ETH Zürich
Chair of Software Engineering

Java and C# in depth

Carlo A. Furia, Bertrand Meyer

C#: Persistence

Outline

- C# Serialization
- Connecting to a RDBMS with ADO.NET
- LINQ (Language Integrated Queries)
- NoSQL Solutions for C# and Java

C# Serialization

.NET serialization

Binary serialization

- To a stream; disk; memory; network
- Mark the class with the `Serializable` attribute
- Mark the fields you don't want to serialize with the `NonSerialized` attribute

XML serialization

- Serializes only public fields and properties
- Used e.g. for Web Services (Data Contracts, describing the data to be exchanged)

Binary serialization restrictions

Differently from Java, the `Serializable` attribute only applies to one class.

If we want descendant and supplier classes to be serializable, we have to mark them as `Serializable` as well

As in Java, constructors are not invoked at deserialization time, so inconsistent objects may be accepted into the system

As in Java, the class private attributes are part of the class default serialized form (API)

.NET binary serialization example

```
[Serializable]
class ClassA
{
    public int Field1 {set; get;}
    public ClassB Field2 {set; get;}
    [NonSerialized]
    private string field3;
    ...
    }
```

Sample serialization

```
class ClientClass
{ //suppose we pass an object of type ClassA
   public void serialize(Object target) {
   //exception handling omitted
     IFormatter frmt = new BinaryFormatter();
     Stream str= new FileStream("fileName",
     FileMode.Create, FileAccess.Write,
     FileShare.None);
     frmt.Serialize(str, target);
     str.Close();
   }
}
```

Sample deserialization

```
class ClientClass
{//exception handling omitted
    public Object deserialize() {
      IFormatter frmt = new BinaryFormatter();
      Stream str= new FileStream(“fileName”,
      FileMode.Open, FileAccess.Read,
      FileShare.Read);
      Object o = frmt.Deserialize(str);
      str.close();
      return o;
    }}
```

Providing a custom serialized form 1

1. Implement `ISerializable`

2. Implement callback, to be used at serialization time

```
public virtual void
GetObjectData(SerializationInfo inf,
                           StreamingContext sc)
{
      inf.addValue(“key1”,Field1);
      inf.addValue(“key2”,Field2);
}
```

Providing a custom serialized form 2

3. Provide a specific constructor intended for deserialization

```
protected ClassA(SerializationInfo inf,
  StreamingContext sc){
      Field1 = inf.GetInt32(“key1”);
      Field2 = (ClassB)inf.GetValue
      (“key2”, typeof(ClassB));
  }
```

Serialization and Security

- Serialization allows access to **private** fields
- To protect the **GetObjectData** method you should use the **SecurityPermissionAttribute** with two flags:

```
[SecurityPermissionAttribute(
            SecurityAction.Demand,
            SerializationFormatter=true)]
  public virtual void
  GetObjectData(SerializationInfo inf,
                           StreamingContext sc)
{ ...}
```

- Default: local machine can access private fields, intranet or internet-downloaded code cannot
- Hint: don't serialize sensitive information

Providing a custom serialized form 3

- Objects are reconstructed from the inside-out: deserialization order of complex object structures matters.
- Implement `IDeserializationCallback` to do something after the standard (non-XML) deserialization process happened
- method `OnDeserialization (Object sender)` will be called after deserialization
- The functionality for `sender` (the object that initiated the callback) is not currently implemented

Deserialization and Schema Evolution 1

- Version Tolerant Serialization (VTS) is a set of features enabled for `Serializable` classes

- An added attribute in a new version does not cause an exception anymore (is this really an improvement?)

- A field can be marked with the `[OptionalField]` attribute so that it does not trigger an exception if not present when deserializing

`OnSerializing`, `OnSerialized`, `OnDeserializing`, `OnDeserialized` attributes are also available to tag methods doing something before and after (de)serialization

Deserialization and Schema Evolution 2

Among others, Microsoft suggests as “best practices”:

- Never remove a serialized field
- Never change the name or type of a serialized field
- When adding a new field, apply the `[OptionalField]` attribute

Basically, try to evolve your classes as little as possible

Connecting to a RDBMS with ADO.NET

ADO.NET

- Library to interact with data sources (databases, text files, Excel spreadsheets, XML files)
- We are now focusing on relational databases
- There are different Data Providers (API data provider names (DPN) listed)
  - Odbc
  - OleDb (Excel, Access)
  - Oracle
  - Sql (Microsoft SQL Server)
  - Bdp (generic access)

ADO.NET Objects

- `DPNConnection` (i.e. `OdbcConnection`, `OleDbConnection`, `SqlConnection`,…)
- `DPNCommand` (send SQL statements to db)
- `DPNDataReader` (result of a query as a read-only sequential stream)
- `DataSet` (in-memory representation of relational data)
- `DPNDataAdapter` (defined for each table in a `DataSet`, can work offline, load and persist in single batches)

Handling a connection

```
SqlConnection con = new SqlConnection
(“Data Source=myServerAddress;
  Initial Catalog=myDBName;
  User ID=anID;
  Password=aPassword”);

con.Open()
```

- Here a connection is established, if possible

ADO.NET connection pooling

SQL Server can create many connection pools

- One per distinct connection string
- One per Windows Identity (if integrated security is used)
- One per process
- One per application domain

Database operations

A **DPNCommand** object encapsulates a db operation

```
SqlCommand cmd = new SQLCommand(queryString,
   con)
```

For query results use **DPNDataReader**

```
SqlDataReader rdr = cmd.ExecuteReader();
```

For inserts, updates and deletes create first a **DPNCommand** passing an SQL string, and then use

```
cmd.ExecuteNonQuery();
```

Handling query result sets

The **DPNDataReader** object iterates through the result

- Method **`Read()`** moves the cursor to the next row, returning false when there are no more rows
- Can use a string indexer to retrieve column values
  `string name=(string)rdr[“Person_Name”]`
- Can also use getters methods to get column values of the type specified in the getter name (e.g. **`GetDateTime`**)

Cleaning up

Always remember to close the connection in a finally block, even if you are using some connection pool facilities

```
…
finally{
// If you have read data
if (rdr!=null) {
rdr.Close();}
// Very important!!!
if (con!=null) {
con.Close();} } ...
```

Prepared statements

Encapsulated, pre-compiled queries

- More readable, more portable
- Favor query optimization

```
SqlCommand cmd = new SqlCommand ("select *
   from Courses where year = @Year", con);

SqlParameter param = new SqlParameter();
   param.ParameterName = "@Year";

param.Value = courseYear;
cmd.Parameters.add(param);
rdr = cmd.executeReader();
```

Spring.NET

Spring.NET supports ADO.NET enterprise applications

- Connection strings and parameter management
- Provider-independent exceptions, mapped to db-specific errors
- Classes like `AdoTemplate` to handle the core workflow
- Developers only need to implement callback interfaces to handle the mapping

LINQ
(Language Integrated Queries)

Object-relational impedance mismatch

- Object oriented applications and relational databases implement different mathematical models
- A mapping layer is therefore necessary to represent objects as tables and vice versa
- Mapping objects to tables may be difficult and error prone
- From here stems the idea of ORM's and data mappers

Language Integrated Queries (LINQ)

Access data from different sources in a unified manner

Express data access logic in C#, enabling execution in a completely different environment

Reduce the impedance mismatch between data models, e.g.

- LINQ to objects
- LINQ to SQL
- LINQ to XML

Sequences

An unbounded data structure providing one element at the time

- `DataReader` (sequence of rows from a db)
- `Stream` (sequence of bytes)
- `TextReader` (sequence of characters)

`IEnumerable<T>` interface models sequences

Useful to think of query expressions as sequences

LINQ to objects: a sample query

```
IEnumerable<string> customerNames =
from customer in customers //step 1: source
where customer.Data > 200000 //step 2: filter
select customer.name; // step 3: projection
```

It starts with the empty sequence and then transforms it into other sequences according to the following steps obtaining:

1. A sequence containing all customers in `customers`
2. A sequence containing customers satisfying 1. having the value of field Data > 200000
3. A sequence satisfying 1. and 2. containing the customer’s names

Query result

Id	Name	Country	Data
123	Reto	Switzerland	100000
132	Cecilia	Italy	300000
213	Hauke	Germany	200000
231	Nadia	Russia	400000
312	Yu	China	500000
321	Viswanathan	India	600000

Query basic structure

```
from rangeVariable dataSource
where filter-expression
select projection-expression;
```

`rangeVariable` is an identifier with an optional type name: used to refer to data consistently within the query

`dataSource` is a source of a sequence of data

`filter-expression` there can be many **`where`** in conjunction

`projection-expression`

Ordering data

```
from rangeVariable dataSource
where filter-expression
orderby ordering-expression orderingKind
select projection-expression;
```

`orderingKind` can be `descending` or omitted (ascending)

There can be multiple pairs `ordering-expression` – `orderingKind`, separated by comma. Each one is interpreted in relation to the previous (e.g.: order by ascending id and, within each id, by descending age)

LINQ to SQL

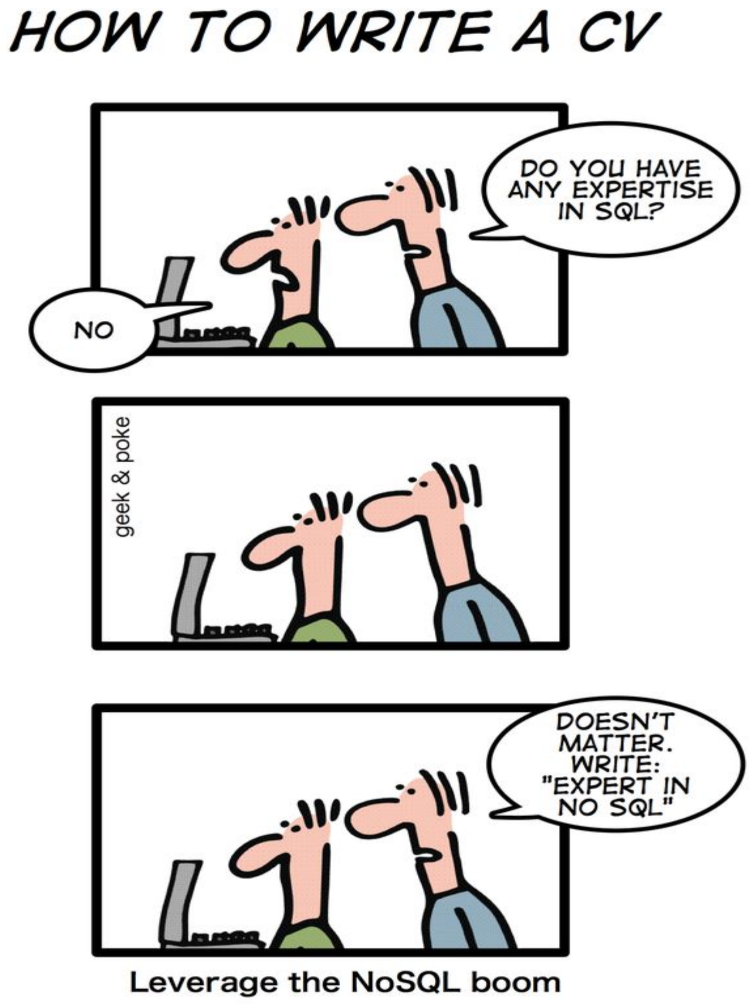
Queries are consistent with LINQ to Objects

The compiler helps validating the queries

Only supports SQLServer

A good ORM solution (but always check the generated SQL!)

NoSQL Solutions for C# and Java


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

What’s wrong with Relational DBs?

- Established solution
- Great with read-heavy web sites
- Scale horizontally by adding web servers
- Scale vertically by adding cores/CPUs/RAM to support higher DB traffic on the (only) DB instance
- Data have structure (schema) and are validated against it
- ACID transactions
- SQL is (mostly) a standard

Web 2.0 a.k.a. Social Web

Poses new challenges that relational DBs don’t address well:

- New kinds of traffic profiles (e.g. massive, “viral” variations in traffic)
- Switch from read-heavy to balanced read-write web sites
- Queries involving the absence of data
- Large quantities of text and images

Not Only SQL

- Movement began in 2009, now becoming a commodity
- “Non-relational databases” would be more appropriate
    - Key-Value
    - Wide Column Store (HBase, Cassandra, SimpleDB)
    - Document
- And also XML DBs, OODBs, Graph DBs, Multi-model

Commonalities of NoSQL solutions

- Scale horizontally very well (no RAM sharing) for large number of simple read/write operations (Web 2.0)
- (Mostly) No fixed schema, no standard query language
- ACID within a node, eventually consistent across cluster
- Terminology is generally inconsistent, but
  - All NoSQL systems store scalars (integers, strings), BLOBs, attribute-value pairs
  - Some NoSQL systems store data structures (tuples, documents, objects)

Key-Value Data Stores

- Basically hash tables
- Examples: Redis, RaptorDB…
- Applied successfully for distributed content caching
- Offer insert, delete, and index lookup
- Use Map Reduce (Google patent) for indexing and searching
    - Map: extract sets of key-value pairs
    - Reduce: merge and sort pairs to show results

Document Data Stores

- Store documents made up of tagged elements
- Allow as values scalars, lists, other nested documents
- Secondary indexes, simple query mechanisms
- No schema (attribute names defined at runtime)
- Examples: CoachDB, MongoDb,…
- Used successfully for Web 2.0 applications