Università degli Studi di Pisa

Master Degree in Informatics

Advanced Programming

Final Term Paper

Start Date: 8/09/2017
Submission deadline: 28/09/2017 (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 system of code annotations. The annotation are denoted with an at sign (@) followed by the annotation name, followed by a list of key-value pairs. Here is an example:

```
@Table(name="book")
public interface Book {
    @Id(name="id")
    Integer id;
    @Column(name="title", length="80")
    String title;
    @Many2One(name="publisher", target="Publisher")
    Publisher publisher;
}
```

```
@Table(name=”publisher”)
public interface Publisher {
    @Id(name="id")
    Integer id;
    @Column(name="name", length="80")
    String name;
    @One2Many(name="books", target="Book",
                    mappedBy="publisher")
   List<Book> books;
}
```

Exercise 1

Design a set of classes to represent annotated code.

Exercise 2

Implement a recursive descent parser for annotated interface declarations. Ensure that each parser method returns a proper object representing the input it has analyzed.

Exercise 3

Write a code generator for producing plain code for each interface without the. Write a generator that produces a corresponding SQL schema for each interface.
Ensure to use polymorphism whenever possible.

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.
Show the SQL queries produced for the example in the introduction.

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.
Show an example of the generated SQL query.

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 those of SQL based ORMs.