Software Architecture Document

Version 1.0

for

A1 SOEN 423

Prepared by

Charles-Antoine Hardy	27417888	m.hardy.inc@gmail.com

Instructor:	Shivaraj Mallikarjun Alagond
Course:	SOEN 423
Date:	October 2018

Document history

Date	Version	Description	Author

Table of contents

1. Introduction

This document is describing the architecture of a SOEN 423 Assignment.

Purpose

This document shall provide a basic understanding of the architecture.

Class Diagram

I will not provide one large class diagram as it will be very confusing. Instead, multiple small class diagram has been put together to explain the software.

Model

The model contains object classes. The main classes described in the requirement: Record, Employee, Manager, Project, Location. The class *PortConfiguration* is used to save the configuration of the server.

The Manager and Employee are direct children of the class Record. Each class has its own override for: 'toString()' , 'Equals' and 'hasCode()'. It's allowing us to write objects in the log file with 'toString()' and to compare objects easily.

(As the image is too large for a document, I've added it to the project Git repository.)
https://trello-attachments.s3.amazonaws.com/5ba8189ab9f30563a9787c4c/5bc91b7e608b241c4297a8a2/2e4b0e2790821ae0fbabab621eaf8044/modelPackage.gif

Making the *Employee* and *Manager* being children of *Record* is allowing to store List of records with manager and employee in the list.

Core

The core architecture of the RMI software is represented by this diagram

The *ServerLauncher* class is launching the server component while the *client* class is using the registry.

The *ServerConfigurator* is creating the 3 servers. For each location (ENUM), the configurator is creating an *IStore* instance, the instance is inserted in a *IHRAction* instance which is also inserted in the UDP server and in the registry. By doing so, the UDP server and registry are united.

As you can see, in the diagram the *IHRAction* contains an instance of *IStore* and the *ServerUDP* also contains an instance of *IStore*. It is allowing the UDP server to write into the log file of the server.

Each *IStore* instance is configured with a *storeName*. The name is the location name (“US”, “UK”, “CA”) The name is used to create a folder or find the folder linked to the server. Each server has 3 default storage files created: **Log.txt, Record.txt,** and **Project.txt**.

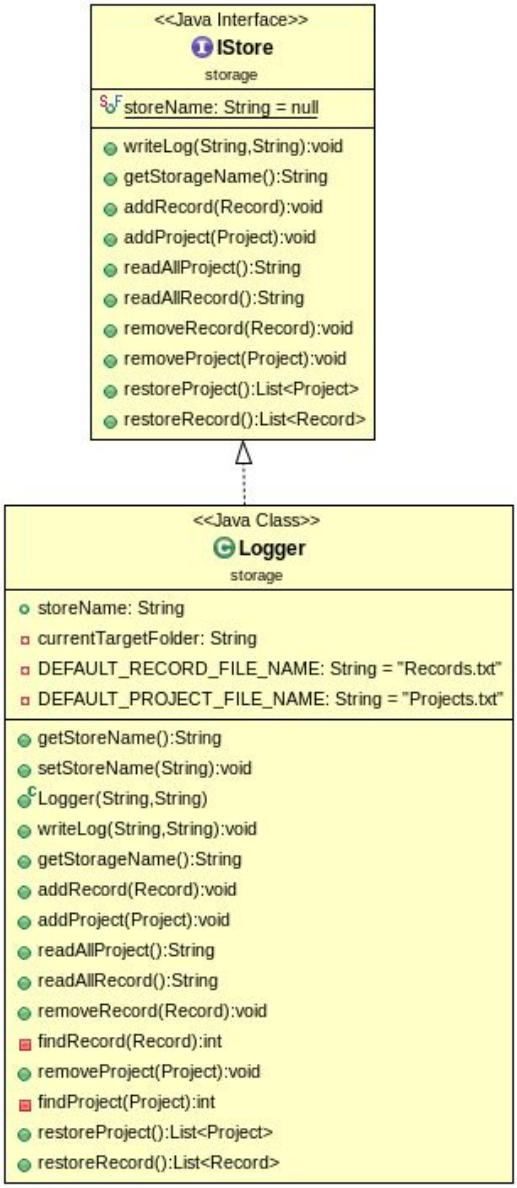
ServerConfigurator - Code Snippet:

```
private void buildCenter(Location loca, int port) {

    // Create and pass a storing engine to the HRAction
    IStore storingEngine = new Logger(loca.toString(), MAIN_TREE_FOLDER + loca.toString() + "/");
    int udpPort = port + 1;
    try {
        HRActions instanceHRAction = new HRActions(storingEngine);
        Thread thread = new Thread(new ServerUDP(instanceHRAction, udpPort));
        thread.start();
        // Update configuration of the server
        HashMap<Location, Integer> serverConf = PortConfiguration.getConfig();
        serverConf.put(loca, port);
        PortConfiguration.updateConfig(serverConf);
        Registry reg = LocateRegistry.createRegistry(port);
        reg.bind(loca.toString(), instanceHRAction);

    }catch(Exception ee) {
        ee.printStackTrace();
    }

    String startingMessage = "The Server: " + loca.toString() +
            " is on port:  " + port;
    String udpStartingMessage = "The UDP Server for: " + loca.toString() +
            " is start on port " + udpPort;
    System.out.println(startingMessage);
    System.out.println(udpStartingMessage);
    configStoring.writeLog(startingMessage, "CentralRepo.txt");

}
```

This is where the ‘magic’ happen. Each registry is bind to a *instanceHRAction* and each UDP server is bound to the same instance. It’s allowing UDP Server thread to communicate with current *HRAction* data. (e.g. HashMap<Integer, List<Record>>)

Storage

The storage is made of two class *IStore* (interface) and the implementation *Logger*. The storage is only made to insert/update/read/delete on a text file.



RMI Interface

The interface has been implemented *IHRAction* and *HRAction.* Few helper methods were added to increase readability.

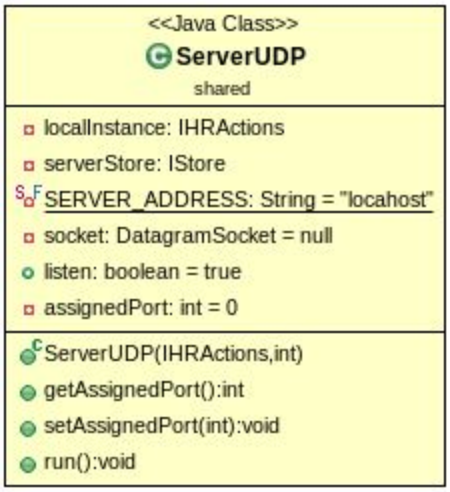

<<Java Interface>>
IHRActions
shared
store: IStore = null
createMRecord(String,String,String,String,String,String):String
createERecord(String,String,String,String,String):String
getRecordCount():String
editRecord(String,String,Object):boolean
createProject(String,String,String):String
getLocalNumberOfRecords():byte[]
<<Java Class>>
HRActions
shared
serialVersionUID: long = 1L
DEFAULT_LOG_FILE: String = "Log.txt"
SERVER_ADDRESS: String = "localhost"
db: Map<Integer,ArrayList<Record>>
dbProject: List<Project>
currentRecordID: List<String>
currentProjectID: List<String>
currentManagerID: List<String>
store: IStore
HRActions(IStore)
restoreFromStorage():void
buildfakeDatabase():void
createMRecord(String,String,String,String,String,String):String
generateUniqueManagerId(Location):String
createERecord(String,String,String,String,String):String
getIndexFirstLetter(String):Integer
emailIsNotValid(String):boolean
getRecordCount():String
getLocalNumberOfRecords():byte[]
getNumberOfRecordsWithServer(int):String
editRecord(String,String,Object):boolean
UpdateProject(Project,String,Object):boolean
UpdateManager(Record,String,Object):boolean
UpdateEmployee(Record,String,Object):boolean
FindProjectWithId(String):Project
FindRecordWithId(String):Record
createProject(String,String,String):String

The method *getRecordCount* is counting the local record and then the other records. It is using the *PortConfiguration* object to get to the other UDP server. Once on the other UDP server, since they have an instance of the local HRAction, the UDP server simply call *localInstance.getLocalNumberOfRecords()*

Another complex method is the *editRecord* the method has to find out if the target is a project, manager or employee then it has to make sure it exists, finally it is updating the value with a add/delete operation in storage.

UDP Server

As discussed, the UDP server is running in a separated thread. The server has an instance of its local HRAction called *localInstance,* it also has an instance of local *IStore* called *serverStore.*



Repository

The project is being built in a private GitHub repository. (https://github.com/Winterhart/java-rmi-simulation). The A2 will simply be done again on the same repository.