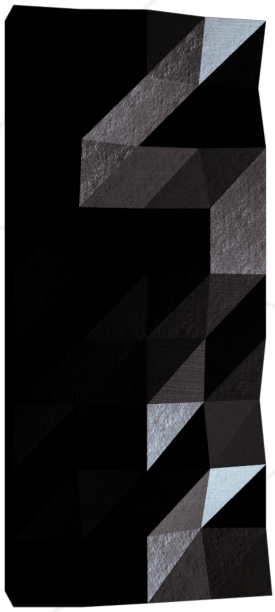

Fixme

Hard network programming

Alex alex@academyplus.ro
42 Staff pedago@42.fr

Summary: This is the last project from the **Java** *world at 42. You will learn to develop multi-threaded network applications with asynchronous sockets and the java executor framework.*

Contents

Chapter I

Forewords

Wall Street is the most famous work of the dadaist artist Marcel Duchamp. Dada, an art form based around the breaking of all previous artistic rules, arose from the stricken post-WWI society of Europe and, to a lesser extent, New York. In Wall Street, Duchamp contradicts our most basic assumptions about walking, societal status, throughfares and bricks by simply signing a brick wall that he came across whilst walking through the City that Never Sleeps. He also set up a sign at the furthermost eastern end of the wall, which simply said "Wall Street, by Marcel Duchamp". To this day, his mastery of the artistic form, and lack of "sembler tout raisonnable foutu", as the French might say, is shown to its best effect in the nihilistic modernistic expressionistic work of Wall Street.

Chapter II

Introduction

This is the last `Java` project, produced by Academy+Plus. Since you already master common Java programming and idioms, we will focus on a complex scenario where threads and sockets will be involved.

Modern computing cannot be conceived without networks of computers communicating with each other. To make this possible you need to understand the TCP protocol, sockets and ports. The problems that arise in peer-to-peer architecture or client-server architecture are numerous and have multiple solutions.

You will have to implement a server component and two client components that exchange messages between each other, over the TCP protocol, ensuring that the messages are well-formed and are delivered correctly.

Chapter III

Goals

One rainy day, you find out about the FIX Protocol and decide that you want to make a lot of money on the stock exchange by using your programming skills and making the computer do the heavy lifting for you. In order to do this you need some hardcore, lightning fast, enterprise grade tools. You will use these tools to simulate electronic trading and experiment with trading algorythms. You will have 3 independent components that will comunicate over the network:

- A market component.
- A broker component.
- A message router.

Some key points need to be met in your project in order to develop a winning solution:

- Use non blocking sockets.
- Use the java executor framework for message handling.
- Multi-module `Maven` build.

Only a good and clear implementation will be accepted. For this to happen, it will have a clean design, will be easy to read and understand by your peers and will be easy to change in case the requirements are modified.

`Chain-of-responsibility pattern`

Chapter IV

General instructions

- You are allowed to use language features up to the latest Java LTS Version included.
- You are are allowed to use any external libraries, build tools or code generators.
- Do not use the default package.
- Create your own relevant packages following the `Java` package naming conventions.
- Java is compiled into an intermediate language. This will generate some .class files. Do not commit them on your repository!
- Make sure you have javac and java available as commands in your terminal.
- Make sure you have the mvn command line tool available, or use one bundled in your IDE.
- Build the project running the command bellow in the root of your project folder. This needs to generate runnable .jar files that can launch each component.

```
$mvn clean package
```

Chapter V

Mandatory part

You need to implement simulation tools for the financial markets that exchange a simplified version of FIX messages. The tools will be able to communicate over a network using the TCP protocol. The focus in this project are not the trading algos (you can experiment with them after the project is finished), but the implementation of a robust and performant messaging platform.

V.1 Router

The router is the central component of your applications. All other components connect to it in order to send messages to other components. The router will perform no buiness logic, it will just dispatch messages to the destination component(s). The router must accept incomming connections from multiple brokers and markets. We call the router a market connetivity provider, because it allows brokers to send messages (in FIX format) to markets, without depending on specific implementation of the market.

The router will listen on 2 ports:

- Port 5000 for messages from Broker components. When a Broker establishes the connection the Router asigns it a unique 6 digit ID and communicates the ID to the Broker.
- Port 5001 for messages from Market components. When a Market establishes the connection the Router asigns it a unique 6 digit ID and communicates the ID to the Market.

Brokers and Markets will include the assigned ID in all messages for identification and the Router will use the ID to create the routing table.

Once the Router receives a message it will perform 3 steps:

- Validate the message based on the checkshum.
- Identify the destination in the routing table.
- Forward the message.

V.2 Broker

The Broker will send two types of messages:

- Buy. - An order where the broker wants to buy an instrument
- Sell. - An order where the broker want to sell an instrument

and will receive from the market messages of the following types:

- Exeuted - when the order was accepted by the market and the action succeeded
- Rejected - when the order could not be met

V.3 Market

A market has a list of instruments that can be traded. When orders are received from brokers the market tries to execute it. If the execution is successfull, it updates the internal instrument list and sends the broker an Executed message. If the order can't be met, the market sends a Rejected message.

The rules by which a market executes orders can be complex and you can play with them. This is why you build the simulator. Some simple rules that you need to respect is that an order can't be executed if the instrument is not traded on the market or if the demanded quantity is not available (in case of Buy orders).

V.4 FIX Messages

All messages will respect the FIX notation.
All messages will start with the ID asigned by the router and will be ended by the checksum.
Buy and Sell messages will have the following mandatory fields:

- Instrument
- Quantity
- Market
- Price

Chapter VI

Bonus part

Bonus points will be given if:

- You store all transactions in a database
- You concieve a fail-over mechanism so that ongoing transactions are restored in case one component goes down.

Chapter VII

Turn-in and peer-evaluation

Turn your work in using your `GiT` repository, as usual. Only work present on your repository will be graded in defense.