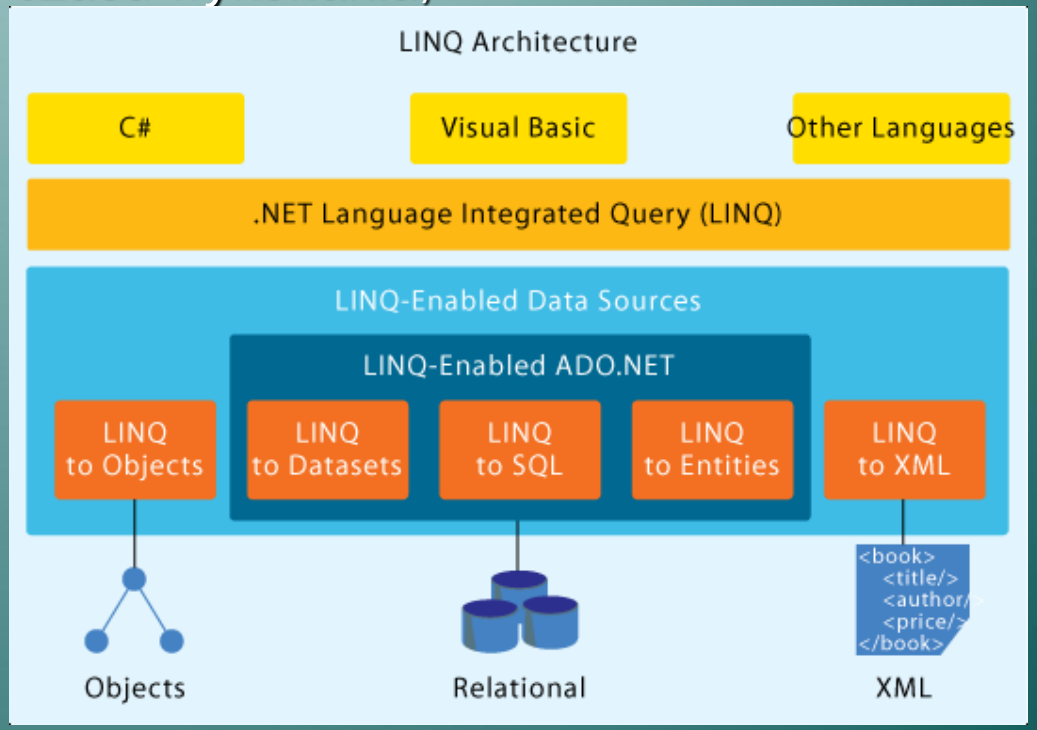
# LINQ

- **L**anguage **In**tegrated **Q**uery
- Autorem tej technologii jest

  Anders Hejlsberg, który:
  - jako pierwszy stworzył IDE (Borland Turbo Pascal),
  - jest również autorem języka TypeScript.
- Wprowadzenie LINQ porównywane jest do przełomu spowodowanego przez dBase oraz FoxPro.
- Do istniejącego języka programowania (m. in. MS C#) dodano składnię i funkcjonalność umożliwiającą pisanie zapytań podobnych do SQL.

# LINQ (2)

- Dzięki temu problem niezgodności jest mocno zredukowany.
- Dodatkowe korzyści:
  - Wykorzystanie metadanych czasu wykonania,
  - Kontrola typologiczna w czasie kompilacji,
  - Wykorzystanie IntelliSense®.
- Różne warianty:
  - LINQ to Objects,
  - LINQ to XML,
  - LINQ to ADO.NET,
  - LINQ to JSON (Json.NET),
- **Rewolucja?**



# LINQ (3)

- Przykłady

```
var locals = from c in customers
             where c.ZipCode == 91822
             select new { FullName = c.FirstName + " " +
                          c.LastName, HomeAddress = c.Address };

IEnumerable<Product> x =
        from p in products
        where p.UnitPrice >= 10
        select p;

IEnumerable<Product> MostExpensive10 =
        products.OrderByDescending(p => p.UnitPrice).Take(10);

var custOrders =
        from c in customers
        join o in orders on c.CustomerID equals o.CustomerID
        select new { c.Name, o.OrderDate, o.Total };

IEnumerable<Product> orderedProducts1 =
        from p in products
        orderby p.Category, p.UnitPrice descending, p.Name
        select p;
```

# LINQ (4)

- Przykłady – c. d.

```
var q =
        from c in db.Customers
        where c.City == "London"
        select c;

foreach (Customer c in q)
        Console.WriteLine(c.CompanyName);



var q =
        from o in db.Orders
        where o.ShipVia == 3
        select o;
foreach (Order o in q) {
        if (o.Freight > 200)
                SendCustomerNotification(o.Customer);
        ProcessOrder(o);
}
```

# Hibernate

- Zgodnie z informacją twórców Hibernate to:
  - *Hibernate is a powerful, high performance object/relational persistence and query service*
  - Wydajna usługa zapewniająca trwałość danych oraz umożliwiająca zadawanie zapytań.
- Istnieją wersje dla Java oraz MS .NET.
- Projekt rozwijany jako *open source* od roku 2001:
  - 76 000 linii kodu *core*,
  - 36 000 linii kodu *unit test*,
  - 3000 pobrań każdego dnia.
- *http://www.hibernate.org/*

# Hibernate (2)

- Teraz piszą o nim: *More than an ORM, discover the Hibernate galaxy*.
- Galaktyka Hibernate, m.in.:
  - *Hibernate ORM. Domain model persistence for relational databases.*
  - *Hibernate Search. Full-text search for your domain model.*
  - *Hibernate Validator. Annotation based constraints for your domain model.*
  - *Hibernate OGM. Domain model persistence for NoSQL datastores.*
  - *Hibernate Tools. Command line tools and IDE plugins for your Hibernate usages.*
- Niestety nie likwiduje całkowicie problemu niezgodności impedancji.

# Hibernate - wydajność

- Twórcy twierdzą, że biblioteka jest wydajna i wykorzystuje różne techniki optymalizacji:
  - Cache'owanie obiektów,
  - Cache'owanie wyników zapytań,
  - Polecenia SQL wykonywane dopiero wtedy gdy są naprawdę potrzebne,
  - Brak uaktualniania niezmodyfikowanych obiektów,
  - Efektywne zarządzanie kolekcjami,
  - Łączenie wielu zmian w jedno UPDATE,
  - Leniwa inicjalizacja obiektów i kolekcji.

# Hibernate - środowisko testowe

- Bazujemy na oficjalnym tutorialu:

  *http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html*
- Hibernate domyślnie działa w oparciu o bazę danych poprzez JDBC.
- Wykorzystamy bazę HSQL (*http://hsqldb.org/*) napisaną w całości w języku Java.
- Utworzenie katalogu z bibliotekami *lib*.
- Uruchomienie (z katalogu data)

```
java -classpath ../lib/hsqldb.jar org.hsqldb.Server
```

# Podstawy biblioteki Hibernate

- Stworzymy prostą aplikację umożliwiającą:
  - planowanie zdarzeń,
  - Łączenie zdarzeń z uczestnikami.
- Każda klasa, która chce wykorzystywać pełnię możliwości biblioteki, musi posiadać specyficzny atrybut służący do identyfikacji wystąpień.
  - `private long id;`
  - Zmianą jego wartości zajmuje się Hibernate.
- Zalecane wykorzystywanie konwencji *JavaBean*
  - `set…()`,
  - `get…()`.

# Podstawy biblioteki Hibernate (2)

```
public class Event {
    private Long id;
    private String title;
    private Date date;
    public Event() {}

    public Long getId() {
        return id;
    }
    private void setId(Long id) {
        this.id = id;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}
```

# Plik mapujący

- Wykorzystywany do mapowania klas języka Java (model obiektowy) na elementy w bazie danych (model relacyjny).
- Nazwa: *NazwaKlasy*.hbm.xml
- Lokalizacja: w tym samym katalogu co odpowiadający mu plik źródłowy *NazwaKlasy*.java.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="events.Event" table="EVENTS">
     [...]
     </class>
</hibernate-mapping>
```

# Plik mapujący (2)

- Tag `class`
  - `name` oznacza nazwę klasy języka Java,
  - `table` informuje w jakiej tabeli przechowywać wystąpienia tej klasy.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="events.Event" table="EVENTS">
     [...]
    </class>
</hibernate-mapping>
```

# Plik mapujący (3)

- Tag `id` służy do tworzenia identyfikatorów (kluczy)
  - `name` oznacza nazwę atrybutu klasy języka Java,
  - `column` informuje w której kolumnie przechowywać jego wartości.
  - `generator` określa strategię zapewniania unikalności liczb.

```
<class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
</class>
```

# Plik mapujący (4)

- Tag `property` mapuje atrybuty.
  - `type` definiuje typ kolumny. Należy go użyć gdy nie da się tego „odgadnąć” automatycznie, np. `java.util.Date` może być przedstawione jako SQL
    - `date`,
    - `Timestamp`,
    - `time`.

```
<class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
</class>
```

# Plik konfiguracyjny

- Lokalizacja: główny katalog z plikami źródłowymi

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</property>
        <!-- Disable the second-level cache  -->
        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Drop and re-create the database schema on startup -->
        <!-- <property name="hbm2ddl.auto">create</property> -->
        <mapping resource="events/Event.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

# Klasa pomocnicza

- Uruchomienie Hibernate
- Globalny obiekt `SessionFactory`,
  - Bezpieczna obsługa wielu wątków,
  - Tworzy pojedyncze sesje: `Session`.

```
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

# Pierwsze użycie Hibernate

- Dodajemy proste wydarzenie „Zdarzenie 01” z aktualną datą.

```
public class EventManager {
    public static void main(String[] args) {
        EventManager mgr = new EventManager();

        mgr.createAndStoreEvent(„Zdarzenie 01", new Date());

        HibernateUtil.getSessionFactory().close();
    }

    private void createAndStoreEvent(String title, Date theDate) {

        Session session = HibernateUtil.getSessionFactory().getCurrentSession();

        session.beginTransaction();

        Event theEvent = new Event();
        theEvent.setTitle(title);
        theEvent.setDate(theDate);

        session.save(theEvent);

        session.getTransaction().commit();
    }
}
```

# Pierwsze użycie Hibernate (2)

- Efekt działania możemy podejrzeć w pliku *log*.

```
20:24:36,897 DEBUG SchemaExport:303 - create table EVENTS (EVENT_ID bigint generated by default as
identity (start with 1), EVENT_DATE timestamp, title varchar(255), primary key (EVENT_ID))

20:24:36,975 DEBUG ConnectionManager:421 - opening JDBC connection

20:24:36,975 DEBUG AbstractSaveEventListener:153 - saving [events.Event#<null>]
20:24:36,975 DEBUG AbstractSaveEventListener:244 - executing insertions
20:24:36,991 DEBUG AbstractSaveEventListener:297 - executing identity-insert immediately
20:24:36,991 DEBUG AbstractEntityPersister:2144 - Inserting entity: events.Event (native id)
20:24:36,991 DEBUG AbstractBatcher:366 - about to open PreparedStatement (open PreparedStatements: 0,
globally: 0)
20:24:36,991 DEBUG SQL:401 - insert into EVENTS (EVENT_DATE, title, EVENT_ID) values (?, ?, ?)
Hibernate: insert into EVENTS (EVENT_DATE, title, EVENT_ID) values (?, ?, ?)
20:24:36,991 DEBUG AbstractBatcher:484 - preparing statement
20:24:36,991 DEBUG AbstractEntityPersister:1992 - Dehydrating entity: [events.Event#<null>]
20:24:36,991 DEBUG AbstractBatcher:374 - about to close PreparedStatement (open PreparedStatements: 1,
globally: 1)
20:24:36,991 DEBUG AbstractBatcher:533 - closing statement
20:24:36,991 DEBUG AbstractBatcher:366 - about to open PreparedStatement (open PreparedStatements: 0,
globally: 0)
20:24:36,991 DEBUG SQL:401 - call identity()
Hibernate: call identity()
20:24:37,007 DEBUG ThreadLocalSessionContext:300 - allowing proxied method [getTransaction] to proceed to
real session
20:24:37,007 DEBUG JDBCTransaction:103 – commit
```

# Pobieranie danych poprzez Hibernate

- Po „wygenerowaniu” danych, należy zakomentować fragment pliku *hibernate.cfg.xml* (w przeciwnym wypadku dane będą usuwane)

```
<!-- Drop and re-create the database schema on startup -->
<!-- <property name="hbm2ddl.auto">create</property> -->
```

- Metoda pobierające „ekstensję” klasy `Event` (za pomocą zapytania HQL)

```
private List listEvents() {

        Session session = HibernateUtil.getSessionFactory().getCurrentSession();

        session.beginTransaction();

        List result = session.createQuery("from Event").list();

        session.getTransaction().commit();

        return result;
}
```

# Pobieranie danych poprzez Hibernate (2)

- Wykorzystanie metody do pobierania danych

```
EventManager mgr = new EventManager();

List events = mgr.listEvents();

for (int i = 0; i < events.size(); i++) {
     Event theEvent = (Event) events.get(i);

     System.out.println("Event: " + theEvent.getTitle() + " Time: "
                                + theEvent.getDate());
}

HibernateUtil.getSessionFactory().close();
```

- Jak widać pracujemy na kompletnych obiektach, a nie (jak w przypadku „czystego JDBC) na wartościach (np. `String`) z bazy danych.

# Asocjacje w Hibernate

- Asocjacje są (prawie) automatycznie mapowane na relacje w bazie danych.
- Elementy, które trzeba uwzględnić to:
  - Kierunek,
  - Liczności,
  - Zachowanie się kolekcji służącej do implementacji (po stronie Javy).

# Dodanie asocjacji skierowanej

- Stworzymy klasę Osoba (*Person*) i powiążemy ją z wydarzeniami (*Events*).

```
public class Person {
    private Long id;
    private int age;
    private String firstname;
    private String lastname;

    public Person() {}
    // Metody get/set oraz prywatny set dla id.
}
```

- Plik mapujący dla klasy *Person*

```
<class name="events.Person" table="PERSON">
        <id name="id" column="PERSON_ID">
            <generator class="native"/>
        </id>
        <property name="age"/>
        <property name="firstname"/>
        <property name="lastname"/>
</class>
```

- Nowy wpis do pliku konfiguracyjnego.

# Dodanie asocjacji skierowanej (2)

- Do klasy *Person* dodamy informację o jej wydarzeniach (*Events*).

```
public class Person {
    // [...]
    private Set events = new HashSet();
    public Set getEvents() {
        return events;
    }
    public void setEvents(Set events) {
        this.events = events;
    }
```

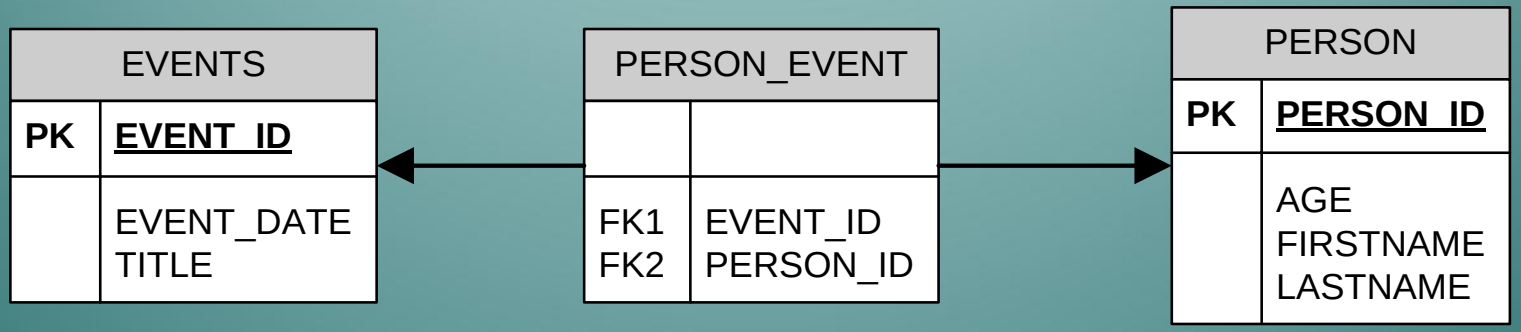
- I uzupełnimy plik mapujący dla klasy *Person*

```
<class name="events.Person" table="PERSON">
     […]
     <set name="events" table="PERSON_EVENT">
          <key column="PERSON_ID"/>
          <many-to-many column="EVENT_ID" class="events.Event"/>
     </set>
</class>
```

- Wykorzystaliśmy pojemnik typu `Set` (obsługiwane są też inne).

# Dodanie asocjacji skierowanej (3)

- W efekcie otrzymaliśmy:
  - poniższy schemat relacyjny (tabela pośrednicząca została wygenerowana „automatycznie”)




  - oraz połączenie asocjacją (referencjami języka Java) od klasy `Person` do `Event` (ale nie w drugą stronę).

# Użycie asocjacji skierowanej

- Metoda tworząca nowe powiązanie pomiędzy Person oraz Event

```
private void addPersonToEvent(Long personId, Long eventId) {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();

        Person aPerson = (Person) session.load(Person.class, personId);
        Event anEvent = (Event) session.load(Event.class, eventId);
        aPerson.getEvents().add(anEvent);
        session.getTransaction().commit();
}
```

  - Powiązanie dodajemy poprzez modyfikację pojemnika języka Java,
  - Hibernate to automatycznie wykrywa i w tle uaktualnia bazę danych,
  - Analogiczna „automatyka” istnieje dla atrybutów.

# Użycie asocjacji skierowanej (2)

- Stworzenie
  - Zdarzenia,
  - Osoby
- Dodanie Osoby do Zdarzenia.

```
EventManager mgr = new EventManager();

Long eventId = (Long) mgr.createAndStoreEvent("Zdarzenie 01", new Date());
Long personId = (Long) mgr.createAndStorePerson("Jan", "Kowalski");
mgr.addPersonToEvent(personId, eventId);
System.out.println("Dodano osobe " + personId + " do zdarzenia " + eventId);

HibernateUtil.getSessionFactory().close();
```

- Zasadnicza wada podejścia Hibernate:
  - **Korzystamy z identyfikatorów, zamiast natywnych referencji!**

# Dodanie asocjacji dwukierunkowej

- Schemat relacyjny nie ulega zmianie.
- Dodajemy pojemnik z osobami uczestniczącymi w zdarzeniu do klasy Event.

```
public class Event {
      // [...]
      private Set participants = new HashSet();
      public Set getParticipants() {
            return participants;
      }
      public void setParticipants(Set participants) {
            this.participants = participants;
      }
}
```

- Uzupełniamy plik mapujący `Event.hbm.xml`.

```
<hibernate-mapping>
    <class name="events.Event" table="EVENTS">
          [...]
          <set name="participants" table="PERSON_EVENT" inverse="true">
              <key column="EVENT_ID"/>
              <many-to-many column="PERSON_ID" class="events.Person"/>
          </set>
    </class>
</hibernate-mapping>
```

# Dodanie asocjacji dwukierunkowej (2)

- Tag `set`
  - Użycie prawie takie samo jak w przypadku asocjacji skierowanej,
  - Nazwy kolumn `key` oraz `many-to-many`, dla dwóch plików mapujących, są zamienione.
  - Atrybut `inverse="true"`
- Dodajemy odpowiednie metody do klasy Person

```
public class Person {
      // [...]
      public void addToEvent(Event event) {
            this.getEvents().add(event);
            event.getParticipants().add(this);
      }
      public void removeFromEvent(Event event) {
            this.getEvents().remove(event);
            event.getParticipants().remove(this);
      }
}
```

# Dodanie asocjacji dwukierunkowej (3)

- Podsumowując: w celu zamiany asocjacji skierowanej na dwukierunkową należy:
  - Oznaczyć jedną ze stron jako `inverse`,
  - Dla liczności „1 - *” musi to być strona „*”,
  - Dla liczności „* - *” można wybrać dowolną stronę.

# Atrybuty powtarzalne

- W Hibernate nazywane są kolekcją wartości (*Collection of values*).
- Różnica w stosunku do asocjacji polega na tym, że wartości nie są współdzielone (a obiekty oczywiście tak).
- Dla klasy Osoba (*Person*) dodamy listę adresów e-mail.
- Specjalny tag `element`
  - Oznacza, że kolekcja nie zawiera odniesień do innych wystąpień, ale listę elementów typu String.

# Atrybuty powtarzalne (2)

- Dodajemy pojemnik z adresami e-mail do klasy `Person`.

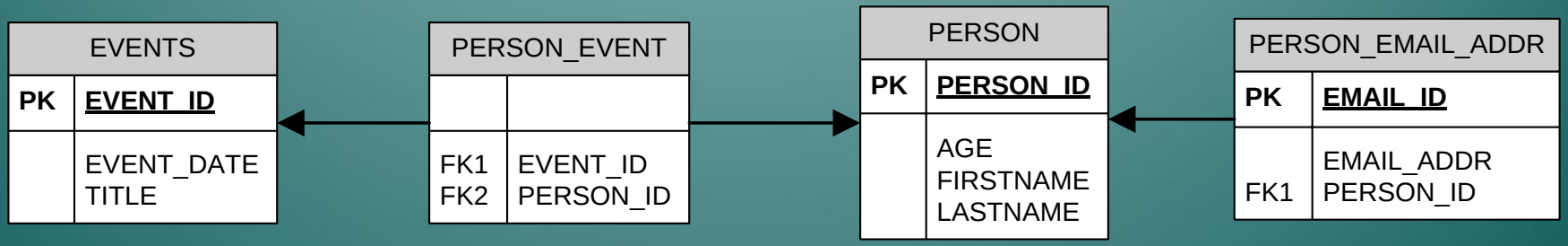
```
public class Person {
      // [...]

      private Set emailAddresses = new HashSet();

      public Set getEmailAddresses() {
            return emailAddresses;
      }

      public void setEmailAddresses(Set emailAddresses) {
            this.emailAddresses = emailAddresses;
      }
}
```

# Atrybuty powtarzalne (3)

- Uzupełniamy plik mapujący `Event.hbm.xml`.

```
<hibernate-mapping>
    <class name="events.Event" table="EVENTS">
          [...]
          <set name="emailAddresses" table="PERSON_EMAIL_ADDR">
              <key column="PERSON_ID"/>
              <element type="string" column="EMAIL_ADDR"/>
          </set>
    </class>
</hibernate-mapping>
```

- Uaktualniony schemat relacyjny



# Zapytania w Hibernate

- Przykładowe zapytania w Hibernate Query Language (HQL)

```
List cats = session.createQuery(
    "from Cat as cat where cat.birthdate < ?")
    .setDate(0, date)
    .list();

List mothers = session.createQuery(
    "select mother from Cat as cat join cat.mother as mother where cat.name = ?")
    .setString(0, name)
    .list();

List kittens = session.createQuery(
    "from Cat as cat where cat.mother = ?")
    .setEntity(0, pk)
    .list();

Cat mother = (Cat) session.createQuery(
    "select cat.mother from Cat as cat where cat = ?")
    .setEntity(0, izi)
    .uniqueResult();]]

Query mothersWithKittens = (Cat) session.createQuery(
    "select mother from Cat as mother left join fetch mother.kittens");
Set uniqueMothers = new HashSet(mothersWithKittens.list());
```

# Zapytania w Hibernate (2)

- Przetwarzanie wyników zapytań

```
Iterator iter = sess.createQuery("from eg.Qux q order by q.likeliness").iterate();

while ( iter.hasNext() ) {
    Qux qux = (Qux) iter.next();  // pobierz obiekt

    // Metoda, ktorej nie mozna bylo zawrzec w zapytaniu
    if ( qux.calculateComplicatedAlgorithm() ) {

        // Usun aktualna instancje
        iter.remove();

        // Koniec
        break;
    }
}
```

# Podsumowanie

- Niezgodność impedancji czyli trudności w pogodzeniu świata relacyjnego oraz obiektowego, jest bardzo poważnym, praktycznym problemem.
- Można wyróżnić dwa generalne podejścia do jego rozwiązania:
  - Modyfikacje języka programowania, tak aby posiadał jak najwięcej zalet baz danych,
  - Stworzenie dodatkowych bibliotek ułatwiających pracę z danymi.

# Podsumowanie (2)

- Przykładem pierwszego podejścia do problemu jest Microsoft C# 3.0 razem z LINQ.
  - Język zapytań (funkcjonalnie podobny do SQL) stał się częścią języka programowania.
  - Niezgodność impedancji znika całkowicie, czyli nie musimy mapować konstrukcji obiektowych na relacyjne i odwrotnie.
  - Dzięki temu mamy m. in. kontrolę typologiczną.

# Podsumowanie (2)

- Drugie podejście jest reprezentowane przez Hibernate
  - Biblioteka rzeczywiście ułatwia pracę z bazami danych,
  - Niestety zamiast natywnych referencji wykorzystuje identyfikatory.
- Wydaje się, że zdecydowanie lepszym podejściem jest dodanie cech baz danych do języka programowania, tak jak w Microsoft LINQ.