ADVANCED DATABASE

Object Oriented Databases

Dr. NGUYEN Hoang Ha

Email: nguyen-hoang.ha@usth.edu.vn

Agenda

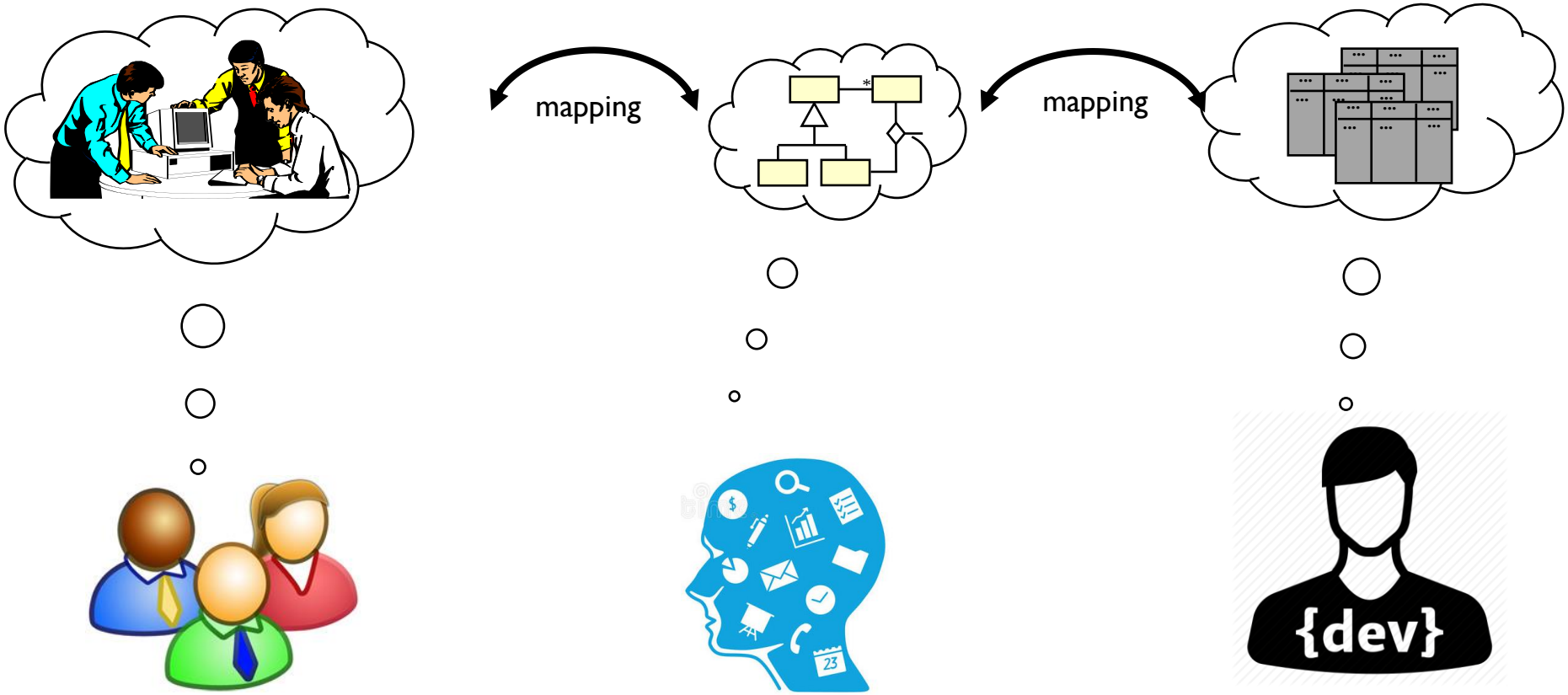
- Problems between OOP and RDBMS
- Object Relation Mapping
- Object Oriented Database

PROBLEMS BETWEEN OOP AND RDBMS

Problems

- The software manufacturing problems:
  - Growing cost of software design, development and deployment,
  - Immature methods of software design and construction,
  - Poor reliability, problems with security
  - Integration of distributed, heterogeneous, redundant and fragmented data and service resources,
  - Obsolete softwares,
  - High cost of software maintenance.
- The main factor of the software problems: Complexity
- **Object-orientation** in new hope in fight with complexity.

Software perception in different views



Users

Software analyst

DB developer

Why relational databases are not enough?

- What are popular?
  - Database: Relational
  - Programming technique: OOP
- → **impedance mismatch**
  - The mapping is difficult, software production cost is higher.
    - complex class structures
    - large unstructured objects
    - object inheritance
  - The relational model and the object model are fundamentally different, and the integrating the two is not straightforward
  - Performance frequently compromised, maintenance cost too high.

Impedance mismatch (cont.)

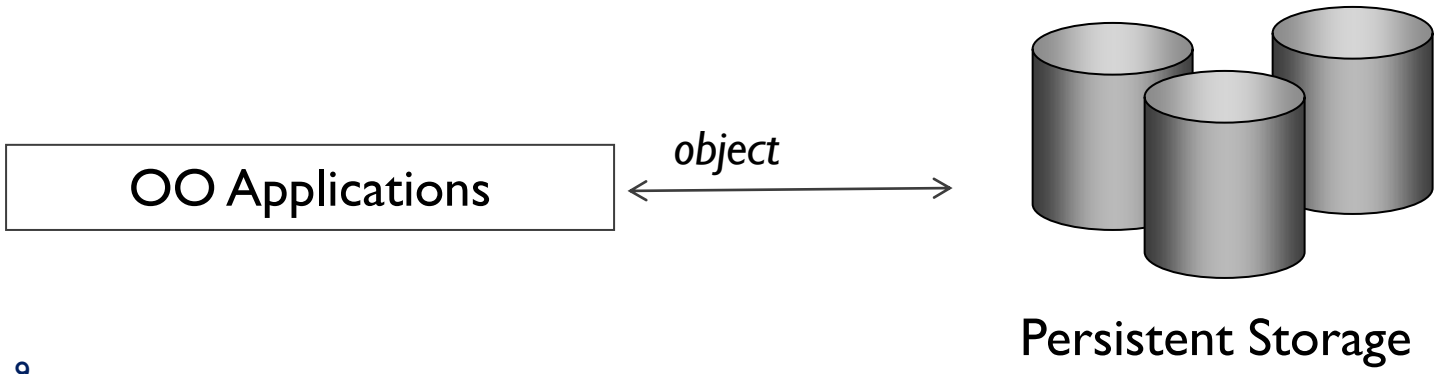
- Goal of object-oriented design is to model a process
- Goal of relational database design is normalisation
- The resulting tables may store data inefficiently, or access to data may be inefficient
- Database stores data as rows in *tables*, which are not like objects.
- Objects have associations and collections databases have relations between tables.

Disappointed Object-relational Wave

- Almost nobody uses object-oriented extensions of relational systems
- Not supported by standards, tools and API-s
- New SQL standards (SQL3) aiming the object-relational model are unsuccessful.

How OO systems should work?

- Applications need to save data to *persistent storage*.
  - Persistent storages::
    - Database
    - Directory service
    - XML files
    - spreadsheet, ...
- For O-O programming, we'd like to save and retrieve *objects* to/from storage.

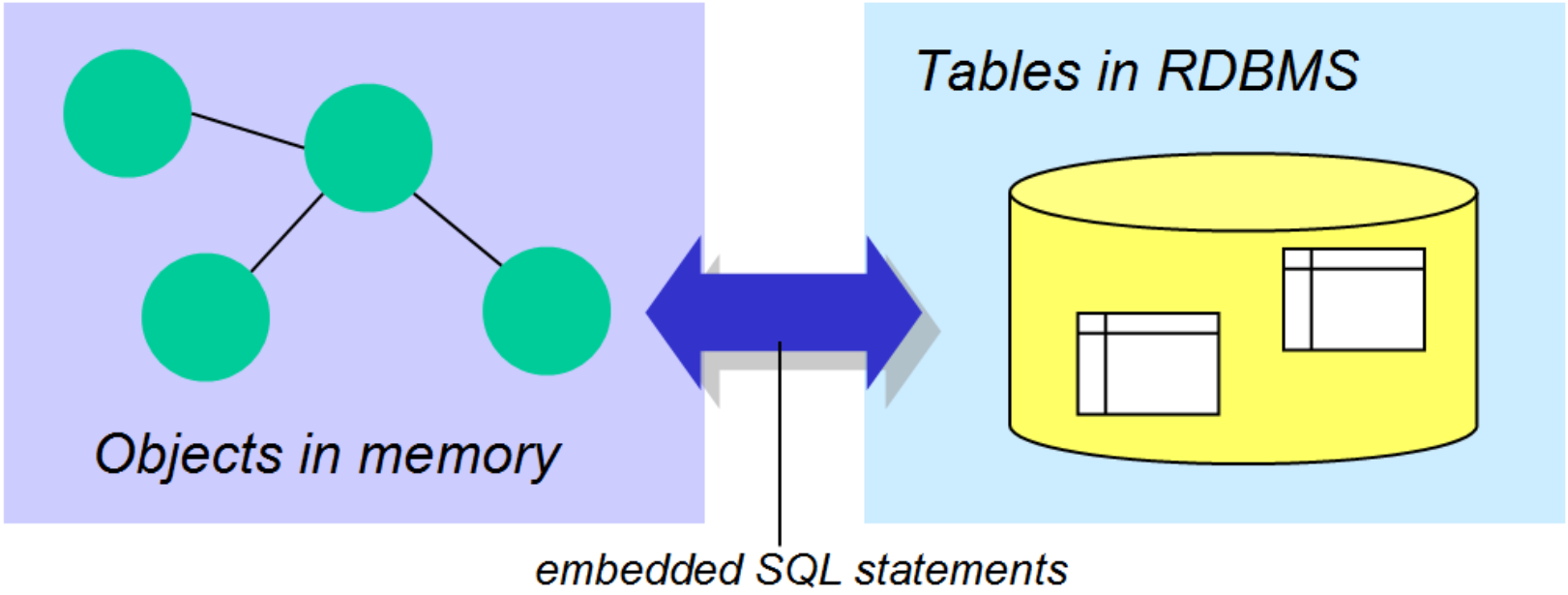


Approaches

- Object-Relation Mapping
- Object Oriented Database

OBJECT RELATIONAL MAPPING

Object Relational Mapping



ORM Overview and Concepts

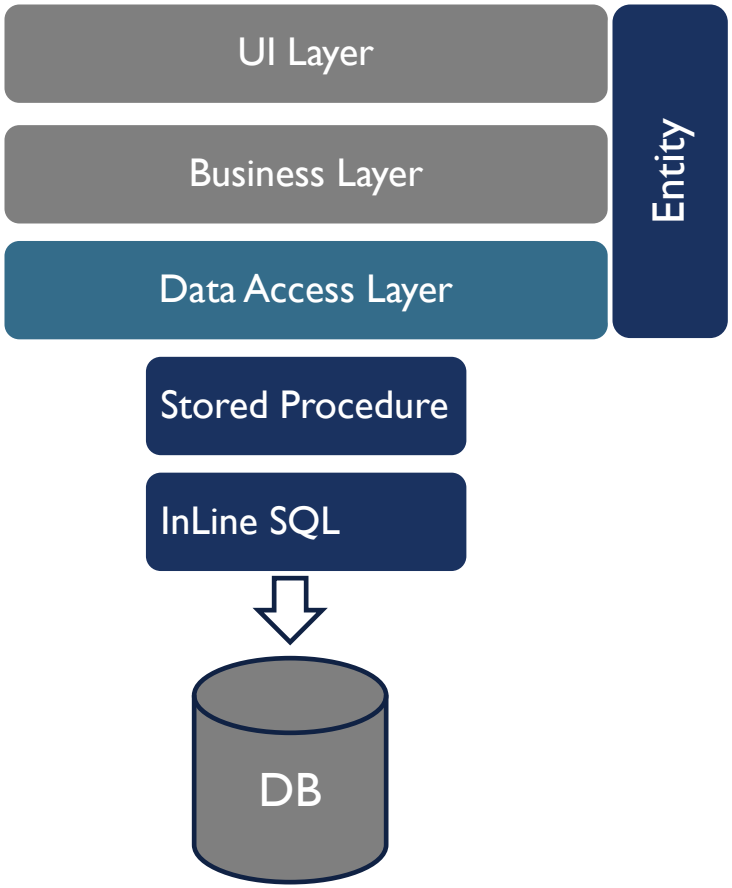
- Is a programming technique for **converting** data between incompatible type systems in **relational databases** and **object oriented programming languages.**
- It creates, in effect, a "virtual object database" that can be used from within the programming language.
- Taxonomy:
  - Free
  - Commercial
  - Private ORM tool of programmers

What does the O/R Mapping Do?

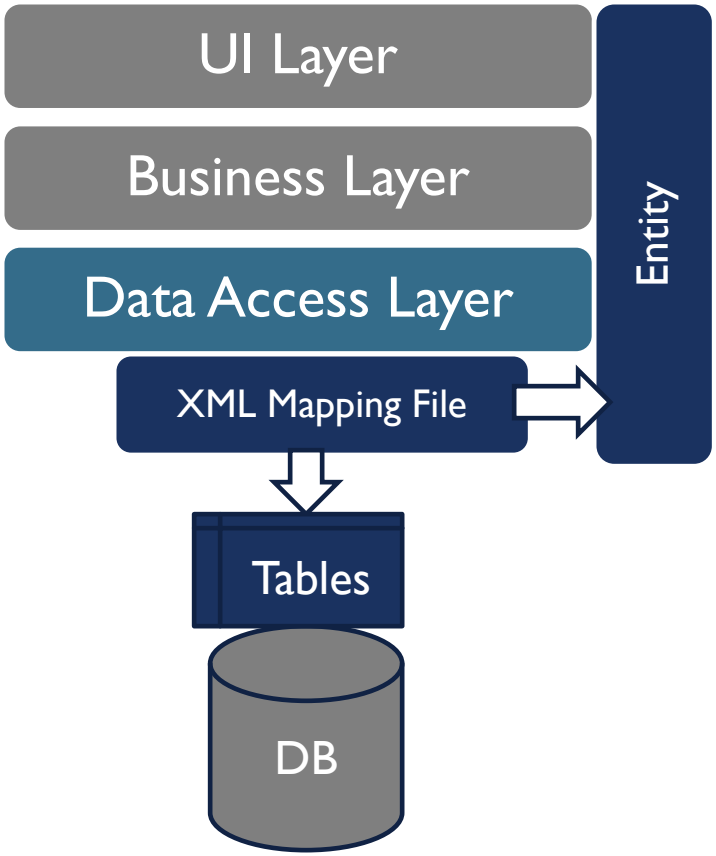
- Maps the *Classes* into the *Tables*
- Maps the *Properties* on Class into Table *Fields*
- Generates dynamic SQL statement.

Comparison

- Traditional Approach

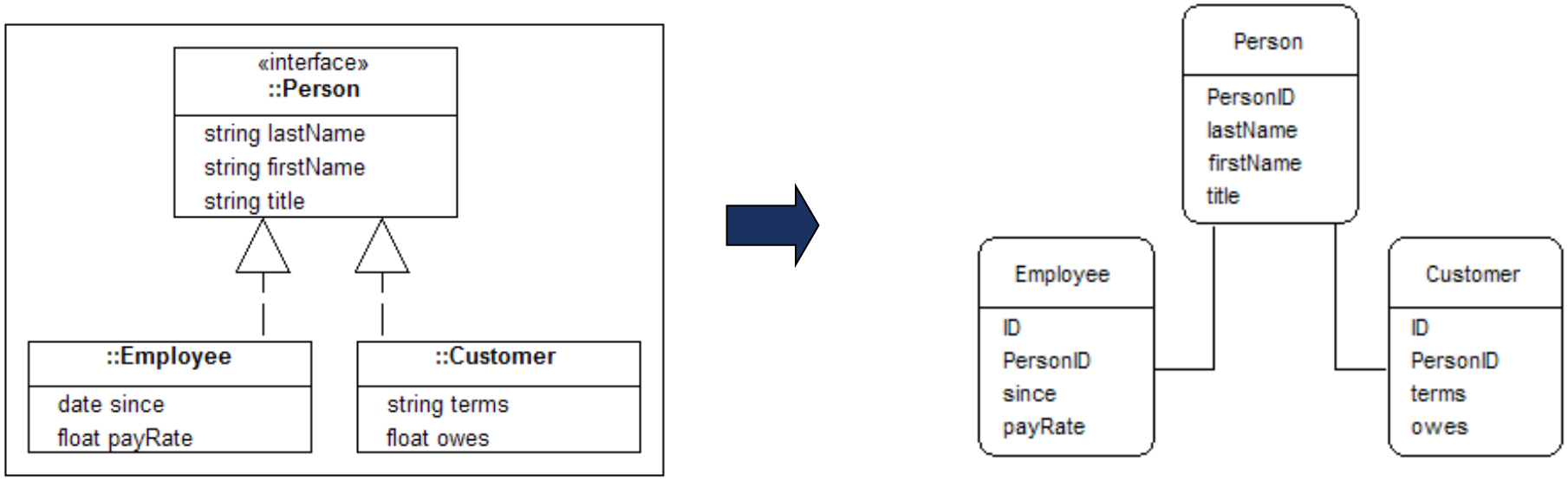


- OR Mapping Approach



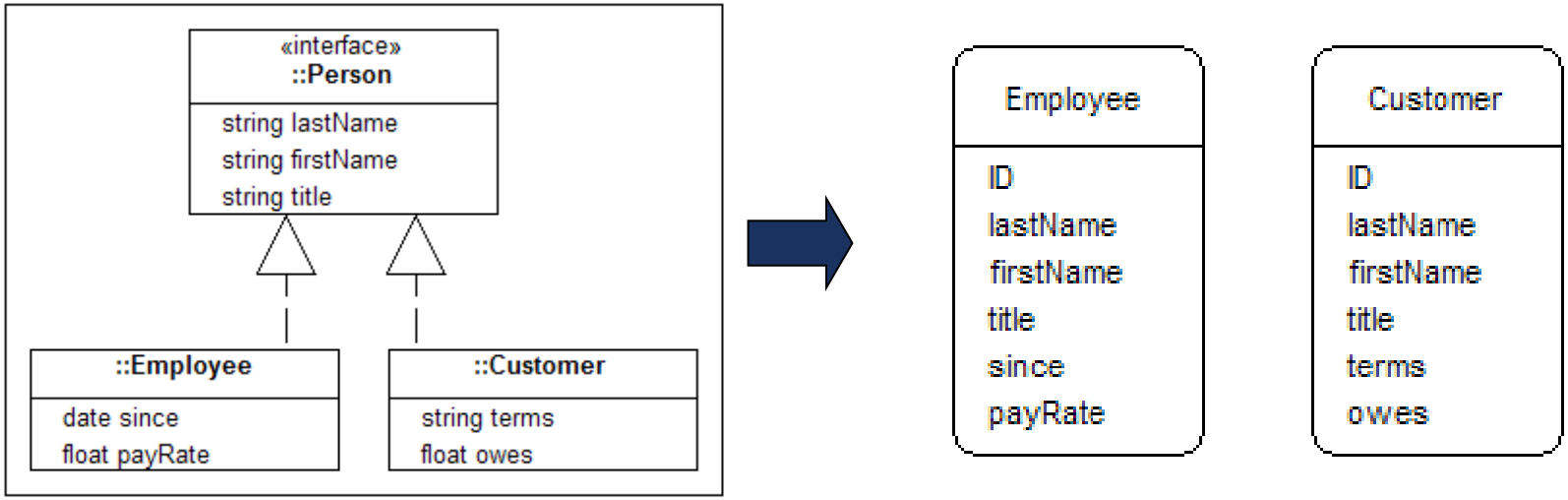
OR Mapping: Inheritance

- Vertical mapping



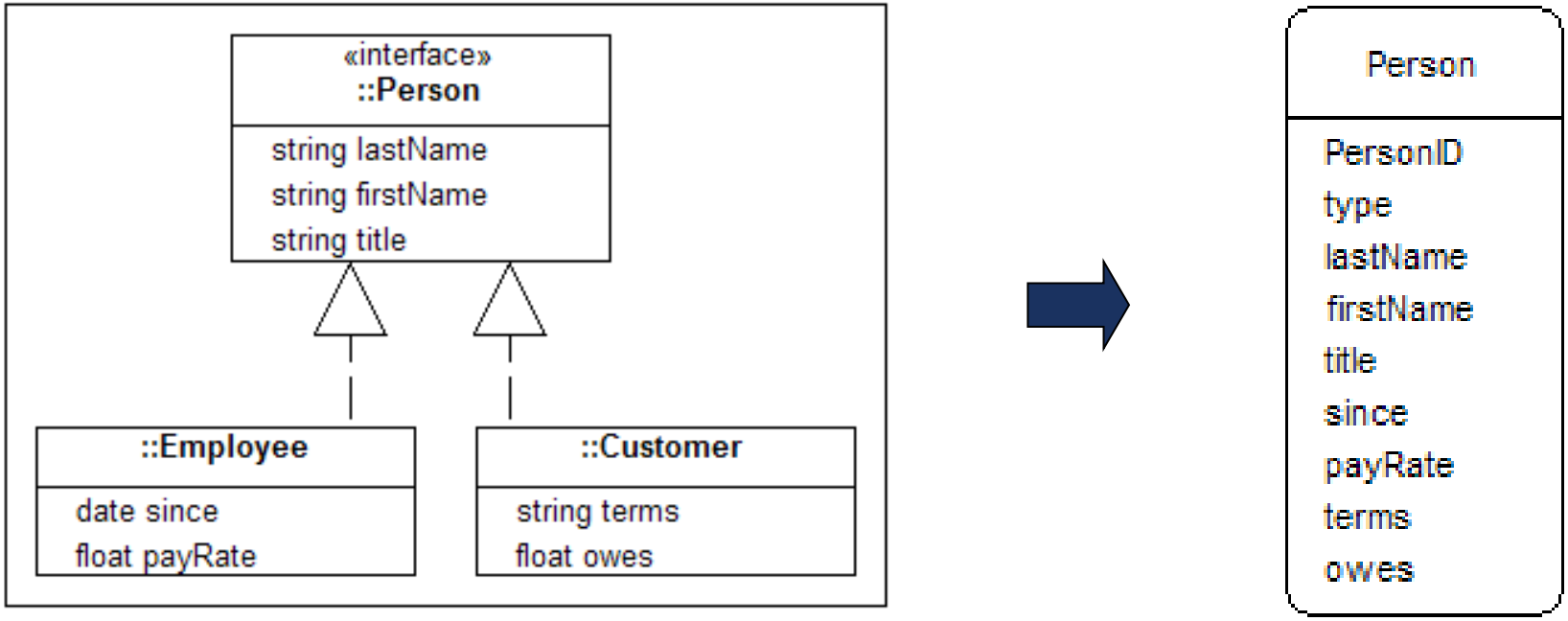
OR Mapping: Inheritance

- Horizontal mapping



OR Mapping: Inheritance

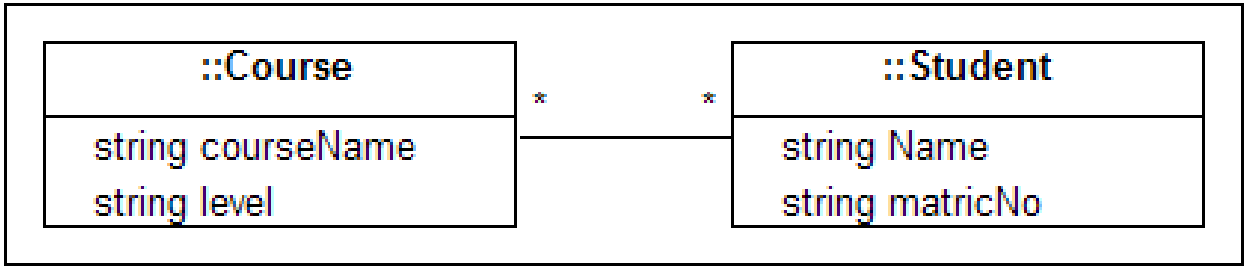
- Filtered mapping



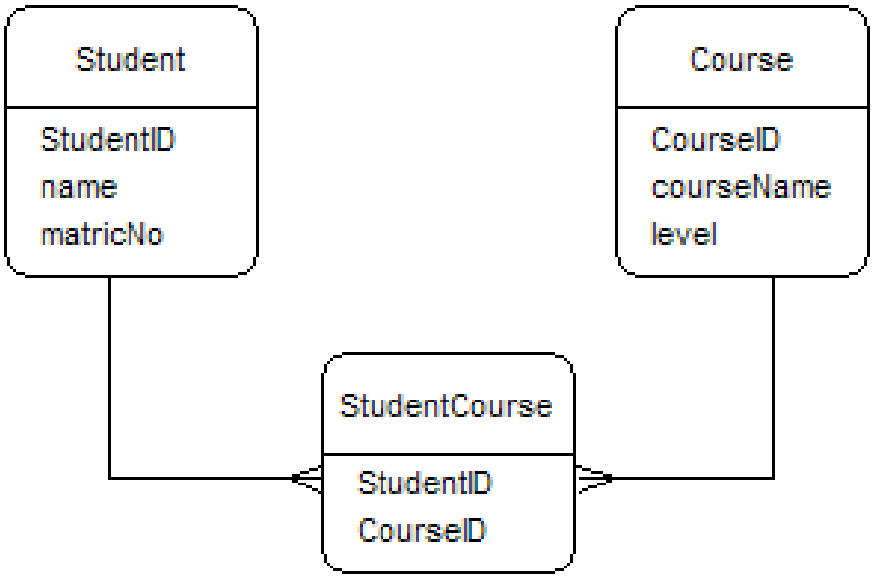
OR Mapping: Guidelines

- Use **Vertical** mapping when:
  - there is significant overlap between types
  - changing types is common
- Use **Horizontal** mapping when:
  - there is little overlap between types
  - changing types is uncommon
- Use **Filtered** mapping for:
  - simple or shallow hierarchies with little overlap between types

OR Mapping: Many-to-Many



In a relational database a join table is required to represent this relationship:



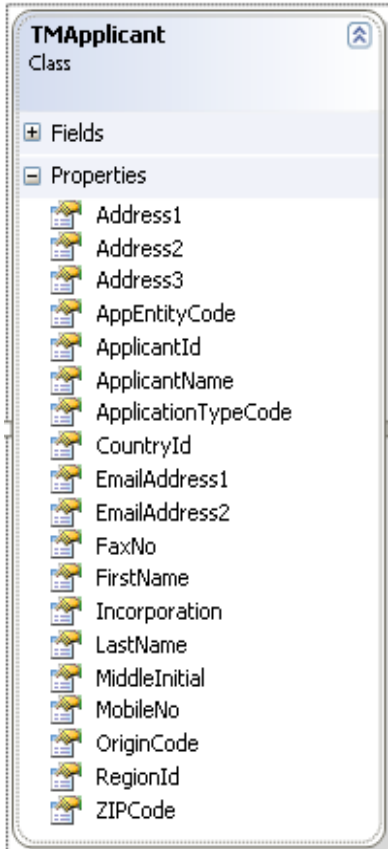
O/R Mapping Tools for .NET

- nHibernate
- Codus
- ORM Framework
- Opf3
- DADO Object Mapper
- eXpress Persistent Objects for .NET
- Itec Data Layer
- Data Mapper
- Objectz.NET
- My Generation
- Nolics.NET

- Wilson ORMapper for .NET
- BizBlox
- EdgeXtend for C#
- OPF.NET
- AgileStudio
- LLBLGen Pro
- Genome
- NDO
- Versant Open Access for the Microsoft .NET Framework
- DataBlock
- DataObjects.NET

Nhibernate: Sample O/R Mapping

- Entity



- XML Mapping

```
<?xml version="1.0" encoding="utf8" ?>
<hibernatemapping xmlns="urn:nhibernatemapping2.2">
 <class name="IPOTeams.TPR.TMApplicant, IPOTeams.TPR" table="dbo.tmapplicant" >
  <id name="ApplicantId" column="ApplicantID_ky"  type="string">
    <generator class="assigned"/>
  </id>

  <property name="ApplicantName" column="Applicant_tx" type="string" length="200"/>
  <property name="FirstName" column="Fname_tx" type="string" length="100"/>
  <property name="LastName" column="Lname_tx" type="string" length="100"/>
  <property name="MiddleInitial" column="Mi_tx" type="string" length="1"/>
  <property name="ApplicationTypeCode" column="AppType_cd" type="string" length="2"/>
  <property name="Address2" column="Address2_tx" type="string" length="100"/>
  <property name="Address3" column="Address3_tx" type="string" length="100"/>
  <property name="CountryId" column="CountryID_fk" type="string" length="2"/>
  <property name="RegionId" column="RegionID_fk" type="string" length="2"/>
  <property name="ZIPCode" column="ZIP_cd" type="string" length="5"/>
  <property name="AppEntityCode" column="AppEntity_cd" type="string" length="1"/>
  <property name="Incorporation" column="Incor_tx" type="string" length="255"/>

  <property name="OriginCode" column="Origin_cd" type="string" length="1"/>
  <property name="FaxNo" column="FaxNo_tx" type="string"/>
  <property name="MobileNo" column="MobileNo_tx" type="string"/>
  <property name="EmailAddress1" column="EMailAdd1_tx" type="string"/>
  <property name="EmailAddress2" column="EMailAdd2_tx" type="string"/>
</class>
</hibernatemapping>
```

Nhibernate: Sample O/R Mapping

- Table

- XML Mapping

```
<?xml version="1.0" encoding="utf8" ?>
<hibernatemapping xmlns="urn:nhibernatemapping2.2">
 <class name="IPOTeams.TPR.TMApplicant, IPOTeams.TPR" table="dbo.tmapplicant" >
  <id name="ApplicantId" column="ApplicantID_ky"  type="string">
    <generator class="assigned"/>
  </id>

  <property name="ApplicantName" column="Applicant_tx" type="string" length="200"/>
  <property name="FirstName" column="Fname_tx" type="string" length="100"/>
  <property name="LastName" column="Lname_tx" type="string" length="100"/>
  <property name="MiddleInitial" column="Mi_tx" type="string" length="1"/>
  <property name="ApplicationTypeCode" column="AppType_cd" type="string" length="2"/>
  <property name="Address2" column="Address2_tx" type="string" length="100"/>
  <property name="Address3" column="Address3_tx" type="string" length="100"/>
  <property name="CountryId" column="CountryID_fk" type="string" length="2"/>
  <property name="RegionId" column="RegionID_fk" type="string" length="2"/>
  <property name="ZIPCode" column="ZIP_cd" type="string" length="5"/>
  <property name="AppEntityCode" column="AppEntity_cd" type="string" length="1"/>
  <property name="Incorporation" column="Incor_tx" type="string" length="255"/>

  <property name="OriginCode" column="Origin_cd" type="string" length="1"/>
  <property name="FaxNo" column="FaxNo_tx" type="string"/>
  <property name="MobileNo" column="MobileNo_tx" type="string"/>
  <property name="EmailAddress1" column="EMailAdd1_tx" type="string"/>
  <property name="EmailAddress2" column="EMailAdd2_tx" type="string"/>
</class>
</hibernatemapping>
```

Sample Implementation Code

- Nhibernate

```
public class TMApplicantDb
{
    protected readonly ISessionManager sessionManager;
    public virtual void Save(TMApplicant persistedObject)
    {
    this.session = this.sessionManager.OpenSession();
        try
      {
      transaction =   this.session.BeginTransaction();
      this.session.Save(persistedObject);
      transaction.Commit();
      }
      catch (Exception ex)
      {

      }
    }
}
```

- ADO.NET

```
public class TMApplicantDb
{

 public virtual void Save(TMApplicant persistedObject)
 {
 string connStr = "Data Source=localhost; User ID=tprAppsUser;Initial
Catalog=dbTEAMSApps;password=tpr01;";
        string SQLInsert = "INSERT INTO tmApplicant (ApplicantID_ky,
Applicant_tx) VALUES(@ApplicantID_ky,@Applicant_tx);";

        //set object
       SqlParameter[] paramIns = new SqlParameter[2];

        paramIns[0] = new SqlParameter("@ApplicantID_ky",SqlDbType.VarChar);
        paramIns[0].Value = persistedObject. ApplicantId
        paramIns[1] = new SqlParameter("@Applicant_tx", SqlDbType.VarChar);
        paramIns[1].Value = ApplicantName .ApplicantId.;

        using (SqlConnection conn = new SqlConnection(connStr))
        {
           conn.Open();
           using (SqlCommand cmd = new SqlCommand(SQLInsert, conn))
           {
              foreach(SqlParameter prm in paramIns)
              {
                 cmd.Parameters.Add(prm);
              }
              cmd.ExecuteNonQuery();

           }
           conn.Close();
        }
 }
}
```

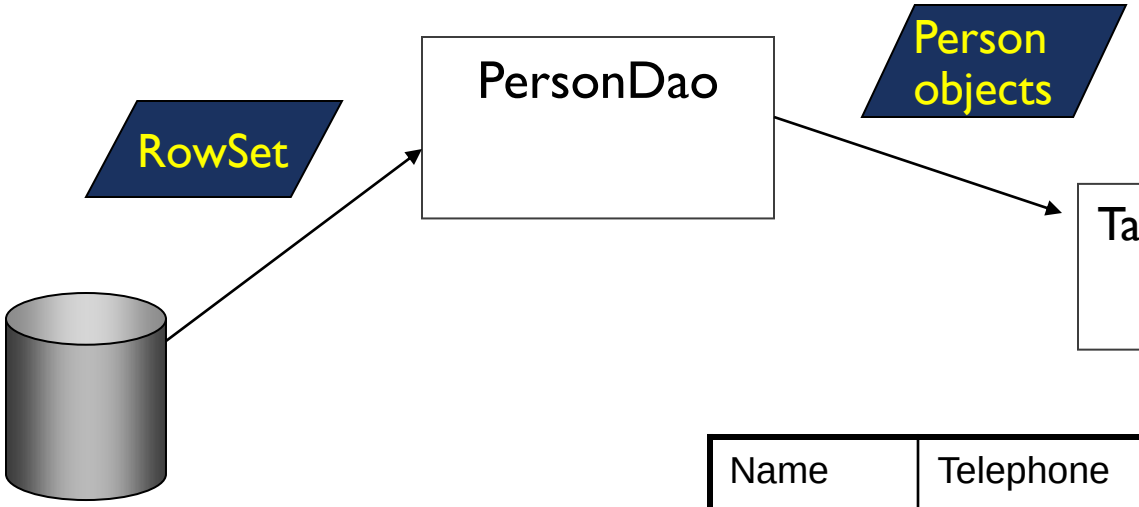
Pros and Cons

- Pros
  - Reduce the amount of code
  - The software is more robust (the fewer the lines of code in a program, the fewer the errors contained within them)
  - Data Access Layer is more portable
- Cons
  - Store Procedure may have better performance
  - Some O/R mapping tools do not perform well during bulk deletions of data

When *Not* to Use O-R Mapping

In some applications, Object-Relational mapping is inefficient.

Example: display a table of attendees



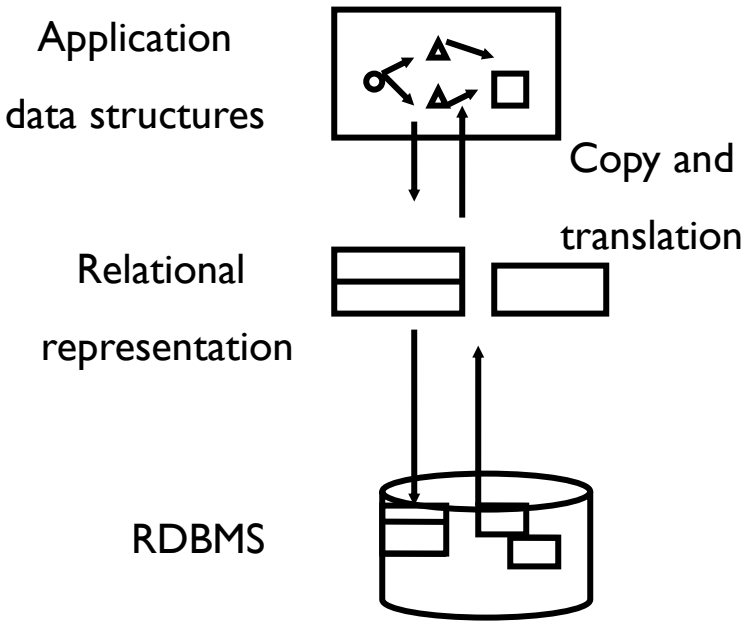
Name	Telephone	Email
Bill Gates	1-215-555-1212	gates@msn.com
B. Obama	1-212-111-1212	president@whitehouse

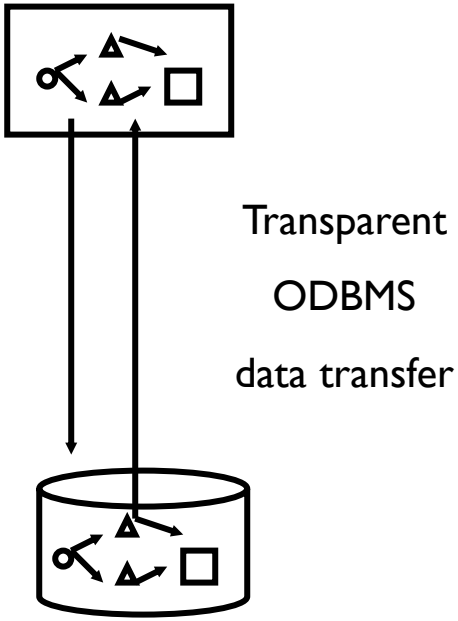
OBJECT ORIENTED DATABASES

What is Object Oriented Database?

- A database system that incorporates all the important object-oriented concepts
- Some additional features
  - Unique Object identifiers
  - Persistent object handling

Motivation of ODBMSs





Why Object-Oriented DBMS?

- Abstract data types
- Interface with host programming language (solve impedance mismatch).
- Object identity:
  - (peter, 40, {(john, 15, {})})
  - (susan, 41, {(john, 15, {})})
  - Same son? Maybe, maybe not.
- Managing large number of objects:
  - Encapsulation, reusability, modularity, protection, less distinction between program and data, type extensibility
- Semantic networks: the world is actually a complex hierarchy (types versus classes).

OODBMS Products

- Versant
- ObjectStore and PSE Pro from eXcelon
- Objectivity/DB
- Intersystems Cache
- POET fastObjects
- **db4o**
- Computer Associates Jasmine
- GemStone

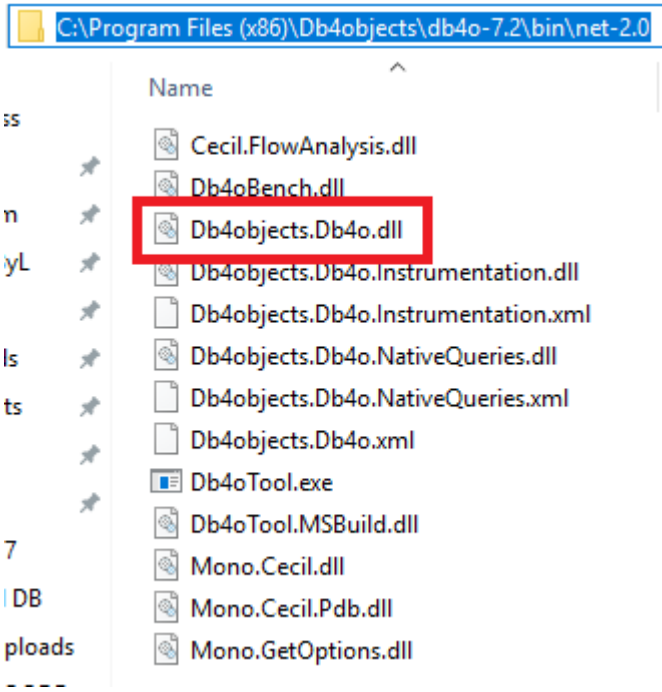
DB4O Concepts

- Object Container
- Querying
- Transaction
- Object Identity
- Inheritance
- Indexing

Getting started with db4o (.NET)

- Db4o engine packed in single DLL
  - /db4o-7.2/bin/net-2.0/Db4objects.Db4o.dll
  - Important classes:
    - Db4objects.Db4o.Db4oFactory: starting point
      - Open db file,
      - Start, connect to server
    - Db4objects.Db4o.IObjectContainer

      → the db4o database



IObjectContainer

- Simple and straightforward interface to object persistence
- IObjectContainer is your db4o database
  - .NET: IObjectContainer db = Db4oFactory.OpenFile(filename);
  - Java: ObjectContainer container = Db4o.openFile(filename)
- All the basic functionality to work with persistent objects
  - save a new or updated object of any class using ObjectContainer#set(object)
  - Deletion is done with => Java: container.delete(object)

Object Container - Storing Objects (.NET)

```
IObjectContainer db = Db4oFactory.OpenFile("first.db");
try
{
            // storeFirstPilot
            Pilot pilot1 = new Pilot("Michael Schumacher", 100);
            db.Store(pilot1);

            // storeSecondPilot
            Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
            db.Store(pilot2);
 }
 finally
 {
            db.Close();
 }
```

Object Container - Storing Objects (Java)

```
public static void storePilot()
{
        new File(YAPFILENAME).delete();
        ObjectContainer db=Db4o.openFile(YAPFILENAME);
        try
        {
                        Pilot pilot=new Pilot("Michael Schumacher",0);
                        db.set(pilot);
                        System.out.println("Stored "+pilot);
                        // change pilot and resave updated
                        pilot.addPoints(10);
                        db.set(pilot);
                        System.out.println("Stored "+pilot);
             }
            finally
            {
             db.close();
            }
        }
}
```

Updating & Deleting

- Update: Store(object) method

```
IObjectSet result = db.QueryByExample(new Pilot("Michael Schumacher", 0));
Pilot found = (Pilot)result.Next();
found.AddPoints(11);
db.Store(found);
```

- Delete: use Delete(object) method

```
IObjectSet result = db.QueryByExample(new Pilot("Michael Schumacher", 0));
Pilot found = (Pilot)result.Next();
db.Delete(found);
```

Querying

- db4o supplies three querying systems
  - Query-By-Example (QBE),
  - Native Queries (NQ),
  - SODA Query API.

Querying - QBE

- you provide db4o with a template object.
- db4o will return all objects matching all non-default field values

```
Pilot proto = new Pilot("Michael Schumacher", 0);
IObjectSet result = db.QueryByExample(proto);
ListResult(result);
```

Microsoft .NET

```
Pilot proto=new Pilot("Michael Schumacher",0);
ObjectSet result=db.get(proto);
listResult(result);
```

Java

Querying - QBE

- QBE has some obvious limitations:
  - db4o must reflect all members of your example object.
  - Not support advanced query expressions: AND, OR, NOT, etc.
  - Cannot constrain on values like 0 (integers), "" (empty strings), or nulls (reference types) because they would be interpreted as unconstrained.
  - Rely on class constructor
    - You need a constructor to create objects without initialized fields.

Querying - NQ

- main db4o query interface
- recommended way to query databases from your application
  - simply use the semantics of your programming language
  - perfectly standardized
  - safe choice for the future
- your provide the ability to run one or more lines of code against all instances of a class

Querying - NQ

- db4o will attempt to optimize native query expressions and run them against indexes

```
IList<Pilot> pilots = db.Query<Pilot>(delegate (Pilot pilot) {
                    return pilot.Points == 100;
});
```

Microsoft .NET

```
List <Pilot> pilots = db.query(new Predicate<Pilot>()
{
      public boolean match(Pilot pilot)
      {
            return pilot.getPoints() == 100;
      }
});
```

Java

More complex NQ

```
IList<Pilot> result = db.Query<Pilot>(delegate (Pilot pilot) {
                    return pilot.Points > 99 && pilot.Points < 199
                      || pilot.Name == "Rubens Barrichello";
});
```

Querying – SODA Query API

- db4o's low level querying API
- allowing direct access to nodes of query graphs
- SODA uses strings to identify fields
  - Not perfectly typesafe
  - compile-time not checked
  - quite verbose to write
- For most applications Native Queries will be the better querying interface

Querying – SODA Query API

- how our familiar QBE queries are expressed with SODA

```
IQuery query = db.Query();
query.Constrain(typeof(Pilot));
query.Descend("_name").OrderAscending();
IObjectSet result = query.Execute();
```


```
Query query=db.query();
query.constrain(Pilot.class);
query.Descend("_name").OrderAscending();
ObjectSet result=query.execute();
```

Transaction

- All work within db4o ObjectContainer is transactional
- Implicitly started when you open a container
- Implicitly committed when you close it again.
- ACID transaction model
- Data transaction journaling
  - zero data loss in case of system failure
  - automatic data recovery after system failure
- db4o core is thread-safe for simultaneous operations

Transaction – Commit

- choose to make a commit explicit or you may leave it for close() call

```
IObjectContainer db = Db4oFactory.OpenFile("first.db");
try
{
    Pilot pilot = new Pilot("Rubens Barrichello", 99);
    Car car = new Car("BMW");
    car.Pilot = pilot;
    db.Store(car);
    db.Commit();

    IObjectSet result = db.QueryByExample(typeof(Car));
    ListResult(result);
}
finally
{
    db.Close();
}
```

Transaction – Commit

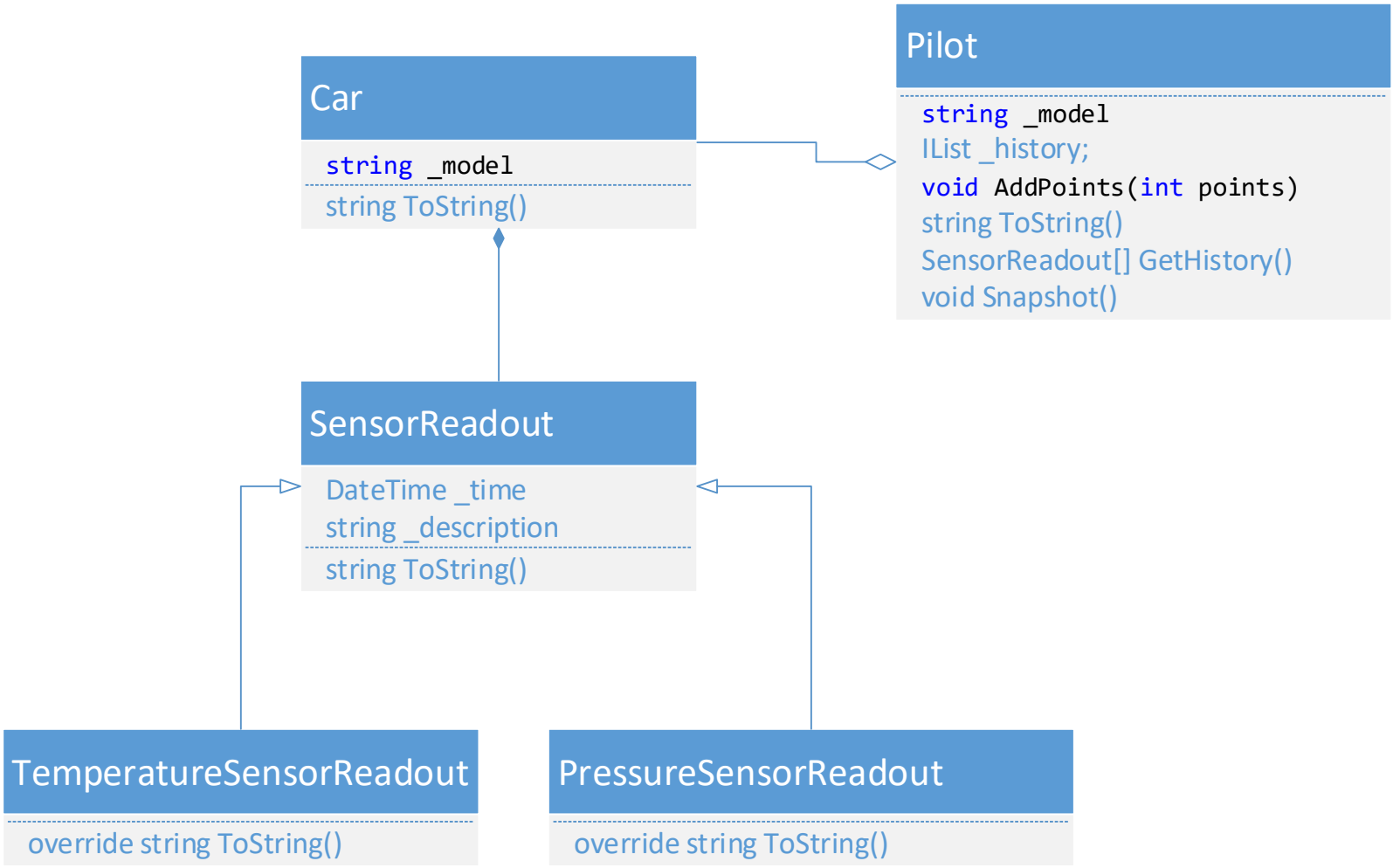
```
IObjectContainer db = Db4oFactory.OpenFile("first.db");
try
{
    Pilot pilot = new Pilot("Rubens Barrichello", 99);
    Car car = new Car("BMW");
    car.Pilot = pilot;
    db.Store(car);
    db.Rollback();

    IObjectSet result = db.QueryByExample(typeof(Car));
    ListResult(result);
}
finally
{
    db.Close();
}
```

Object Identity

- Db4o keeps references to all persistent objects whether they were retrieved, created or activated in this session
- The main role of the reference system
  - to provide access to the required data with the best speed
  - lowest memory consumption
- Performance and usability of the reference system depend much on how the system manages objects identities

Inheritance



Inheritance

- Sensors are added to the database (two times)

```
public void Snapshot()
{
    _history.Add(new TemperatureSensorReadout(DateTime.Now, this, "oil", PollOilTemperature()));
    _history.Add(new TemperatureSensorReadout(DateTime.Now, this, "water",
PollWaterTemperature()));
    _history.Add(new PressureSensorReadout(DateTime.Now, this, "oil", PollOilPressure()));
}
```

- RetrieveTemperatureReadouts by QBE

```
SensorReadout proto = new TemperatureSensorReadout(DateTime.MinValue, null, null, 0.0);
IObjectSet result = db.QueryByExample(proto);
ListResult(result);
```

- Retrieve all sensors by QBE

```
SensorReadout proto = new SensorReadout(DateTime.MinValue, null, null);
IObjectSet result = db.QueryByExample(proto);
ListResult(result);
```

Inheritance

- Output:

```
4
BMW[Rubens Barrichello/99]/6 : Tue Jan 23 23:32:01 CET 2007 : oil
temp : 0.0

BMW[Rubens Barrichello/99]/6 : Tue Jan 23 23:32:01 CET 2007 : water
temp : 0.2

BMW[Rubens Barrichello/99]/6 : Tue Jan 23 23:32:01 CET 2007 : oil
temp : 0.30000000000000004

BMW[Rubens Barrichello/99]/6 : Tue Jan 23 23:32:01 CET 2007 : water
temp : 0.8
```

Indexing

- Allows to index fields
- To request an index to be created by API method called in configuration file

```
class Foo
{
    String bar;
}

Db4oFactory.Configure().ObjectClass(typeof(Foo)).ObjectField("bar").Indexed(true);
```

- Once created will remain in database

Disadvantages of OODBMS

- Schema changes
- Lack of agreed standards
- Lack of ad-hoc querying

- In general, RDBMSs are probably more suitable for databases with a variety of query and user interface requirements (i.e. most mainstream business applications), while OODBMSs are appropriate for applications with complex, irregular data, where data access will follow predictable patterns (e.g CAD/CAM systems, manufacturing databases)

OODBMS Users

- The Chicago Stock Exchange - managing stock trades
- CERN in Switzerland - large scientific data sets
- Radio Computing Services – automating radio stations (library, newsroom, etc)
- Adidas – content for web site and CD-ROM catalogue
- Federal Aviation Authority – passenger and baggage traffic simulation
- Electricite de France – managing overhead power lines