IS 523/SM 503 Assignment #3

Due Date: December 10, Tuesday 22:30

In this and the following assignment you are expected to develop a limited functionality version of the Inventory Management System (IMS). In this version of the system, there will be only one user: Store Manager. The use case diagram of the system to be developed and brief descriptions of the use cases are given in the following pages. The system will be developed in two iterations (each iteration constitutes an assignment). The requirements of the first iteration are as follows:

1) "Manage Products" and "Manage Suppliers" use cases will be realized.

a) Write these **Use Cases** in fully-dressed format.
b) Bounded by these use cases' scenarios Draw a **Domain Model** using the UML class diagram notation.
c) Draw **System Sequence Diagram(s)** for the main success scenario of each of these use cases.
d) Draw UML **Interaction Diagrams** for the design of the system realizing these use cases. Annotate every message with the hint GRASP and/or other pattern that justifies it.
e) Bounded by the interaction diagrams you have drawn in part (c) draw a **Design Class Diagram** in UML notation.
f) Realize your design **in an object-oriented programming language**.
g) You need to persist your objects. In order to **persist objects**, you may use the approach explained in the following pages.
h) You can implement **a text-based UI** for your application.

2) Following artifacts should be uploaded to ODTUCLASS assignments link by the due date:

a) Use Cases in a fully dressed format
b) Domain Model
c) System Sequence Diagram(s)
d) Interaction Diagrams
e) Design Class Diagram
f) Source Code
g) Executable Program

Notes:

- You are expected to **work in groups of 2 members.** The assignment should be submitted by only one member of the group.
- You have to adhere to the university's policies of academic integrity.
- The penalty for late submission is 15% per day.
- The solution should be submitted in a **single zip** file via the assignment link in ODTUCLASS.
- If you have any questions, please do not hesitate to contact me at gmert@metu.edu.tr

Inventory Management System

- The use case diagram of the system is given in the figure below.

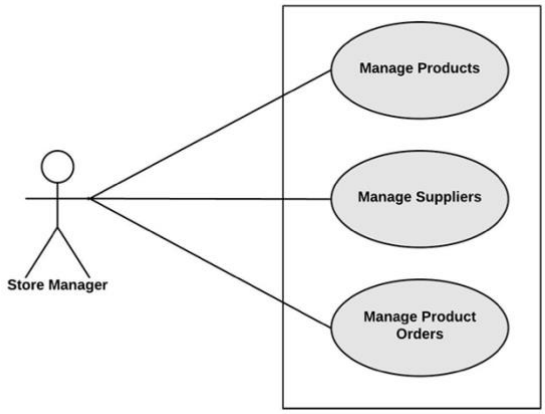

Figure 1. Use Case Diagram of IMS Project

Note: In this and the following assignment, we assume that Store Manager is responsible for Manage Products, Manage Suppliers, and Manage Product Orders use-cases.

- Brief descriptions of these use cases (CRUD Operations) are as follows;
  - **Manage Products:** Store Manager defines/updates/deletes/lists the products. For each request, the following data may be captured:
    - Product ID
    - Description
    - Maximum stock level
    - Minimum stock level
    - Current stock level
  - **Manage Suppliers:** Store Manager can define/update/delete/list suppliers. For each supplier, the following information can be remembered;
    - Supplier ID
    - Product ID
    - Name
  - **Manager Product Orders:** Store Manager can manager orders by defining/updating/deleting/listing orders. The following information should be remembered for each order:
    - Product ID
    - Supplier ID
    - Quantity
    - Order Date

    The order quantity should be checked according to the current stock level and the maximum stock level of the related product such that;

    $$quantity_{ordered} \leq \ (\text{maximum}_{\text{stock\_level}} - current_{stock\_level})$$

Persistency

- You may use the following mechanism for persistence or you may implement your own simple persistence mechanism based on binary/text files. Please note that, you

are **not allowed to use any third-party database management systems and tools**.

- In order to persist objects (save/retrieve objects), you can create a root object and put (i.e., associate) all objects requiring persistence into that object.
- When the program starts, load the root object from the configuration file using an object input stream. Before program terminates, save the root object to the configuration file using an object output stream.
- A sample java program that illustrates the approach is given below. If you prefer using Java as a programming language, you can use the provided code-snippet in your implementation.

```
import java.util.List;

/**
 * This is an example root, it contains all other objects. * Do not forget to write 'implements
java.io.Serializable'.
 */ public class Root implements java.io.Serializable {
    /**
     * Keep each type of object (that needs persistence) in a separate container...
     */
    List<Other> otherObjects;
    public Root() { // default constructor
    }
    public void aMethod() {
    }
 }
```

```
/**
 * This is an example entity class. * Do not forget to write 'implements java.io.Serializable'.*/
 public class Other implements java.io.Serializable {
    String data;
    public Other() { // default constructor
    }
    public void writeData(){
       System.out.println(data);
    }
 }
```

```
public class Main {
   public static final String fileName = "objects.dat";
   public static Root root;
   public static void main (String[] args) throws Exception {
      loadRoot();
      // initialize and run your application here....
      // do not forget to call saveRoot() before exiting!!!
   }
   public static void loadRoot() throws Exception {
      java.io.ObjectInputStream stream = new java.io.ObjectInputStream(new
java.io.FileInputStream(fileName));
      try {
         root = (Root) stream.readObject();
      } finally {
         stream.close();
```

```
    }
  }
  public static void saveRoot() throws Exception {
    java.io.ObjectOutputStream stream = new java.io.ObjectOutputStream(new
java.io.FileOutputStream(fileName));
    try {
      stream.writeObject(root);
    } finally {
      stream.close();
    }
  }
}
```