Introduction to NHibernate

Motorola Software Group
Argentina Software Center

Gabriel Cerutti

Revision History

Version	Date	Summary of changes	Author
1.0.0	20 Nov 09	Initial version	Gabriel Cerutti

Agenda

- Introduction to ORM
- Introduction to NHibernate
- Demo

Introduction to ORM

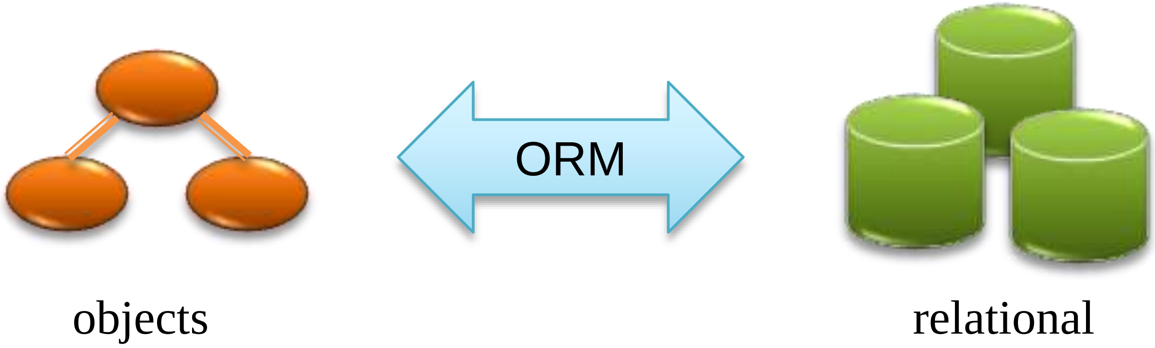
- **Object-relational impedance mismatch**
- **What is ORM?**
- **ORM Benefits**
- **ORM into n-tier architecture**

Object-relational impedance mismatch

- The **object-relational impedance mismatch** is a set of conceptual and technical difficulties that are often encountered when a relational database management system is being used by a program written in an object-oriented programming language.
- Mismatches:
  - **Object-oriented concepts:** encapsulation, Invariance, Accessibility, Interface, inheritance and polymorphism.
  - **Data type differences**
  - **Structural and integrity differences**
  - **Manipulative differences**
  - **Transactional differences**

What is ORM?

- **Object Relational Mapping is a programming technique for converting data between incompatible type systems in relational databases and object oriented programming languages.**
  - **Objects are hierarchical**
  - **Databases are relational**
  - **ORM minimizes Object-relational impedance mismatch**



ORM Benefits

Productivity

- Eliminates lots of repetitive code – focus on business logic
- Database schema is generated automatically

Maintainability

- Fewer lines of code – easier to understand
- Easier to manage change in the object model

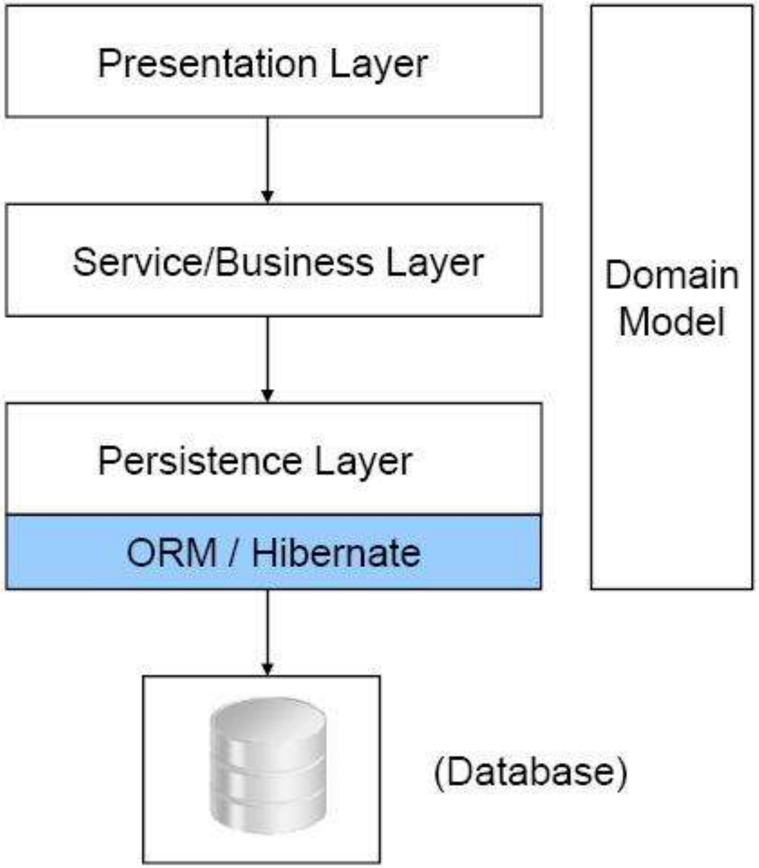
Performance

- Lazy loading – associations are fetched when needed
- Caching

Database vendor independence

- The underlying database is abstracted away
- Can be configured outside the application

ORM into n-tier architecture



MySQl, Oracle, SQL Server, etc.

Introduction to NHibernate

- **Key Features**
- **High Level Architecture**
- **API Main Components**
- **Instance States**
- **Configuration**
- **Persistent Classes**
- **Basic O/R Mapping**
- **Collection Mapping**
- **Inheritance Mapping**
- **Data Manipulation**

Key Features

- **NHibernate is a port of Hibernate Core for Java to the .NET Framework**
- **Natural programming model:**
  - NHibernate supports natural OO idiom; inheritance, polymorphism, composition and the .NET collections framework, including generic collections
- **Native .NET:**
  - NHibernate API uses .NET conventions and idioms
- **Support for fine-grained object models:**
  - A rich variety of mappings for collections and dependent objects

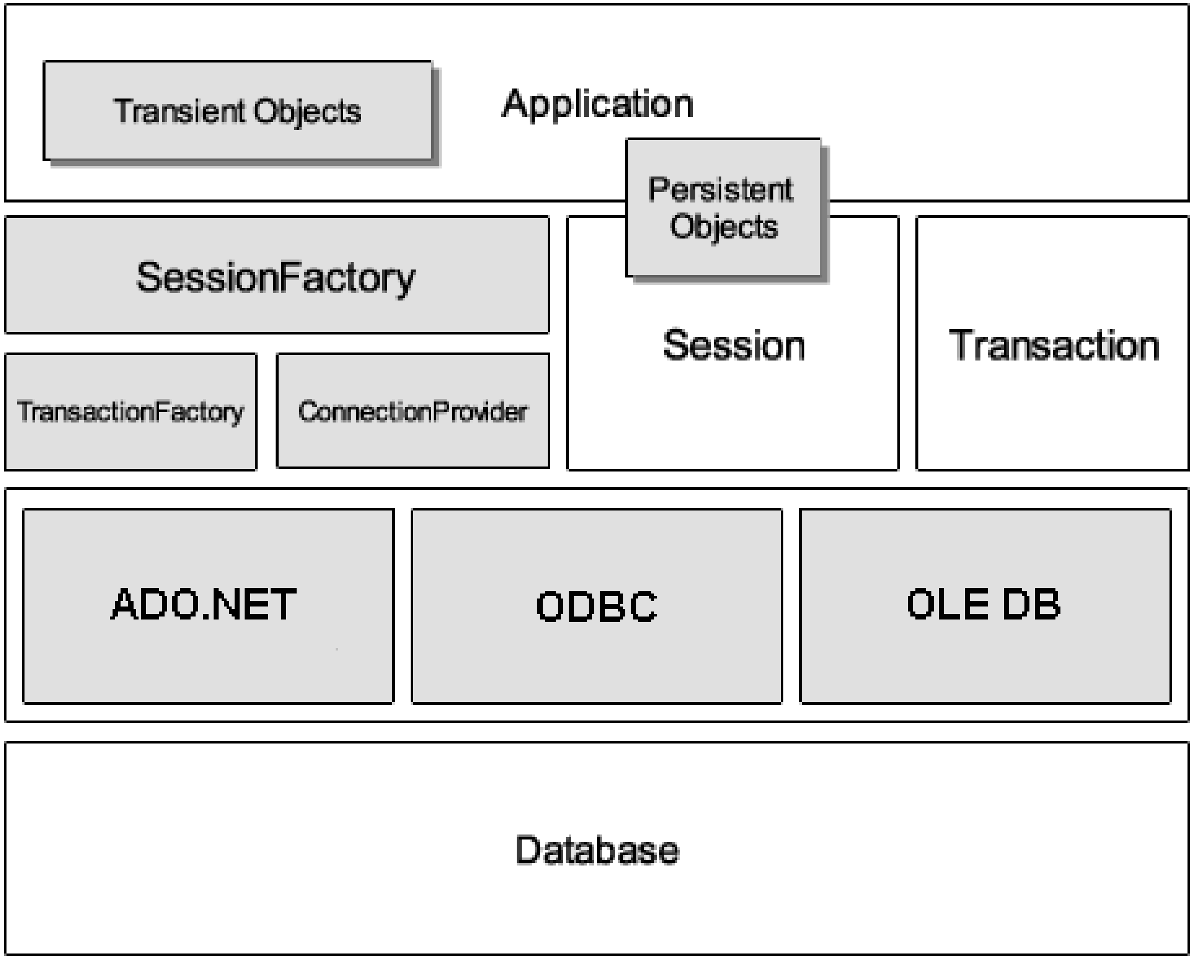
Key Features

- **The query options:**
  - NHibernate addresses both sides of the problem; not only how to get objects into the database, but also how to get them out again.
  - Query API with HQL
  - Criteria API
  - Native SQL
- **Custom SQL:**
  - Specify the exact SQL that NHibernate should use to persist your objects. Stored procedures are supported on Microsoft SQL Server
- **Support for "conversations“:**
  - NHibernate supports long-lived persistence contexts, detach/reattach of objects, and takes care of optimistic locking automatically

Key Features

- **All popular databases supported**
  - Oracle, SQL Server, DB2, SQLite, PostgreSQL, MySQL, Sybase, etc.
- **XML-based configuration files and Attribute-based configuration**
- **Good community support**
- **Free/open source:**
  - NHibernate is licensed under the LGPL (Lesser GNU Public License)

High Level Architecture



Main Interfaces

- **ISessionFactory:** A threadsafe (immutable) cache of compiled mappings for a single database. A factory for ISession and a client of IConnectionProvider.
- **ISession:** A single-threaded, short-lived object representing a conversation between the application and the persistent store. Wraps an ADO.NET connection.
- **ITransaction:** (Optional) A single-threaded, short-lived object used by the application to specify atomic units of work.
- **IQuery and ICriteria**

Very Important Concept

Persistent Objects and Collections

- Short-lived, single threaded objects containing persistent state and business function. These might be ordinary POCOs, the only special thing about them is that they are currently associated with (exactly one) ISession.

Transient Objects and Collections

- Instances of persistent classes that are not currently associated with a ISession. They may have been instantiated by the application and not (yet) persisted or they may have been instantiated by a closed ISession.

Instance States

Transient

- The instance is not, and has never been associated with any persistence context. It has no persistent identity (primary key value).

Persistent

- The instance is currently associated with a persistence context. It has a persistent identity (primary key value) and, perhaps, a corresponding row in the database.

Detached

- The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process. It has a persistent identity and, perhaps, a corresponding row in the database.

Configuration

- **Programmatic Configuration**

1- Add hbm.xml files

```
Configuration cfg = new Configuration()
cfg.AddFile(“Student.hbm.xml”);
```

2- Add entity class (mapping file must be an embedded resource)

```
Configuration cfg = new Configuration()
cfg.AddClass(typeof(Motx.NHDemo.Student));
```

3- Add an assembly (mapping file must be an embedded resource)

```
Configuration cfg = new Configuration()
cfg.AddAssembly(“Motx.NHDemo.Core”);
```

Configuration

- ## File Configuration

```
Configuration cfg = new Configuration()
cfg.Configure(sessionFactoryConfigPath);
```

Example:

```
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2" >
  <session-factory name=“NHDemo">
    <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
    <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
    <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
    <property name="connection.connection_string">
                    Server=127.0.0.1;Database=nhdemo;Uid=root;Pwd=motorola;
    </property>
    <!-- HBM Mapping Files -->
    <mapping assembly="Mds.NHDemo.Core" />
  </session-factory>
</hibernate-configuration>
```

- ## Obtaining an ISessionFactory

When all mapping have been parsed by the configuration the app must obtain a factory for ISession:

```
ISessionFactory sessionFactory = cfg.BuildSessionFactory();
```

Persistent Classes

- Classes that implement the entities of the business problem
- They can be transient and also persistent instance stored in the database
- Also known as the Plain Old CRL Object (POCO)
  - Declare accessors for persistent fields
  - Implement a default constructor (no-argument)
  - Provide an identifier property (optional)
  - Prefer non-sealed classes and virtual methods (optional)

Persistent Classes

- **Example:**

```
public class Student
  {
      private String _id;
      private String _name;

      public Student() { }

      public virtual String Id
      {
         get { return _id; }
         set { _id = value; }
      }

      public virtual String Name
      {
         get { return _name; }
         set { _name = value; }
      }
  }
```

Basic O/R Mapping

- **Class:** The <*class*> element declares a persistent class.
- **Id:** Mapped classes must declare the primary key of the database table. The <*id*> element defines the mapping to the primary key column.
- **Property:** The <*Property*> element declares a persistent property of the class.
  - NHibernate types
  - Dot NET native types
  - Enumeration types
  - Custom types
- **Many-To-One:** An ordinary association to another persistent class (It´s really just an object reference)
- **One-To-One:** A one-to-one association to another persistent class
  - Primary key associations
  - Unique foreign key associations

Collection Mapping

- **Many-To-Many:** A collection table is required with foreign key columns. Example:

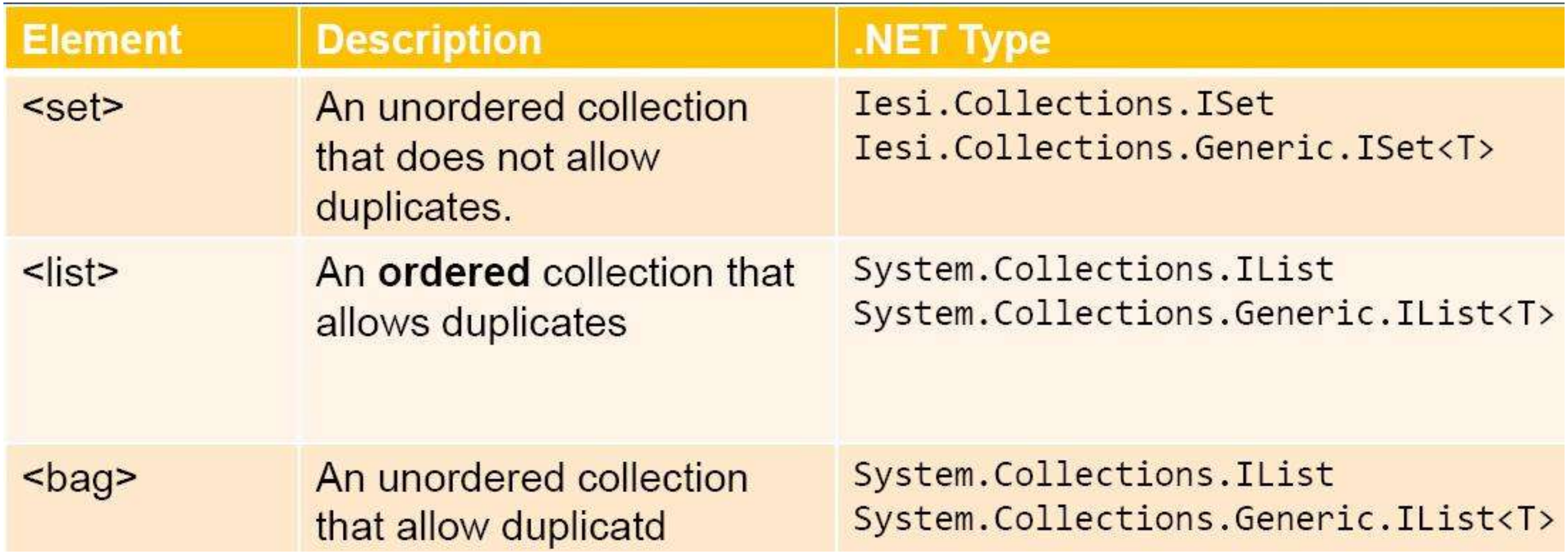
```
<bag name="Students" table="Course_has_Student" lazy="false" cascade="none">
    <key column="idCourse" />
    <many-to-many class="Student" column="idStudent"/>
</bag>
```

- **One-To-Many:** Links the tables of two classes directly, with no intervening collection table

```
<bag name="Courses">
    <key column="idCatalog" />
    <one-to-many class="Course"/>
</bag>
```

- **Lazy Initialization:** Collections may be lazily initialized, meaning they load their state from the database only when the application needs to access it.

Collection Mapping



Element	Description	.NET Type

<set>	An unordered collection that does not allow duplicates.	`Iesi.Collections.ISet`
		`Iesi.Collections.Generic.ISet<T>`

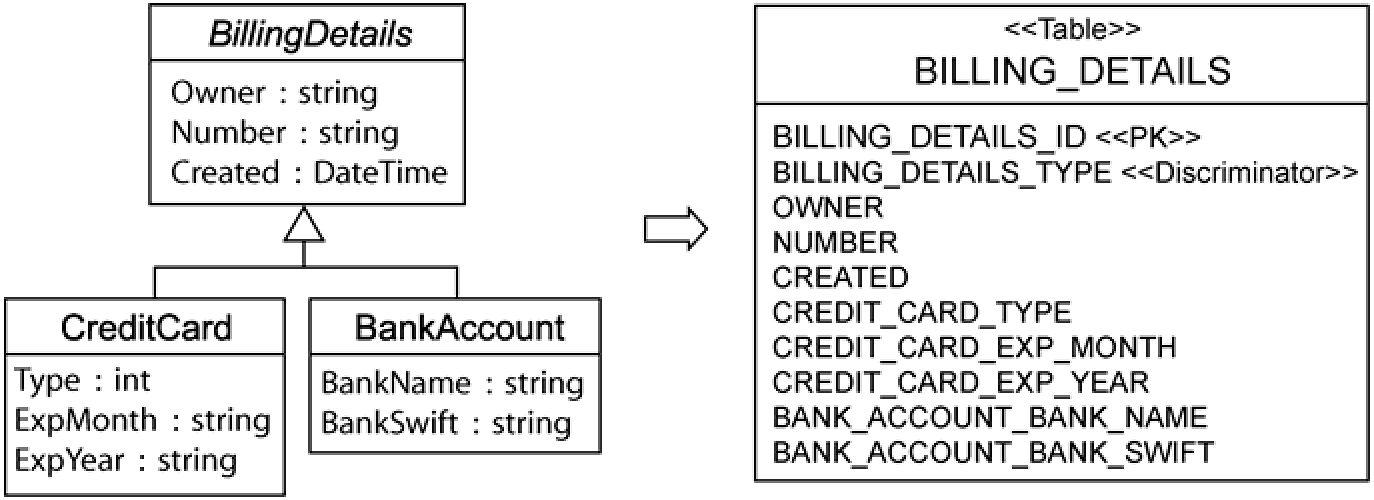
<list>	An **ordered** collection that allows duplicates	`System.Collections.IList`
		`System.Collections.Generic.IList<T>`

<bag>	An unordered collection that allow duplicatd	`System.Collections.IList`
		`System.Collections.Generic.IList<T>`

Inheritance Mapping

- **The Three Strategies:**

  1- Table per class hierarchy



BillingDetails

Owner : string
Number : string
Created : DateTime

CreditCard

Type : int
ExpMonth : string
ExpYear : string

BankAccount

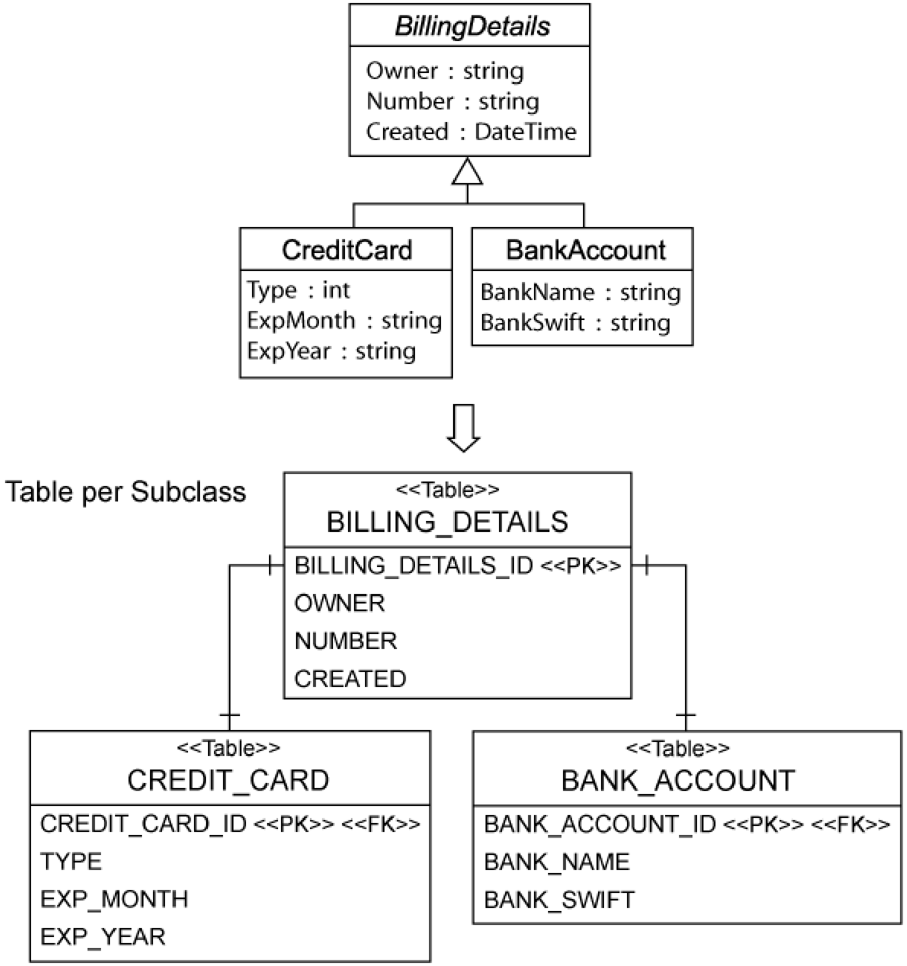
BankName : string
BankSwift : string

<<Table>>
BILLING_DETAILS

BILLING_DETAILS_ID <<PK>>
BILLING_DETAILS_TYPE <<Discriminator>>
OWNER
NUMBER
CREATED
CREDIT_CARD_TYPE
CREDIT_CARD_EXP_MONTH
CREDIT_CARD_EXP_YEAR
BANK_ACCOUNT_BANK_NAME
BANK_ACCOUNT_BANK_SWIFT

Inheritance Mapping

2- Table per subclass



BillingDetails

Owner : string
Number : string
Created : DateTime

CreditCard

Type : int
ExpMonth : string
ExpYear : string

BankAccount

BankName : string
BankSwift : string

Table per Subclass

<<Table>>
BILLING_DETAILS

BILLING_DETAILS_ID <<PK>>
OWNER
NUMBER
CREATED

<<Table>>
CREDIT_CARD

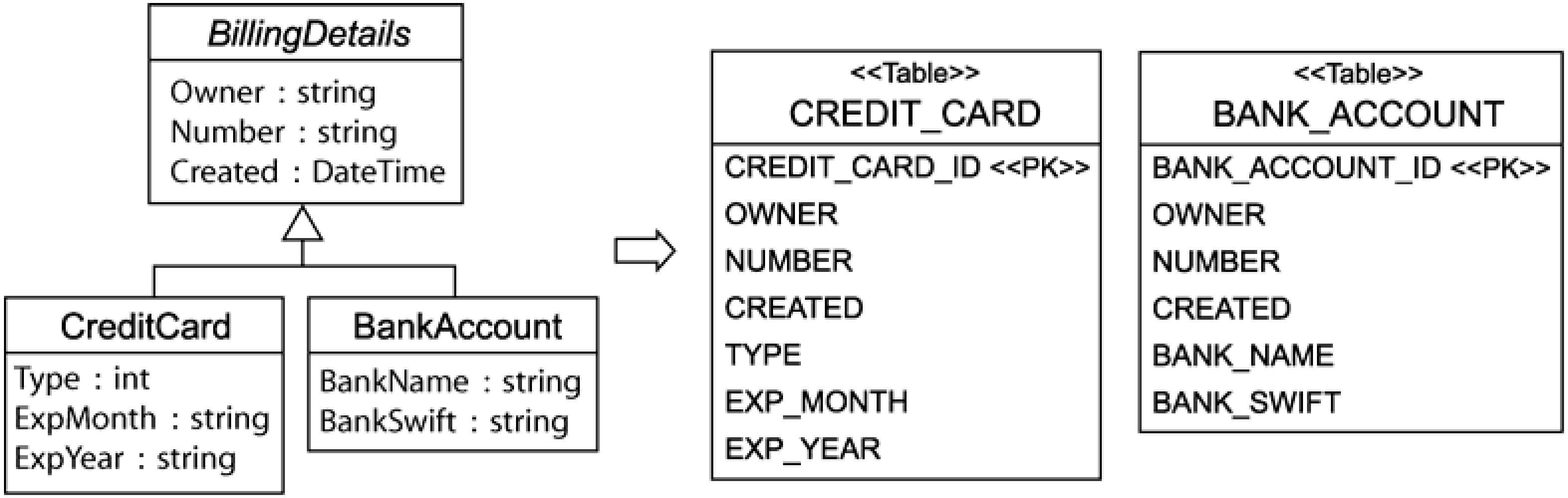
CREDIT_CARD_ID <<PK>> <<FK>>
TYPE
EXP_MONTH
EXP_YEAR

<<Table>>
BANK_ACCOUNT

BANK_ACCOUNT_ID <<PK>> <<FK>>
BANK_NAME
BANK_SWIFT

Inheritance Mapping

3- Table per concrete class



BillingDetails

Owner : string
Number : string
Created : DateTime

CreditCard

Type : int
ExpMonth : string
ExpYear : string

BankAccount

BankName : string
BankSwift : string

<<Table>>
CREDIT_CARD

CREDIT_CARD_ID <<PK>>
OWNER
NUMBER
CREATED
TYPE
EXP_MONTH
EXP_YEAR

<<Table>>
BANK_ACCOUNT

BANK_ACCOUNT_ID <<PK>>
OWNER
NUMBER
CREATED
BANK_NAME
BANK_SWIFT

Attribute Mapping

- **One way to define the mapping metadata is to use .NET attributes. Example:**

```
using NHibernate.Mapping.Attributes;

[Class(Lazy=false)]
public class Category {
...
[Id(Name="Id")]
[Generator(1, Class="native")]
public long Id {
...
}
...
[Property]
public string Name {
...
}
...
}
```

XML Mapping or .NET Attributes?

XML Mapping document	.NET Attributes
XML mapping document are external (independent of the domain model)	Reduce the lines of metadata significantly
They are easier to manipulate for very complex mapping	Type-Safe
They can contain some useful information and named queries	Support auto-completion in the IDE
More configurable at deployment time	Make refactoring of classes and properties easier

Manipulating Persistent Data

- **Saving to the database:**

```
session.Save(entity);
```

- **Loading one entity from database:**

```
session.Load(entityType, entityId);
```

- **Loading an entity list from database:**

```
IQuery query = session.CreateQuery(“from Student”);
IList<Student> students = query.List<Student>();
```

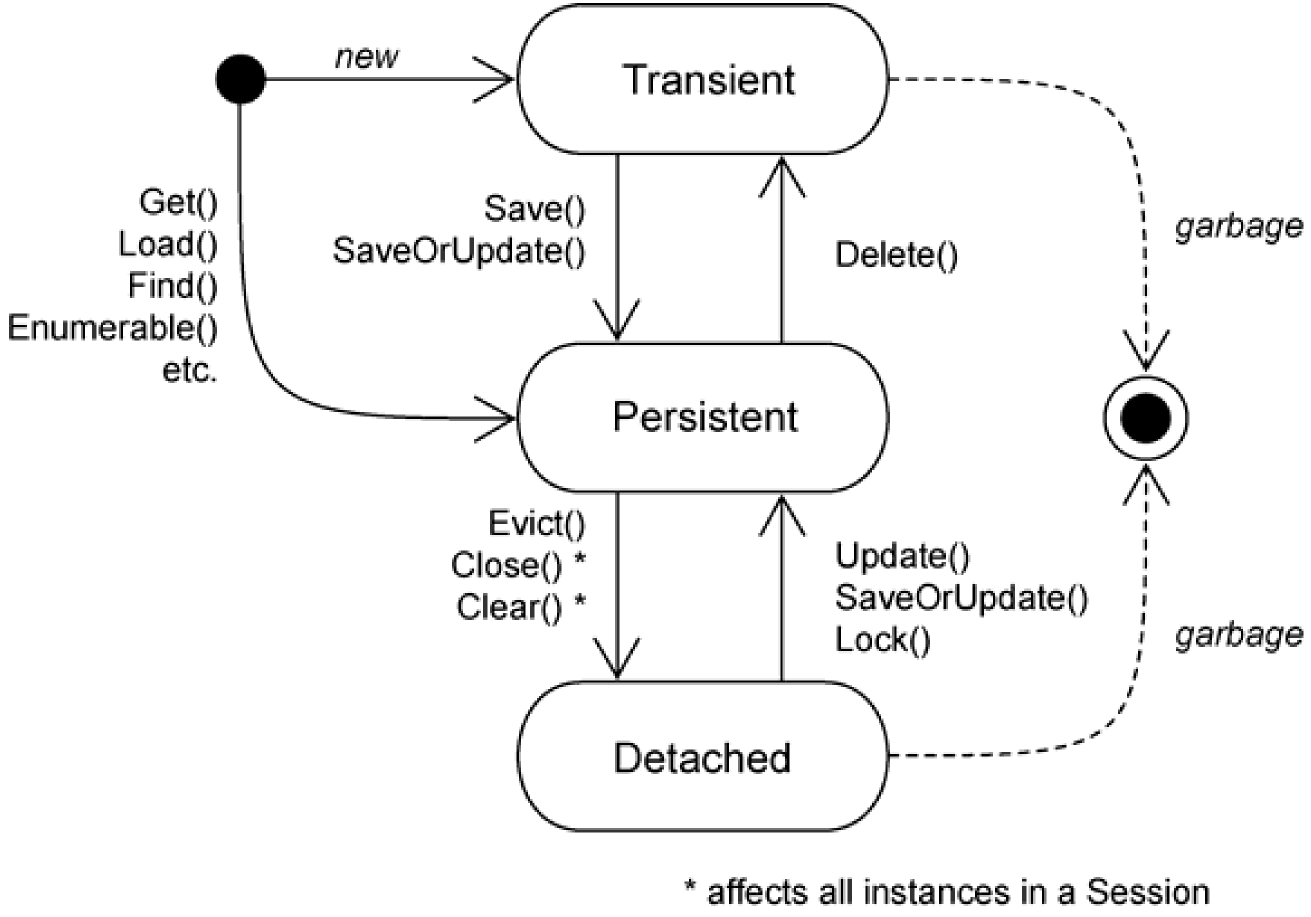
- **Updating an entity:**

```
session.Update(entity);
session.SaveOrUpdate(entity);
```

- **Deleting an entity:**

```
session.Delete(entity);
```

Instance State Diagram



Ending The Session

Flushing the session

- To synchronize the changes with the database
- Not needed if the ITransaction API is used

Commit the transaction

- ITransaction.commit()
- Flushing will be performed implicitly

Close the session

- The ADO.NET connection will be relinquished by the session

Handle exceptions

- Usually NHibernateException
- You should rollback the transaction, close and discard the session instance.

Retrieving Persistent Data

- **Query API**
  - **NHibernate is equipped with an extremely powerful query language, HQL, it is fully object-oriented**
  - **You may obtain an instance of IQuery interface, using CreateQuery()**
  - **You may even define a named query in the mapping document**
  - **Example:**

```
IQuery q = sess.CreateQuery("from Student");
q.SetFirstResult(20);
q.SetMaxResults(10);
IList students= q.List();
```

Retrieving Persistent Data

- **Criteria API**
  - **Builds queries dynamically using an Object-Oriented API (rather than embedding strings in .NET code)**
  - **Example:**

```
ICriteria crit = session.CreateCriteria(typeof(Student));
crit.Add( Expression.Eq(“average", 8) );
crit.SetMaxResults(10);
IList students= crit.List();
```

Retrieving Persistent Data

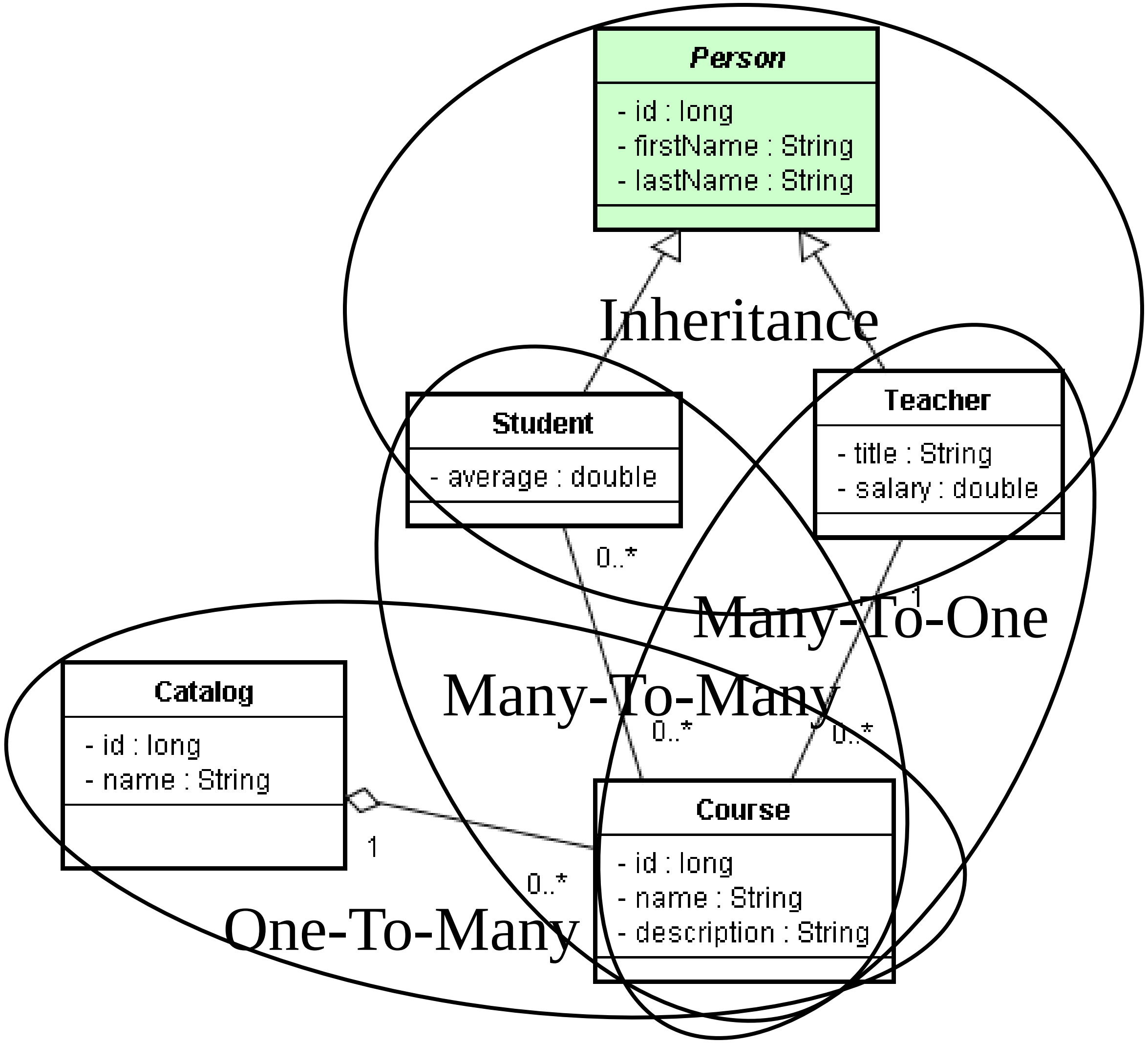
- **Native SQL**
  - **You may a query in SQL, using CreateSqlQuery()**
  - **You must enclose SQL aliases in braces**
  - **Example:**

```
IList students= session.CreateSQLQuery(
"SELECT {student.*} FROM STUDENT{student} WHERE ROWNUM<10",
“student",
typeof(Student)
).List();
```

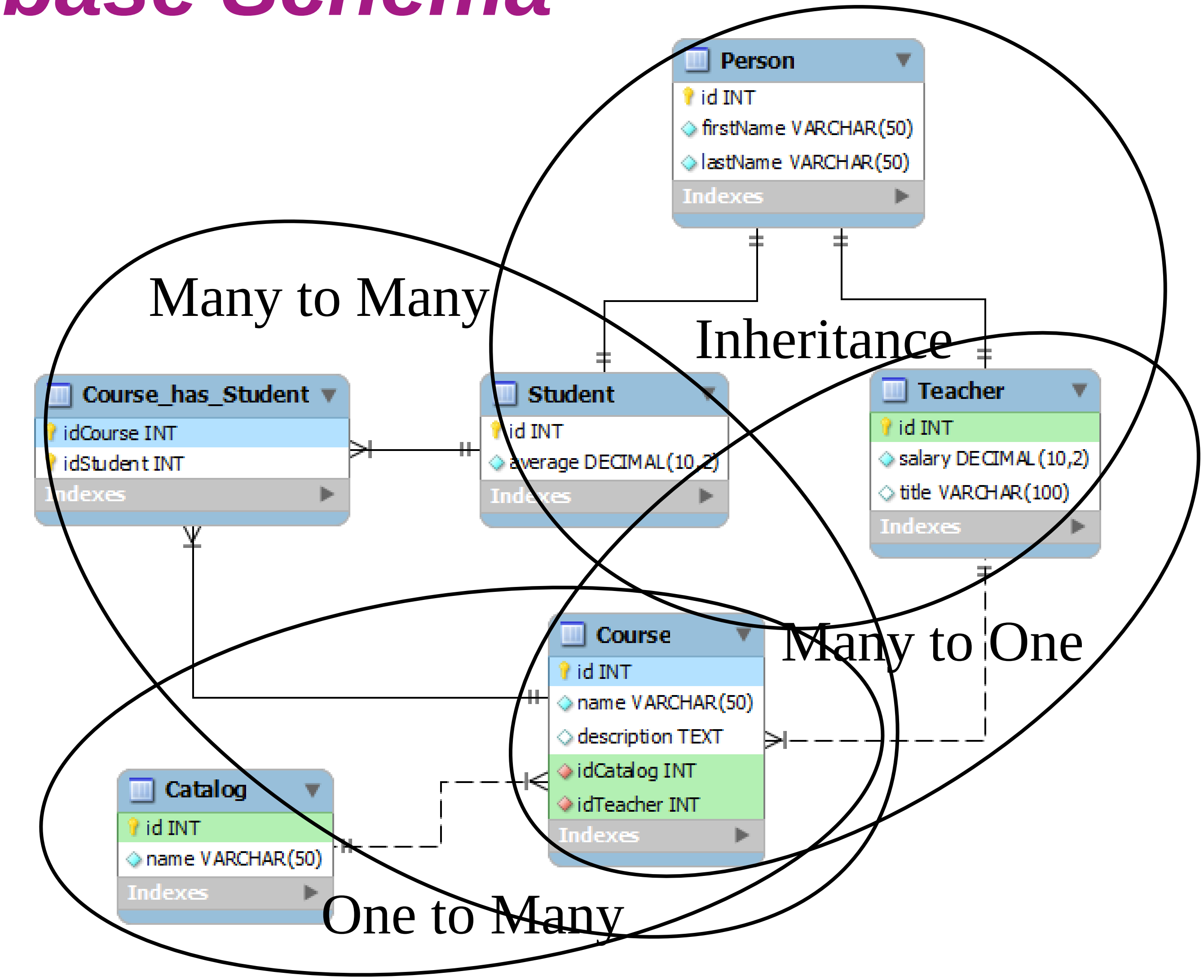
Demo

- Domain Model
- Database Schema
- Create a course
- Register an student in the course

Domain Model



Database Schema



Many to Many

Inheritance

Many to One

One to Many

Lets go to the code…