COMP 6231
Aymeric Grail
#26810020
aymeric.grail@gmail.com

1st assignment: Documentation

Loan Management System - RMI

General notes

This documentation is related to the code submitted electronically for the 1st Assignment (via the EAS system).
The code can be found in the same zip file as this documentation electronic version.
1 hard copy has been produced.

The **java Documentation** has been redacted and is available electronically in a folder at the root of the sources folder.
No hard copy has been produced.

Coding philosophy

The provided code intend to follow the principles of "Clean Code"[1]. It includes:

- design by interface
- descriptive names
- short methods (both vertically and horizontally speaking)
- declarative syntax whenever possible (eg: use of the Java 8 Stream library)
- limited comments in exchange of numerous and verbose method names and variables
- regular small refactoring to maintain code readability and S.O.L.I.D. principles[2]

Package

[1] "Clean Code: A Handbook of Agile Software Craftsmanship", 2008, Robert C. Martin.
[2] Wikipedia : https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

The source files contains contained in submitted archive contains the following folders:

- Client: contains the sources for both the customer and manager UI clients (simple consoles).
- Documentation: a folder that contains this current document as well as the generated java documentation (html format). Note that only public methods are being documented in the javadoc as per the “Clean Code” philosophy.
- Logs: contain the log files for both the customer and the manager clients. One file per user.
- LogsServer: contains the server logs. One per server.
- Server: contains the sources for the servers.
- Shared: contains the sources for the code shared between client and server (contracts, etc…)

High level: modules

At a higher level, the code is separated in modules that are relevant to the client, or the server or both (shared).
Then, each module is organized in different layers such as:

Neutral layers:

- Contracts: for holding the different interfaces
- Helpers: for static tools such as a generique console, serializers, …

Higher layers:

- Presentation (client side only): for the UI interaction
- Server (server side only): for API endpoints

Medium layers:

- Services: the business layer providing core functionalities

Lower layers:

- Data: for data entities and projections
- IO: for lower level IO operations
- Transport: for handling communications (RMI, UDP, …)

Medium level: design patterns

Singletons

Singletons have been used for single instances access such as the session service or the locker factory.

Factory

A simple factory has been created to provide access to read and write lock depending on a given value.

Decorator

A simple decorator was used to abstract the access to the console from the customer and manager consoles for cross-platform concerns

---

Lower level: data structures

Collections:

Streamable collections have generally been prefered to regular collections (eg: static arrays) in order to easily manipulate data using streams, such as ArrayList, and HashTable.

HashTable have been used to properly handle concurrency[3] since they implement a fail fast design using ConcurrentModificationException. Those allow to detect when a collection being read is also being modified.

Custom serializable DTO:

Custom data class made serializable have been preferred to native data structure for data transfer. For example, the UDP client use a custom message class to serialize the information needed by the UDP server to answer the request.

Serialization:

A simple *static generic* serializer has been created in order to ensure that object used in UDP communication are serialized and deserialized in a consistent manner.

---

[3] Oracle documentation on hashtable: http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html

Testing

Unit testing:

No unit test framework has been put in place for this assignment, mostly due to a lack of time. A Behavior Driven Development (BDD) testing framework would be the next step for the following assignments if time be.

Testing coverage:

Simple manual tests have been created on server side in order to test in particular the following aspects:

- simple threading support
- read and write concurrency
- UDP communications

At least one of those 3 aspects is tested for each of the 4 functionality provided by the API:

- open account
- get loan
- delay payment
- get customers information

That means that all functionality have at least one test to ensure they're not broken.

Difficulties

Main difficulties: concurrency

The main difficulty of this assignment revolved around threading and concurrency, particularly:

- how to secure access at a finer grain to allow different threads to access a shared data on different indexes
- how to avoid starvation and deadlocks in a readers-writers problem[4].
- how to return a value from a thread (it's been decided not to do it)
- I tried to use many different synchronization solution (synchronized method, block, locks) before implementing a semaphore answer to the readers-writers problem
- debugging concurrency proved to be tedious
- developing concurrency tests took some time (manipulating threads).

[4] Readers-Writers problem on wikipedia:
https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem#Third_readers-writers_problem

Other difficulties:

- learning about java serialization
- UDP communication sometimes won't throw errors, but instead will result in request timeout (in particular when data returned is invalid).
- Designing an architecture for a distributed system and supporting logging operations, transport operations, UI operations, IO operations etc.
- Setup tools in order to run RMI, and UDP server.
- Developing techniques for client-server debugging. For example finding a tool to be able to spawn 3 servers at once.

Source Control

GitHub has been used as a source control. The github repository is publicly accessible at the following address:
https://github.com/erkkiperkele/6231-01-Loan

Process documentation

Checkins have been pushed on a regular basis on github. Checkins comments are public and provide the sole and rather minimal documentation on the process of creation of this assignment.

The main aspect of the process were more or less chronologically:

- setting up higher level modules (presentation, transport, data, etc.)
- trying to setup interfaces first (design by interface)
- then setting up data structures
- include constant small refactoring to try to maintain and refine separation of concerns

Although all those aspects of development were a concern at all times, I am conscious that their resolution can still be greatly improved.
Development process improvement is a constant concern.