Advanced Programming

Final Term Paper

Start Date: 8/01/2014
Submission deadline: 14/02/2014 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone's code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary. **Do not include in an exercise code only needed for a later exercise**.
4. cite references to literature or Web pages from where information was taken.

Introduction

In this project, you will develop a DSL for expressing ORM mappings, similar to JPA (Java Persistence API) or Doctrine2. The PAORM language provides an XML notation for defining entities and their mappings to database tables. Here is an example:

```
<entity class="Book">
   <attributes>
      <id name="id">
      </id>
      <basic name="title" type="String">
          <column length="80" />
      </basic>

      <many-to-one name="publisher" target-entity="Publisher" />
```

```
    </attributes>
</entity>

<entity class="Publisher">
    <attributes>
        <id name="id" type="String">
            <column length="80"/>
        </id>
        <one-to-many name="books" target-entity="Book"
                     mapped-by="publisher"/
    </attributes>
</entities>
```

The PAORM generator will produce both class definitions and a SQL schema like these:

```
public class Book {
    public integer id;
    public String title;
    public Publisher publisher;
}

public class Publisher {
    public String name;
    public List<Books> books;
}
```

Exercise 1

Design a set of classes to represent PAORM entity definitions.

Exercise 2

Implement a recursive descent parser for PAORM, using if needed a streaming XML reader like XmlTextReader or XmlStreamReader. Ensure that each parser method returns a proper object representing the input it has analyzed.

Exercise 3

Write two generators for producing respectively the code for the entities and the SQL schema. Ensure to use polymorphism in the implementation of the generator.

Exercise 4

Design and implement an `IEntityManager` class, providing the following interface:

```
interface IEntityManager<T> {
    void persist(T entity);
    void remove(T entity);
    T find(Object primaryKey);
}
```

Method `persist()` serializes the entity and inserts it in the appropriate tables. For example, an instance of `Book` must store a row in table `Book` as well as one in the in the `Publisher` table corresponding to the book publisher attribute object. Method `remove()` instead will just perform the removal of the row in the Book table.
Method `find()` should retrieve and deserialize the object `Book` with the given key as well as the corresponding `Publisher` object from the `Publisher` table.
In order to access the database the `IEntityManager` should use suitable SQL queries: write a generator for such queries.

Exercise 5

Extend the `IEntityManager` interface to include the following method:

```
    Query createQuery(String query);
```

where parameter query is a string expressing a SQL query and the `Query` implements this interface:

```
interface IQuery<T> {
   List<T> getResultList();
   void execute();
}
```

The method `getResultList()` performs the query and "hydrates" (i.e. deserializes into objects) the result set into a list of objects.

The method `execute()` is used to perform update or delete SQL queries.

Exercise 6

Explain what is an Object-Relational Mapping and discuss and compare Linq and JPA as techniques to facilitate access to data in databases. In particular compare the expressiveness of logical operator available in Linq with respect to thos of SQL based ORMs.

Exercise 7 (for students of Laurea Specialistica in Tecnologie Informatiche)

Consider the addition of mapping between two tables through a join table, for example a table `BooksAuthors` to model the many-to-many relationship between authors and books:

```
<entity class="Book">
   <attributes>
   …
      <many-to-many name="BooksAuthors"
                  join-column="author"
                  inverse-join-column="book" />
   </attributes>
</entity>
```

The generator should produce a new table definition like this:

```
create table BooksAuthors (
    id integer,
    book integer,
    author integer,
    primary key (id)
);

alter table BooksAuthors
    add constraint FK33F1
    foreign key (book)
    references Book;

alter table BooksAuthors
    add constraint FK33F2
    foreign key (author)
    references Author;
```

Extend the parser and the generators to deal with this notation.